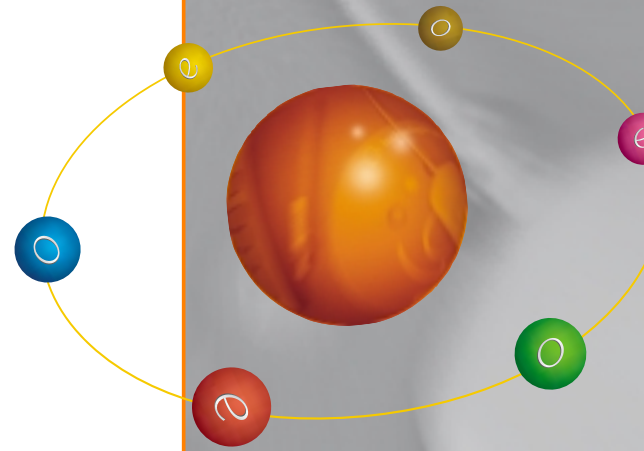


Objecteering/UML

Objecteering/UML Profile Builder
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.

© 2003 Objecteering Software

Objecteering/UML version 5.2.2 - CODOBJ 001/002

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	
Introducing Objecteering/UML Profile Builder	1-3
UML profiling projects	1-7
Glossary	1-9
Chapter 2: First Steps: Creating a module	
Creating a UML profiling project	2-3
Creating a UML Profile	2-6
Referencing a metaclass	2-9
Procedure for entering a J method	2-13
Creating a module.....	2-16
Configuring a module.....	2-22
Testing a UML profiling project.....	2-24
Chapter 3: First Steps: Building a generator	
Creating a generator	3-3
Creating a UML profile.....	3-6
Creating parameters	3-8
Creating note types.....	3-9
Creating tagged value types	3-11
Creating J attributes.....	3-12
Creating a generation work product	3-14
Creating J methods.....	3-18
Implementing J methods.....	3-23
Creating a module.....	3-36
Creating commands.....	3-37
Configuring a module.....	3-39
Testing a module	3-40
Chapter 4: Using Objecteering/UML Profile Builder	
Launching Objecteering/UML Profile Builder	4-3
Creating or opening a UML profiling project.....	4-4
Receiving and upgrading UML profiling projects	4-7
The main window	4-11
The meta-explorer.....	4-13
The properties editor	4-18

Chapter 5: Elements customizing UML and Objecteering/UML	
Modules.....	5-3
Using modules	5-4
Commands.....	5-7
Tagged values	5-10
Notes.....	5-13
Stereotypes.....	5-16
Document and generation templates	5-19
Work products.....	5-22
Other customizable services	5-25
Chapter 6: Defining UML profiles	
Referencing a metaclass	6-3
Creating a type of tagged value.....	6-6
Creating a type of note	6-9
Creating a stereotype.....	6-12
Creating a constraint.....	6-16
Creating a J class attribute	6-18
Creating a J method.....	6-21
Textually editing a J method	6-25
Redefining a J method.....	6-28
Defining a module parameter	6-31
Creating a generation or document template.....	6-34
Defining a new kind of work product	6-47
Chapter 7: Defining modules	
Overview of module definition	7-3
Referencing UML Profiles.....	7-6
Using UML Profiles	7-8
Defining a UML installation profile	7-10
Creating commands.....	7-13
Changing the default values of parameters	7-17
Specializing a module	7-18
Packaging a module	7-20
Chapter 8: Test projects	
Test project definition.....	8-3
Testing J methods.....	8-9
Index	

Chapter 1: Introduction

Introducing Objectteering/UML Profile Builder

Introduction

Welcome to the *Objectteering/UML Profile Builder* user guide!

Objectteering/UML Profile Builder is a unique and powerful tool, used to parameterize Objectteering/UML, in order to adapt it to your specific needs. With *Objectteering/UML Profile Builder*, you can define your own tagged values, documents and document items, as well as checking, generation and validation rules. The *Objectteering/UML Profile Builder* tool is a powerful aid to providing completely project-oriented model-driven and process-driven development.

The *Objectteering/UML Profile Builder* tool is used to design and implement UML profiles, which are predefined sets of stereotypes, tagged values, constraints and notation icons, used to collectively specialize and tailor UML to a specific domain or process. A UML profile does not extend UML by adding new basic concepts, but instead provides conventions for applying standard UML to a particular environment or domain.

UML profiles can, for example, provide UML rules specific to analysis, or dedicated to technical design, as well as those relative to C++ generation or to the generation of database schemas. They define UML usage contexts, in order to adapt the UML to particular needs.

Objectteering/UML Profile Builder uses different workspaces than those of *Objectteering/UML Modeler* UML modeling projects. The information stored in UML profiles is thus managed separately, and represents UML development know-how. Each UML modeling project can select those UML profiles which are of interest, and may thus use specific rules, assistance and automation.

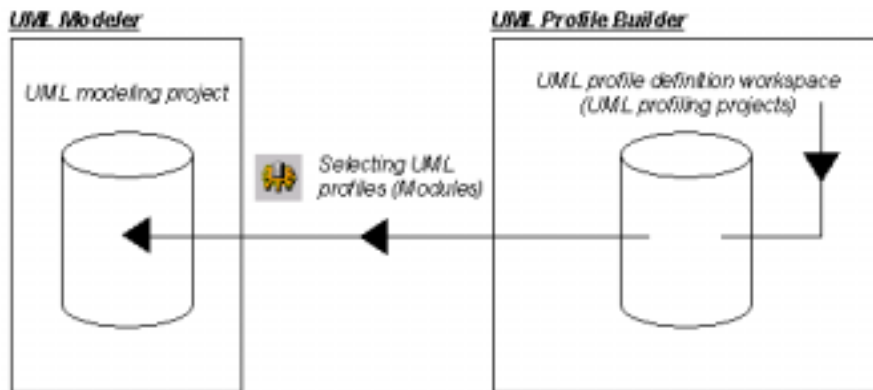


Figure 1-1. The know-how contained in UML profiles is applied to UML modeling projects

For example, *Objectteering/UML Profile Builder* is used to define new code generators, to adapt existing generators, to establish consistency checking rules, to automate design patterns, to carry out inquiries into tool data, to define document templates or generation templates, and so on. *Objectteering/UML Profile Builder* provides a language dedicated to the use of UML by UML profiles, called the *J Language*. J language syntax is close to that of Java.

Functions of the Objectteering/UML Profile Builder tool

Objectteering/UML Profile Builder can also be used to create modules (for example, a module to generate documentation or indeed to generate C++), and to this end offers the following services:

- ◆ the creation of new J services (J methods)
- ◆ definition of parameters, which allow the user to supply module options
- ◆ definition of commands, which are translated into menus in the *Objectteering/UML Modeler*
- ◆ definition of types of notes, types of tagged value and stereotypes
- ◆ definition of generation work products
- ◆ definition of document templates or code generation templates

Objectteering/UML allows the user to edit parameterization environments, which are structured into UML profiling projects. A UML profiling project is used in the same way as a UML modeling project in its modeling phase.

Note: For more information on the *Objectteering/Metamodel* and the *Objectteering/J Libraries*, please refer to the related user guides. Similarly, for further details on the J language, please see the *Objectteering/The J Language* user guide (for details on all available modules, please see the "General Contents" section of the *Objectteering/Introduction* user guide).

Structure

Parameterization elements are structured into UML profiles and modules:

- ◆ a UML profile contains J methods, parameter definitions, types of notes and types of tagged values, document templates and generation work products.
- ◆ the module, which is a high-level entity, references one or several UML profiles. For example, the *Objectteering/C++* module references a C++ code generation UML profile and a C++ makefile generation UML profile. A module contains commands and parameter values, and can be delivered to users in the *Objectteering/UML Modeler*. Users then take advantage of specific menus, parameters and clean generation processing.

The assumption is made that the user of *Objectteering/UML Profile Builder* has some prior knowledge of object modeling.

This user guide makes reference to the following user guides:

- ◆ *Objectteering/The J language* user guide
- ◆ *Objectteering/Metamodel* user guide (based on the UML 1.4 standard).

Working with the Objectteering/UML Profile Builder tool

For the user of the *Objectteering/UML Profile Builder* module, the work consists firstly of organizing parameterization into hierarchical UML profiles. In these UML profiles, the user defines parameters, tagged value types and notes types. He also creates attributes and J methods which can, amongst other things, exploit parameters, notes according to their type and modeling objects according to their tagged values.

The user must then group his UML profiles into modules. The module is the entity used by the standard *Objectteering/UML* user (specifier, designer, etc.). Within these modules, he creates commands, which activate those J methods which belong to the module's UML profiles.

A UML modeling project which uses this modules will now have new menus available. Thus, for each module command, a menu item is added to the contextual menu available for model elements (class, operation, etc), whose type is, or specializes, the metaclass which the J method references.

UML profiling projects

Definition

A UML profiling project is to *Objectteering/UML Profile Builder* what a UML modeling project is to the *Objectteering/UML Modeler*. It is a development environment that contains UML profiles organized into a certain hierarchy. These UML profiles structure tool parameters, J methods and attributes, as well as note and tagged value types, and stereotypes.

A UML profiling project also contains modules which group together UML profiles and commands, as well as providing default values for parameters. For the end user, modules correspond to functional and coherent "packages". (for example, Gen C++, Gen Doc, etc.). Several users can work at the same time on different UML profiling projects.

UML profiles

UML profiles group together a set of J methods and module parameters for a given theme (for example, Java code generation, makefile generation or RDB generation). They organize, in hierarchical order, J rules, which are added to the model's metaclasses.

In a UML profile, it is possible to:

- ◆ create a child UML profile which can then redefine some of the parent UML profile's methods (UML profile generalization)
- ◆ create a reference to a metaclass
- ◆ create a parameter
- ◆ create a type of work product
- ◆ create a generation document template
- ◆ create a document template

Default UML profiles

A UML profiling project is initialized with a certain number of UML profiles. These cannot be modified (Figure 1-2). A UML profile is always created from a parent UML profile.

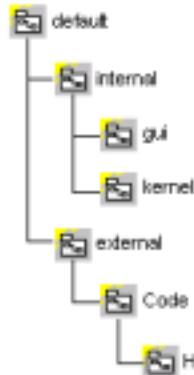


Figure 1-2. Default UML profiles supplied with a UML profiling project

Note: Default UML profiles can neither be destroyed nor modified.

The module

A module is a set of consistent features developed using J methods on the metamodel's classes.

From a module, it is possible to:

- ◆ reference UML profiles
 - ◆ use UML profiles
 - ◆ create a command
 - ◆ specialize the UML profiling project's module
-

Glossary

Command: module component and entry point for a J method. A command will appear in *Objectteering/UML Modeler* as a context menu (pop-up menu) entry for model elements. The J method referenced by the command is defined once the metaclass which is concerned (model elements in *Objectteering/UML Modeler*) contains the command in its context menu.

Document item: description of information that should be included in documentation.

Document template project: hierarchy, composed of "*document items*", describing the typical form of documentation.

Generation item: description of information which should be included in source code.

Generation template project: Hierarchy composed of "*generation classes*" which describe the typical source code form.

J attribute: "class" attribute added to a metaclass, used by the J language.

J method: method defined on a metaclass, in a UML profile, which contains operations used to exploit the metamodel.

Metaclass: metamodel element. It is used to structure J methods, J attributes, note types and tagged value types in a UML profile. "*Class*" or "*Attribute*", for example, are metaclasses.

Metamodel: model used to describe another model. All the elements supported by *Objectteering/UML Modeler* are described in the metamodel.

Module: a "functional and consistent" group of UML profiles and commands.

Module parameters: information entered by the user which has an impact on the execution of J methods.

Module transformation: mechanism that applies transformation rules in order to modify a model. The J language is used to describe these rules.

Note type: definition of a note for the objects of a given metaclass. For example, the "*description*" note type is used to attach "*description*" notes to "*Class*" type objects.

Objecteering/UML Profile Builder: module used to customize the tool's through the addition of new features.

Stereotype: (UML Notion) specific adaptation of *ModelElement* semantics. Through stereotypes, the end user can create new icons and new adaptations of model elements.

Tagged value: (UML term) A tagged value is an annotation of a model element. This term covers note types and tagged value types defined below.

Tagged value type: definition of a tagged value for the objects of a given metaclass. For example, the "*persistent*" tagged value type on the "*Class*" metaclass allows you to annotate a "*Client*" class in a model with the "{*persistent*}" tagged value.

Test project: project allowing you to simultaneously test the modules developed in a UML profiling project.

UML Profile: a UML profile is a way of structuring J methods. A UML profile represents a certain way of viewing a model, for a functional purpose. A metaclass has different J methods according to the current UML profile.

UML profiling project: environment for developing UML profiles and modules.

Work product: reference to one or more work products or deliverables created for a model element. A product appears in the properties editor, and can represent documentation, generated source codes, etc. It manages the external elements of the model and their consistency. It is possible to create new types of work products with *Objecteering/UML Profile Builder*.

Chapter 2: First Steps: Creating a module

Creating a UML profiling project

Introduction

The following example illustrates how to create a module, which will be used to study the impact of the modification of a class in *Objecteering/UML Modeler*. You must work in an existing UML modeling project, since this will allow you to immediately test work carried out in the UML profiling project.

In the example, you are going to create the "NewUMLProfilingProject" UML profiling project, the "CalculateImpact" module and the "default#external#Impact" UML profile.

Creating a UML profiling project

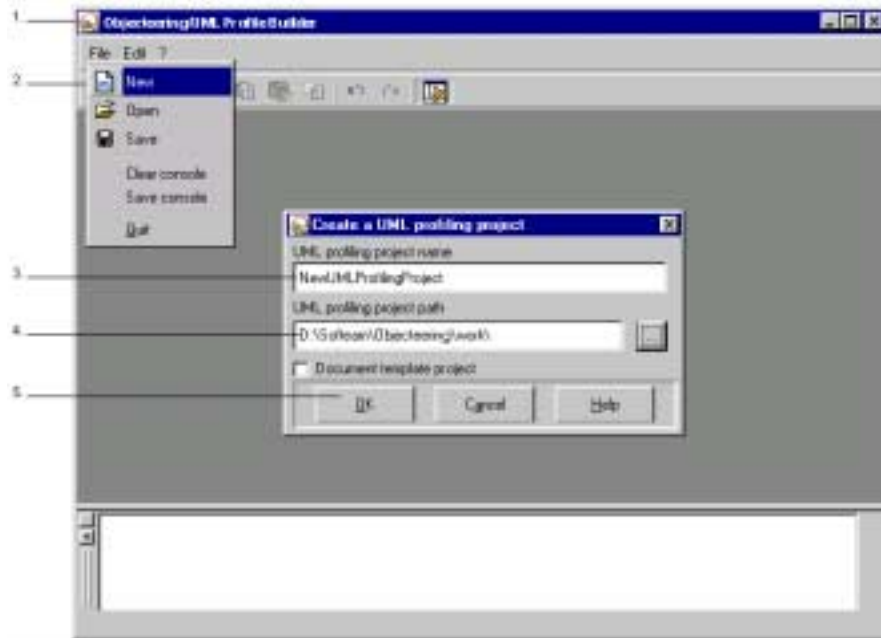




Figure 2-1. Creating a UML profiling project

Steps:

- 1 - Click on the  *Objecteering/UML Profile Builder* icon in your desktop. The window shown in Figure 2-1 will then appear.
- 2 - Click on the "File/New" menu. The "Create a UML profiling project" window will then open.
- 3 - In the "UML profiling project name" field, enter the "NewUMLProfilingProject" name.
- 4 - In the "UML profiling project path" field, enter the path of the directory where the new UML profiling project is to be created. You may also use the  icon to open a file browser through which you can select your UML profiling project path.
- 5 - Confirm by clicking on the "OK" button.

Note: Document templates are created by checking the "Document template project" tickbox.

Creating a UML Profile

Creating a child UML Profile

The first step in our example is to create an "Impact" UML profile in the UML profiling project's default UML profile (Figure 2-2).

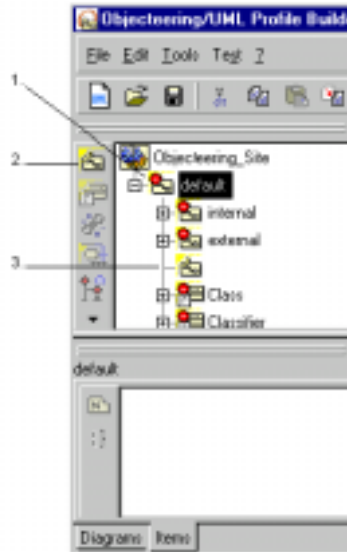



Figure 2-2. Creating the "Impact" UML Profile

Steps:

- 1 - Expand the UML profiling project and select the "default" UML profile.
- 2 - Click on the  "Create a child UML profile" button.
- 3 - Enter the name directly in the explorer and confirm by left-clicking.

Note: If you press the "Return" key on your keyboard, a new UML profile will be created (through the continuous entry creation mode).

Creating a module parameter

We are now going to create the "Scan the child classes" module parameter in the "Impact" UML profile.

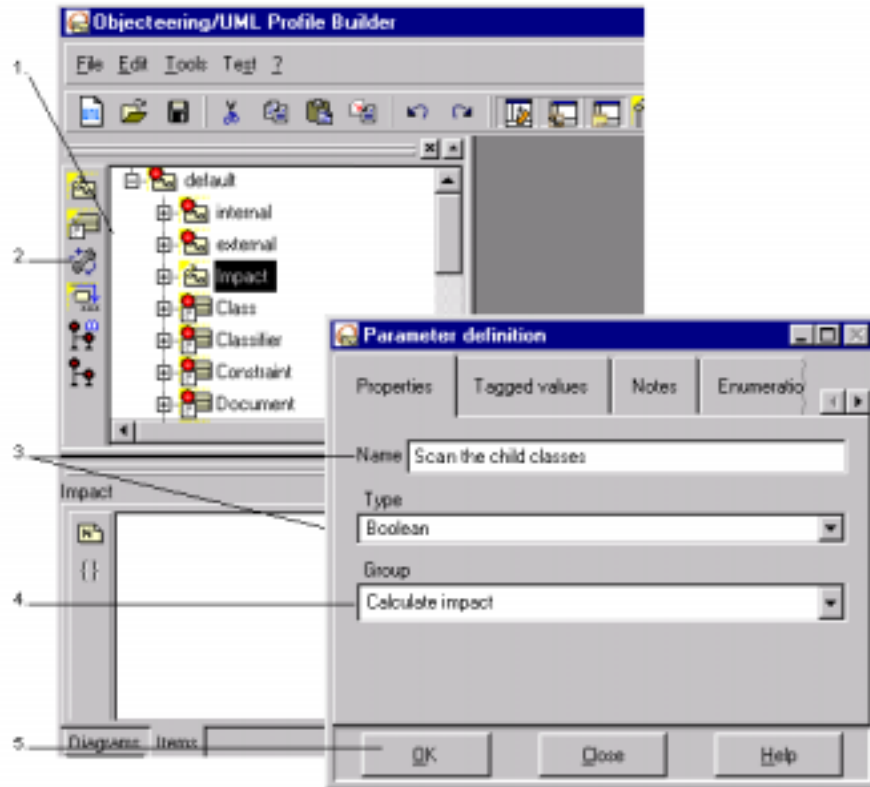



Figure 2-3. Creating the "Scan the child classes" module parameter

Chapter 2: First Steps: Creating a module

Steps:

- 1 - Select the "*Impact*" UML profile.
- 2 - Click on the  "Create a parameter" button.
- 3 - Enter the name and select the parameter type from the scrolling list. Please note that seven parameter types are now available: *Boolean*, *String*, *Enumeration*, *File open (String)*, *File save (String)*, *Directory (String)* and *Password (String)*.
- 4 - In the "*Group*" field, type "*Calculate impact*". This field is used to group parameters together in the different categories you specify. Several default groups are available for selection, but you may also, as in our example, create new groups.
- 5 - Confirm.

Create three other boolean parameters ("*Scan the heir classes*", "*Scan the method parameters*" and "*Scan the linked classes*") in the same group ("*Calculate impact*") and within the same UML profile.

Note: These parameters are visible during the "model configuration" phase (see figure 2-11).

Referencing a metaclass

Procedure

When a metaclass is referenced, (shown in Figure 2-4) you can designate those metaclasses that are of interest to different methods, according to the current UML profile. In a metaclass, we can define a UML profile's J methods, J attributes, note types and tagged value types.

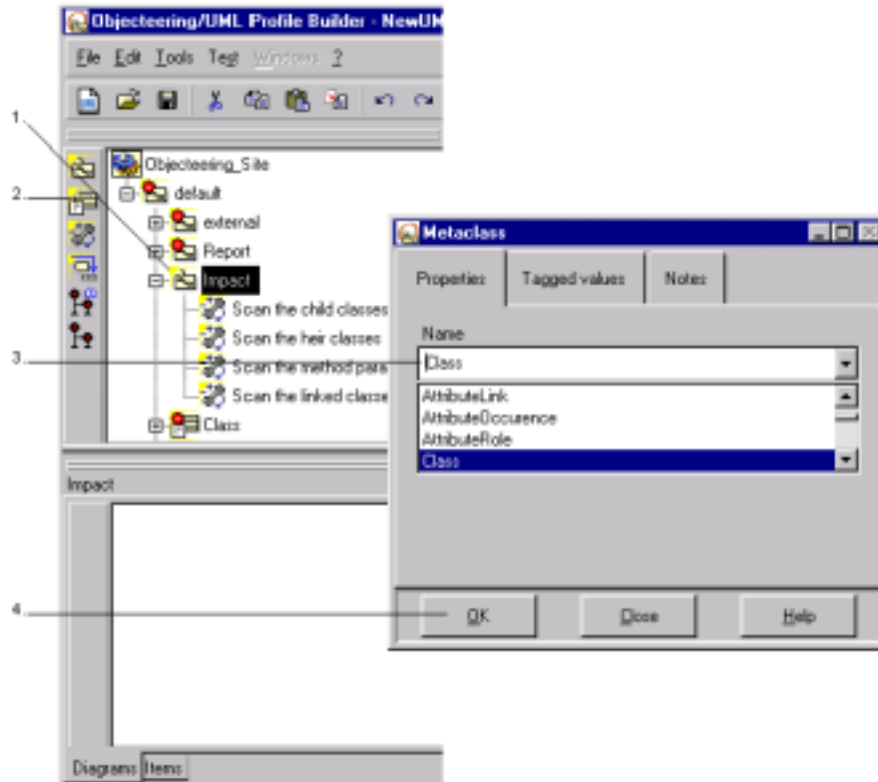



Figure 2-4. Referencing the "Class" metaclass

Chapter 2: First Steps: Creating a module

Steps:

- 1 - Select the "*Impact*" UML profile.
- 2 - Click on the  "Create a *metaclass reference*" button.
- 3 - Select the "*Class*" metaclass from the scrolling list.
- 4 - Confirm.

Creating a J method

We are now going to create the "*PrintImpact*" method(Figure 2-5) for the "*Class*" metaclass.

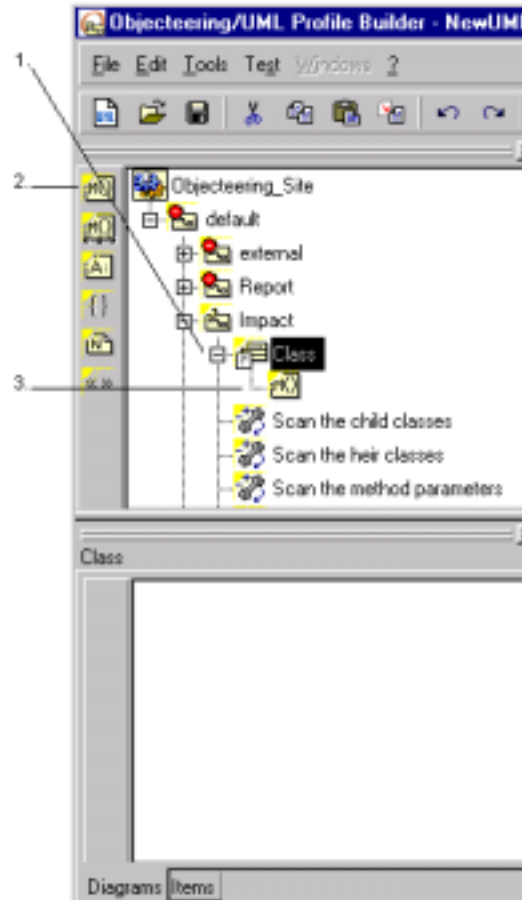



Figure 2-5. Creating the "*PrintImpact*" J method

Chapter 2: First Steps: Creating a module

Steps:

- 1 - Select the "Class" metaclass.
- 2 - Click on the  "Create a J method" button.
- 3 - Enter the "PrintImpact" name directly in the explorer and left-click to stop the continuous entry creation mode.

Note: If you wish to modify this name, right-click on the method, and choose the "Modify" option from the context menu which appears.

Procedure for entering a J method

Procedure

We are now going to launch an editor on the J method. The code which must be entered during the third step is given on the following page.

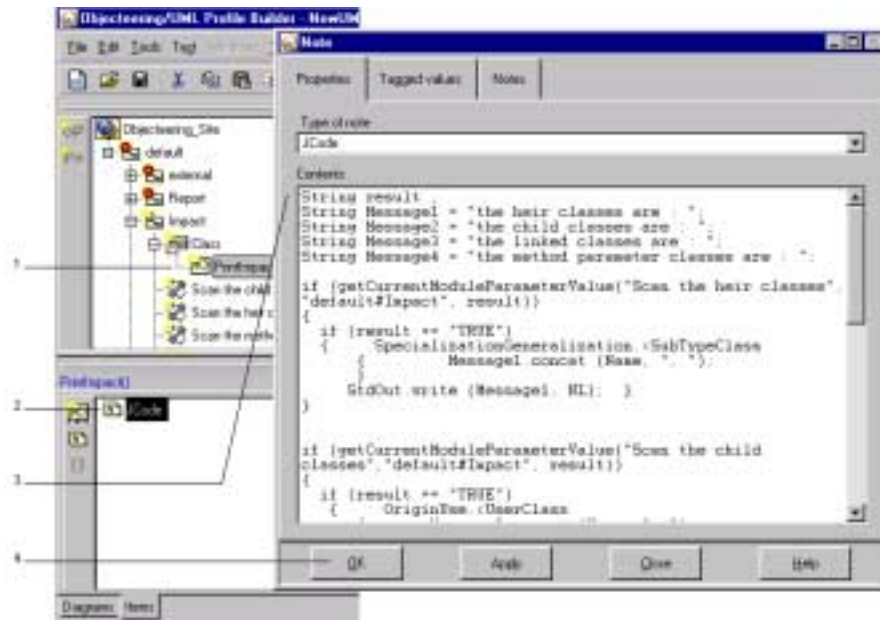


Figure 2-6. Entering the J code

Steps:

- 1 - Select the J method.
- 2 - Double-click on the method's "JCode" Note.
- 3 - Enter the code.
- 4 - Confirm.

Content of the J method

The following J code is entered. If you find entering the following code too long, simply enter "StdOut.write ("hello world");" (Figure 2-6):

```
String result ;
String Message1 = "the heir classes are : ";
String Message2 = "the child classes are : ";
String Message3 = "the linked classes are : ";
String Message4 = "the method parameter classes are : ";

if (getCurrentModuleParameterValue("Scan the heir
classes","default#Impact", result))
{
  if (result == "TRUE")
  {
    SpecializationGeneralization.<SubTypeClass
    {
      {
        Message1.concat (Name, ", ");
      }
      StdOut.write (Message1, NL); }
  }
}

if (getCurrentModuleParameterValue("Scan the child
classes","default#Impact", result))
{
  if (result == "TRUE")
  {
    OriginUse.<UserClass
    {
      {
        Message2.concat (Name, ", ");
      }
      StdOut.write (Message2, NL);
    }
  }
}
```

```
if (getCurrentModuleParameterValue("Scan the linked
classes","default#Impact", result))
{
  if (result == "TRUE")
  {
    PartAssociationEnd.<getOpposedRelationLink.<OwnerClass
    {
      Message3.concat(Name, ", ");
    }
    StdOut.write (Message3, NL);
  }
}

if (getCurrentModuleParameterValue("Scan the method
parameters","default#Impact", result))
{
  if (result == "TRUE")
  {
    OccurrenceParameter
    {
      Message4.concat (Name, ", ");
    }
    StdOut.write (Message4, NL);
  }
}
```

Creating a module

We are now going to create a module called "CalculateImpact" in the UML profiling project (Figure 2-7).

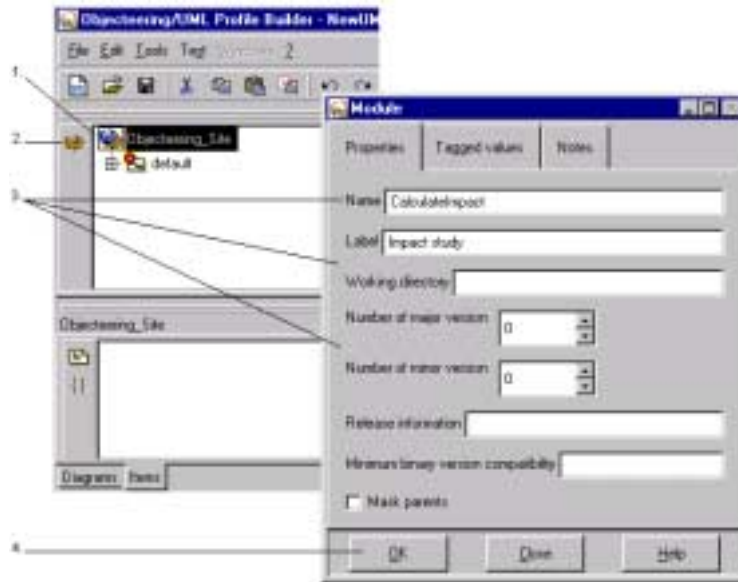



Figure 2-7. Creating the "CalculateImpact" module

Steps:

- 1 - Select the UML profiling project.
- 2 - Click on the  "Create a module" button.
- 3 - Fill in the necessary information (please see table below).
- 4 - Confirm.

The ... field	is used to ...
Name	give a name to the module which is being created (used internally).
Label	provide a label for the module. This will appear in the pop-up menus and other interfaces.
Working directory	enter the packaging target directory and the directory containing the module's external resources. If the " <i>Working directory</i> " field is left blank, then the working directory is \$OBJING_PATH/modules/<ModuleName>/<Version>. Otherwise, the user may specify the working directory of his choice.
Number of major version	indicate the first number in the version number (for example, if the complete version number is "4.6", then the major version number is 4.
Number of minor version	indicate the second number in the version number (for example, if the complete version number is "4.6", then the minor version number is 6.
Release information	indicate the release number for the module (for example, "a").
Minimum binary version compatibility	indicate Objectteering/UML binary requirements for the module.
Mask parents	mask the parameters and menus of the parent of the module which has been selected. If this box is not checked, the module's parameters and menus will cohabit with those of its parents. This box has no effect on modules which do not have parents.

Referencing a UML profile for the module

The "CalculateImpact" module must reference a UML profile. This operation allows you to proceed with the creation of commands (Figure 2-9).

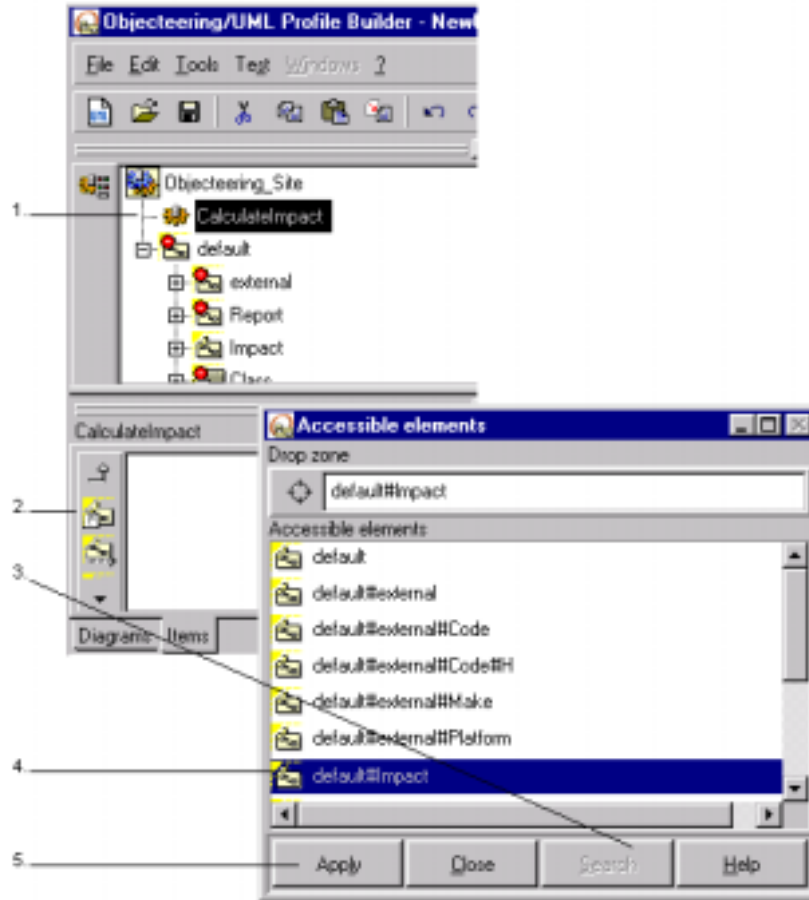



Figure 2-8. Referencing the "CalculateImpact" module

Steps:

- 1 - Select the "*CalculateImpact*" module.
- 2 - Click on the  "Reference a UML Profile" button.
- 3 - Click on the "Search" button.
- 4 - Select the "*default#Impact*" UML profile.
- 5 - Confirm.

Note: It is possible to use the drag and drop function to select the "*Impact*" UML profile, by selecting it in the explorer and dragging it into the drop zone. Click on "*Apply*" to confirm.

Creating a command

We can now create a command that references a module UML profile method.

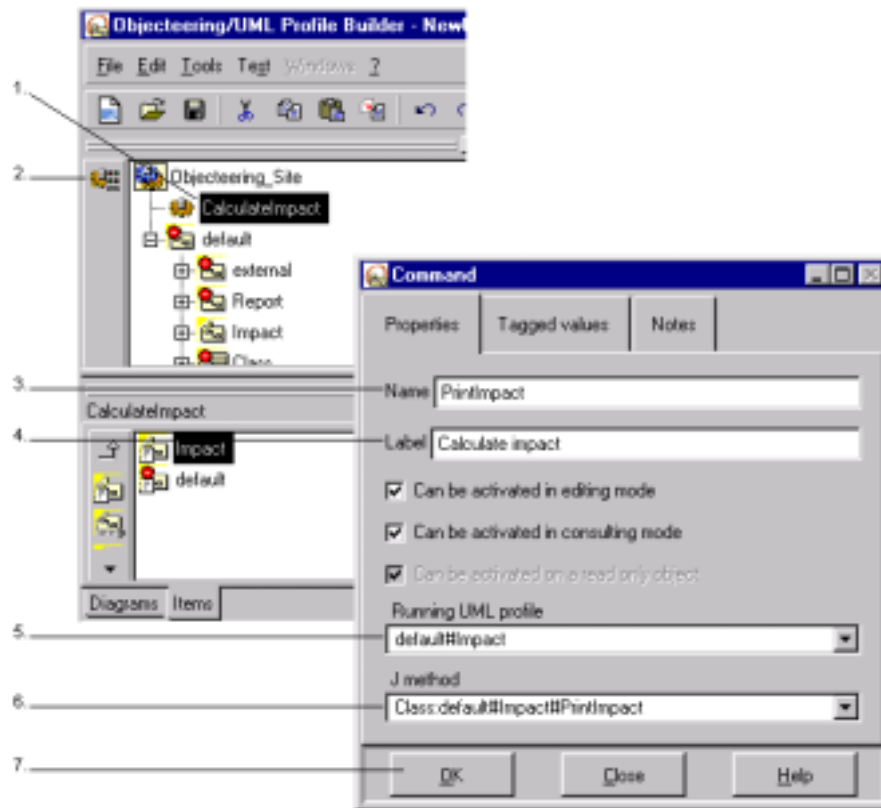



Figure 2-9. Creating a command

Steps:

- 1 - Select the "*CalculateImpact*" module.
- 2 - Click on the  "Create a command" button.
- 3 - Enter the name.
- 4 - Enter the label.
- 5 - Select the "*default#Impact*" UML Profile.
- 6 - Select the "*Class:default#Impact#PrintImpact*" J method.
- 7 - Confirm.

This command is now available in a pop-up menu available in all the UML modeling projects which use this module. We will have a look at this later in the test project.

Configuring a module

We are now going to define the default values of the module's parameters. The "CalculateImpact" module configuration window is opened from the main window (Figure 2-10).

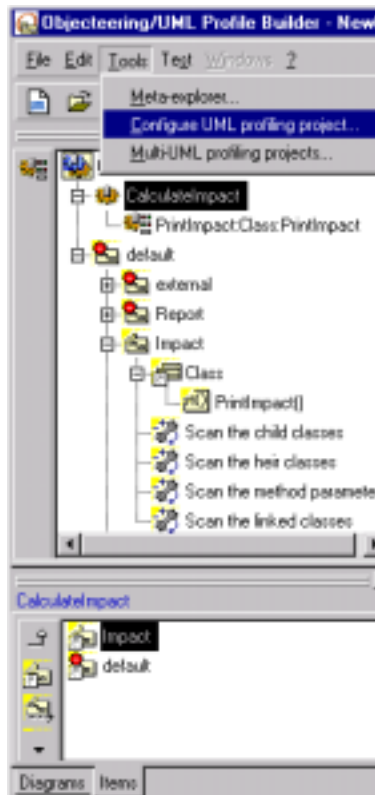


Figure 2-10. Configuring the "CalculateImpact" module

Select the "Tools" menu in the menu bar and then select the "Configure UML profiling project..." option.

Module configuration window

This configuration window allocates a part of the displayed hierarchy to each UML profiling project module which has at least one parameter. In this example, there is, therefore, only one section, "Calculate impact", with the four parameters previously created in the "Impact" UML profile. In figure 2-11, the default value has been set to "TRUE".

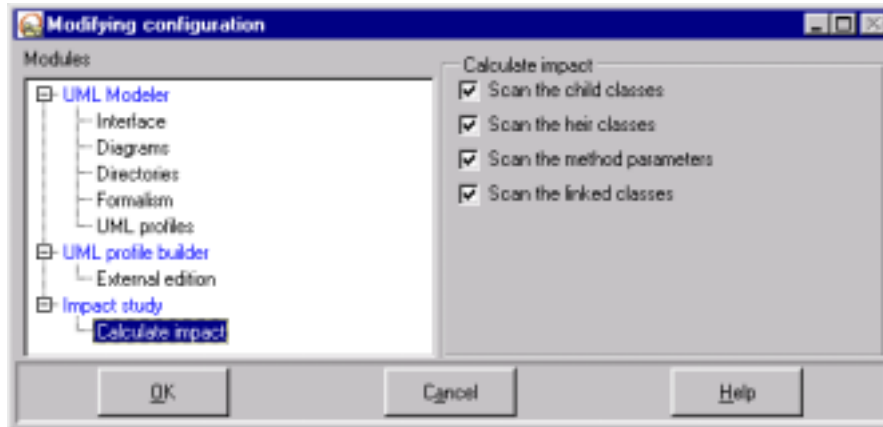


Figure 2-11. The module configuration window

Note: The configuration window of a module is updated automatically when the module has been modified (for example, when a new UML profile is referenced, when a parameter is added or deleted in a referenced UML profile).

Testing a UML profiling project

Selecting a test project

To carry out a test on the UML profiling project, you must first select a test project (Figure 2-12).

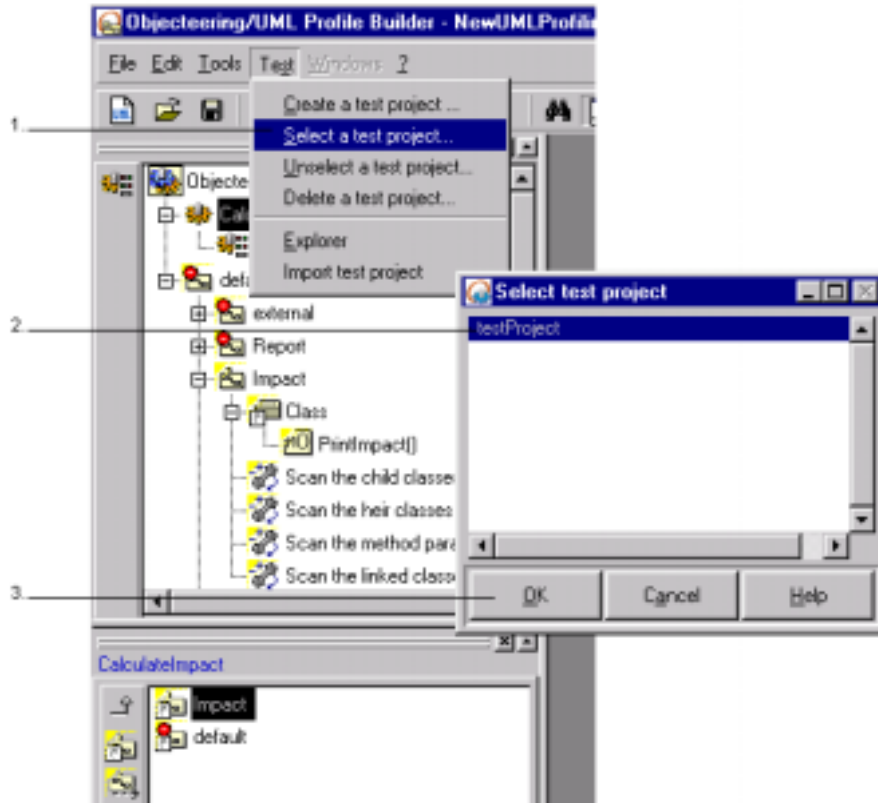


Figure 2-12. Selecting a test project.

Steps:

- 1 - Select the "*Select a test project...*" option from the "*Test*" menu.
- 2 - Select the test project concerned.
- 3 - Confirm. An explorer for this test project then opens automatically.

Note: To create a test project, simply select the "*Create a test project*" option and enter a name. The new test project then automatically opens.

Test project

We are now going to complete our example by entering classes in a test project. This will allow us to execute J code on the user model. If the project already has a model, this operation is optional.

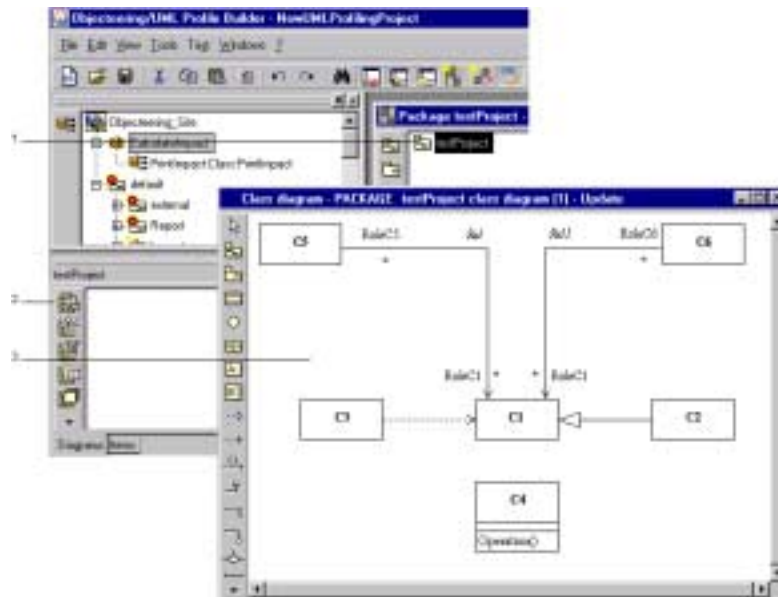



Figure 2-13. Selecting test project classes

Steps:

- 1 - Select the "testProject" package in the test project explorer. Once selected, the properties editor is activated for this element.
- 2 - Create a class diagram by clicking on the  "Create a class diagram" icon in the "Items" tab of the properties editor. The newly created diagram is then automatically opened.
- 3 - Create the classes and associations shown in the diagram in Figure 2-13.

Launching the command

The code is executed from a class belonging to the test project (Figure 2-14).

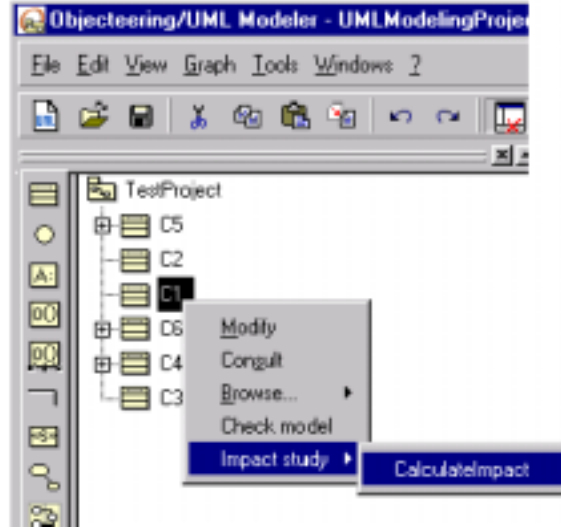


Figure 2-14. Executing J code

Steps:

- 1 - Click on the class in the test project explorer using the right mouse-button to activate the context menu.
- 2 - Select the "*Impact study/Calculate impact*" menu entries.

Chapter 2: First Steps: Creating a module

The code executes the following process, which is displayed in the console (Figure 2-15):

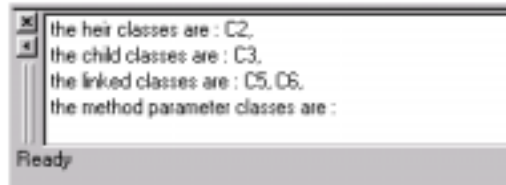


Figure 2-15. The console displaying the results of code execution

Chapter 3: First Steps: Building a generator

Creating a generator

Overview

Objectteering/UML Profile Builder can be used to create new code generators.

This chapter presents the generators and the different services available for defining new code generation.

The example which follows shows how to create a Java code generator.

This generator is used to:

- ◆ generate Java code
- ◆ visualize the generated files
- ◆ externally edit the generated files and retrieve the code in the Objectteering/UML repository after edition

Characteristics of Java code generation

To avoid complicating these first steps, the code generator will be very limited.

It will be used to generate:

- ◆ from a package or a class
- ◆ one file per class
- ◆ methods without parameters. Certain methods can be defined as being "*synchronized*", through the addition of a tagged value. Method code is entered in a note.
- ◆ attributes

Creating a UML profiling project

Create the "JavaProfilingProject" UML profiling project. This UML profiling project should contain the Java module, the Java UML profile, the tagged value types, the note types and the generation work product which are necessary for the Java generator.

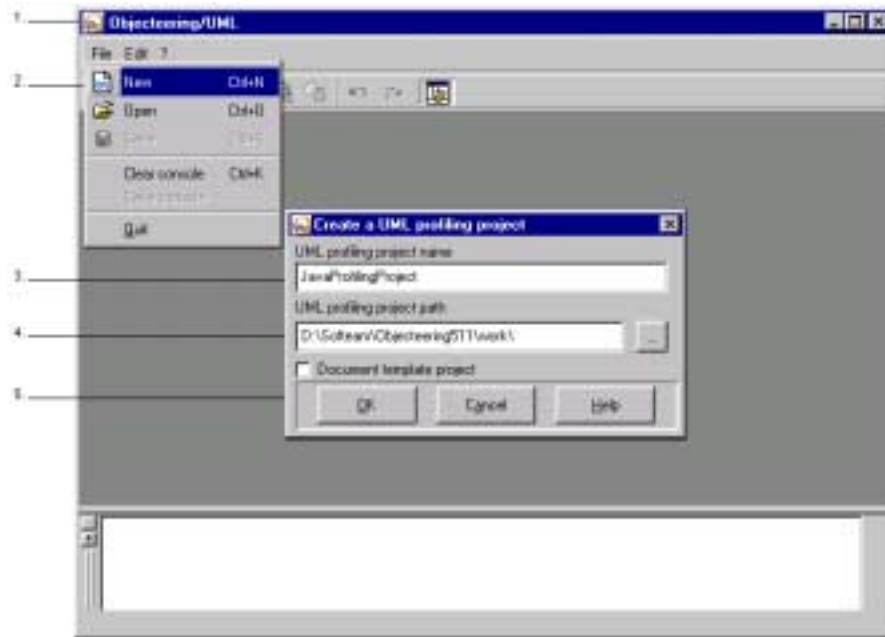




Figure 3-1. Creating the "JavaProfilingProject" UML profiling project

Steps:

- 1 - Click on the  *Objectteering/UML Profile Builder* icon in your desktop. The window shown in Figure 3-1 will then appear.
 - 2 - Click on the "File/New" menu. The "Create a UML profiling project" window will then open.
 - 3 - In the "UML profiling project name" field, enter the "JavaProfilingProject" name.
 - 4 - In the "UML profiling project path" field, enter the path of the directory where the new UML profiling project is to be created. You may also use the  icon to open a file browser through which you can select your UML profiling project path.
 - 5 - Confirm by clicking on the "OK" button.
-

Creating a UML profile

Introduction

A UML profile for a generator must be defined in the "*default#external#Code*" UML profile.

This UML profile contains services which:

- ◆ manage the markers of the generated files
- ◆ manage the edition
- ◆ retrieve the code after external edition

Creating a Java UML profile

Create the "Java" UML profile in the "default#external#Code" UML profile by following the steps illustrated in Figure 3-2. The Java generator will be defined on this UML profile.

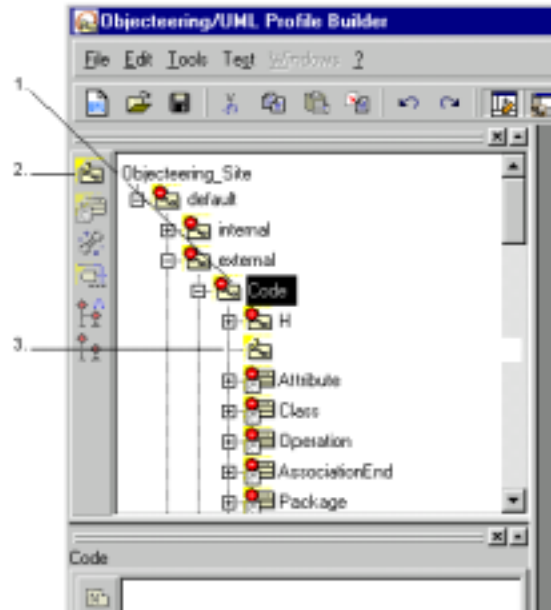



Figure 3-2. Creating a "Java" UML Profile

Steps:

- 1 - Select the "Code" UML profile in the "external" UML profile.
- 2 - Click on the  "Create a child UML profile" button.
- 3 - Enter the "Java" name.

Confirm by clicking outside the note type zone. If you press return, another note type will be created, using the continuous data entry mode.

Creating parameters

Existing parameters

Since the "Java" UML profile inherits from the "Code" UML profile, the following parameters are inherited:

The ... parameter	is used to ...
IdGenerated	generate markers in the generated files
ExtEditorCommandLine : command to invoke the external editor	launch an external editor to edit a generated file

Note: The parameters in this table are not visible from *Objectteering/UML Profile Builder*. (They can be parameterized through the "Tools/Configure the UML profiling project..." menu, in the "UML Profile Builder" section).

Creating new parameters

In the generation, we will use two parameters, which allow the user to identify the generation path and the default suffix.

In the "default#external#Code#Java" UML profile, create the "Java generation path", which allows you to define the default generation path for the generated files. Enter the following values:

- ◆ "Name" attribute: "Java generation path"
- ◆ "Type" attribute: "String"
- ◆ "Group" attribute: "Java product parameters"

In the "default#external#Code#Java" UML profile, create the "Java generation suffix", which allows you to define the extension of the generated files. Enter the following values:

- ◆ "Name" attribute: "Java generation suffix"
- ◆ "Type" attribute: "String"
- ◆ "Group" attribute: "Java product parameters"

Creating note types

Creating a note type

We are going to define a note type used to enter Java code on J methods. This note type will only be accessible for the "Operation" metaclass.

In the "default#external#Code#Java" UML profile, create a reference to the "Operation" metaclass.

For this reference, create the "JavaCode" note type.

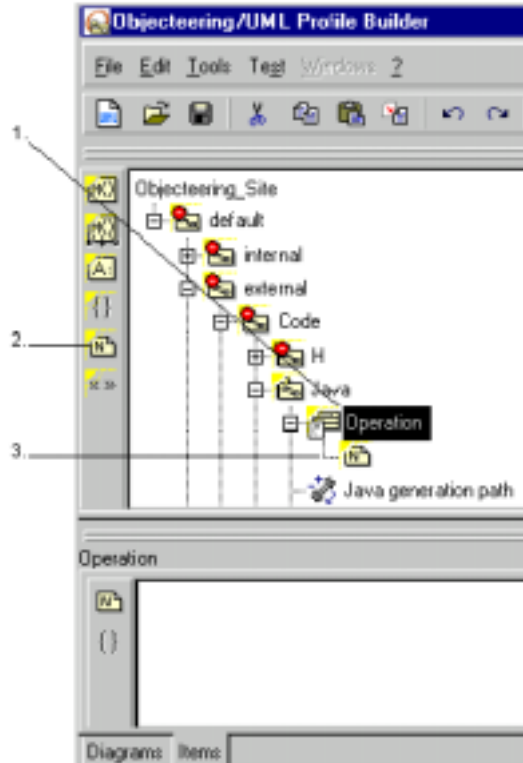



Figure 3-3. Creating the "JavaCode" text type

Steps:

- 1 - Select the "*Operation*" metaclass reference.
- 2 - Click on the  "Create a type of note" button.
- 3 - Enter the "*JavaCode*" name for the note type.

Confirm by clicking outside the note type zone. If you press return, another note type will be created, through the continuous data entry mode.

Creating tagged value types


Types of tagged values

Since the "Java" UML profile inherits from the "Code" UML profile, the following tagged values are available for Java:

The ... tagged value	on the ...metaclass	is used to ...
{nocode}	Attribute	generate nothing for attributes annotated with this tagged value
{nocode}	Class	generate nothing for classes annotated with this tagged value
{nocode}	Operation	generate nothing for operations annotated with this tagged value
{nocode}	AssociationEnd	generate nothing for associations annotated with this tagged value
{nocode}	Package	generate nothing for packages annotated with this tagged value

Creating a type of tagged value

The Java code generator will suggest that you use the `{synchronized}` tagged value on all the methods. This tagged value will allow you to automatically generate the "synchronized" keyword, which will be the header of the methods that will own it.

In the "Operation" metaclass reference, create the `{synchronized}` tagged value type, by clicking on the  "Create a tagged value type" icon and entering the following values:


- ◆ "Name": "synchronized"
- ◆ "Number of parameters": 0
- ◆ "Qualified": "FALSE"
- ◆ "Inclusion in the signature": "FALSE"

Creating J attributes

Creating J attributes

So as to keep generation information accessible in any context, we are now going to create three J attributes.

In the "default#external#Code#Java" UML profile, create an "Object" metaclass reference.

For this reference, create the "JavaOriginPath", "JavaOriginSuffix" and "OriginName" attributes, by clicking on the  "Create a J attribute" icon.

The ... J attribute	allows you to retain ...
JavaOriginPath	the generation path value
JavaOriginSuffix	the extension value
OriginName	the associated modeling element

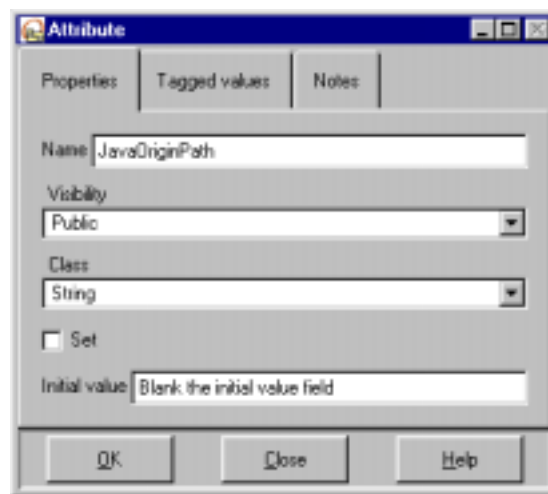


Figure 3-4. J Attribute dialog box

The "JavaOriginPath" attribute

Create the "JavaOriginPath" attribute, by entering the following values:

- ◆ In the "Name" field, enter "JavaOriginPath".
- ◆ In the "Visibility" field, enter "Public".
- ◆ In the "Class" field, enter "String".
- ◆ In the "Set" field, enter "FALSE".
- ◆ In the "Initial value" field, enter nothing.

The "JavaOriginSuffix" attribute

Create the "JavaOriginSuffix" attribute, by entering the following values:

- ◆ In the "Name" field, enter "JavaOriginSuffix".
- ◆ In the "Visibility" field, enter "Public".
- ◆ In the "Class" field, enter "String".
- ◆ In the "Set" field, enter "FALSE".
- ◆ In the "Initial value" field, enter nothing.


The "OriginName" attribute

Create the "OriginName" attribute, by entering the following values:

- ◆ In the "Name" field, enter "OriginName".
 - ◆ In the "Visibility" field, enter "Public".
 - ◆ In the "Class" field, enter "String".
 - ◆ In the "Set" field, enter "FALSE".
 - ◆ In the "Initial value" field, enter nothing.
-

Creating a generation work product

Creating a work product

In the "default#external#Code#Java" UML profile, create the "JavaProduct" generation work product, by clicking on the  "Create a product" icon.

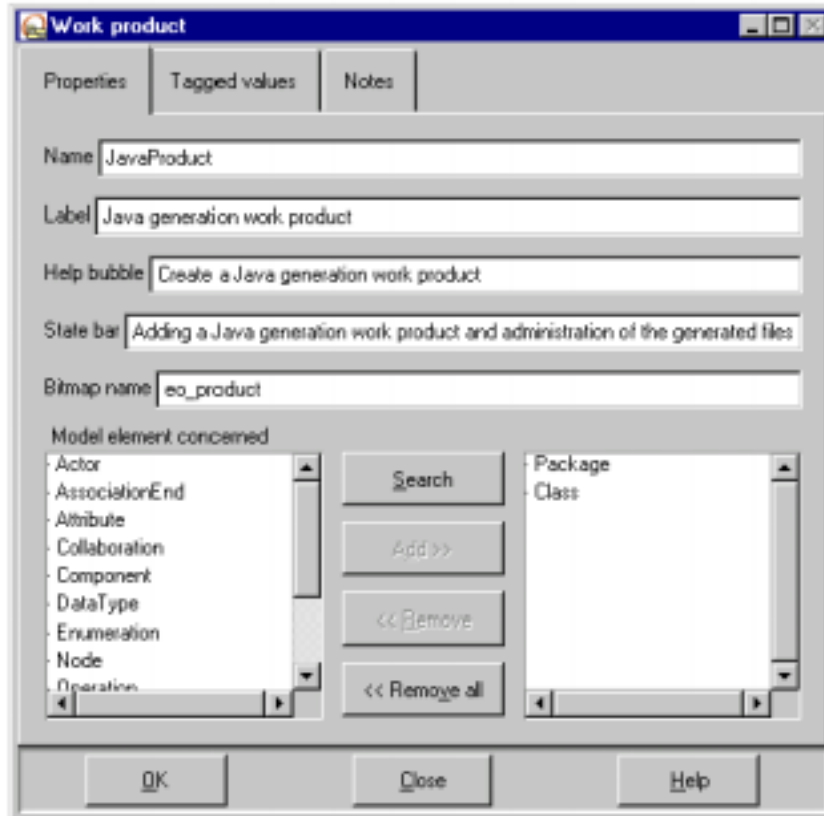



Figure 3-5. Work product dialog box

Enter the following values:


- ◆ In the "Name" field, enter "JavaProduct".
- ◆ In the "Label" field, enter "Java generation work product".
- ◆ In the "Help bubble" field, enter "Create a Java generation work product".
- ◆ In the "Status bar" field, enter "Adding a Java generation work product and administration of the files generated".
- ◆ In the "Bitmap name" field, enter "eo_product".
- ◆ In the "Modeling element concerned" field, enter "Package", "Class".

Creating parameters

Parameters are created for UML profiles and used by attributes, which are, in turn, created on generation work products. In the "default#external#Code#Java" UML profile, create the "Java generation path" and "Java generation suffix" parameters,

both of *String* type, by clicking on the  "Create a parameter" icon and defining the data entry fields in the dialog box which then appears.

Creating attributes

In the "*JavaProduct*" generation work product, create the "*path*" attribute, by clicking on the  "Create a meta-attribute" icon, which will allow you to enter the Java generation path (directory where the generated files will be stored). The suffix generation default value, displayed when the work product's dialog box opens, will be retrieved through the parameter defined for the module.

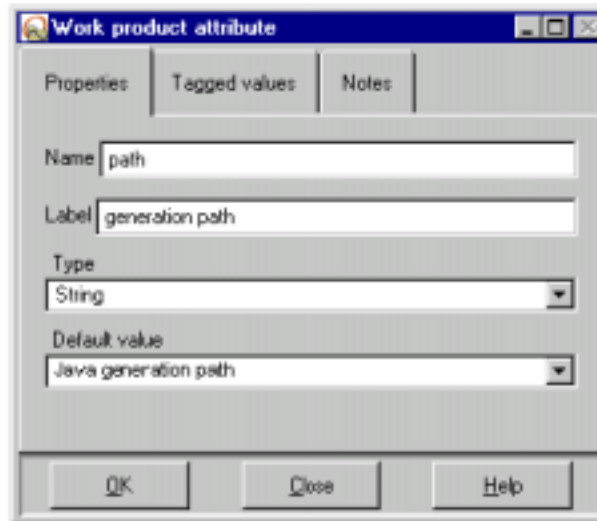


Figure 3-6. Meta-attribute dialog box

Enter the following values:

- ◆ In the "*Name*" field, enter "*path*".
- ◆ In the "*Label*" field, enter "*generation path*".
- ◆ In the "*Type*" field, enter "*String*".
- ◆ In the "*Default value*" field, enter "*Java generation path*".

In the "*JavaProduct*" generation work product, create the "*suffix*" attribute, which will allow you to enter the generated Java files suffix. The default path value, displayed when the work product's dialog box opens, will be retrieved through the parameter defined for the module.

Enter the following values for the "*suffix*" attribute:

- ◆ In the "*Name*" field, enter "*suffix*".
 - ◆ In the "*Label*" field, enter "*generation suffix*".
 - ◆ In the "*Type*" field, enter "*String*".
 - ◆ In the "*Default value*" field, enter "*Java generation suffix*".
-


Creating J methods

Creating J methods for the code generation

The following J methods allow you to generate Java code from the model.

In the "default#external#Code#Java" UML profile, first create the "Attribute", "Class", "Package" and "JavaProduct" metaclass references.

The ... J method	on the ... metaclass	is used to ...
generate () return String	Attribute	generate the attribute.
getType () return String	Attribute	get back the attribute type.
generate ()	Class	generate the Java code for the class.
generate ()	Package	recursively generate the Java code for its classes and those of its sub-packages.
generate ()	JavaProduct	launch Java code generation.
getCode () return String	Operation	generate the method body.
generate () return String	Operation	generate the method (only the methods without parameters or without return parameters are generated) .

Certain J methods contain return parameters. To create a return parameter, select the method concerned and click on the  "Add a return parameter" icon.

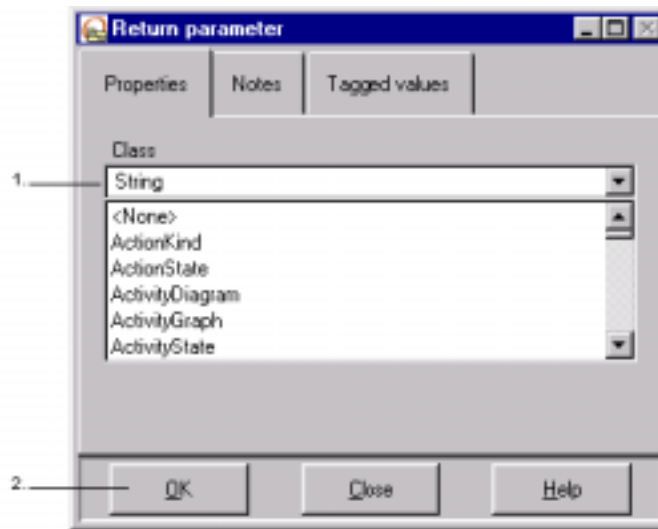


Figure 3-7. Adding a return parameter to a J method


Steps:

- 1 - Select a class.
- 2 - Confirm.

Creating J methods for managing work products

The ... J method	on the ... metaclass	is called on the updateGraph method...
initProduct (Product : in MpGenProduct)	JavaProduct	when the current work product is created from a parent product.
update (Product : in MpGenProduct)	JavaProduct	when the current work product or associated modeling element has been modified.
mustPropagate () return boolean	JavaProduct	to allow the generation work products to spread to the sub-packages and classes.
isPresent (Product : in MpGenProduct) return boolean	JavaProduct	to avoid creating several similar work products on the same model element while they spread. This method allows you to define the criteria for creating work products.

Warning!

Certain J methods contain parameters. To create a parameter, select the method concerned and click on the  "Add a Parameter" icon.

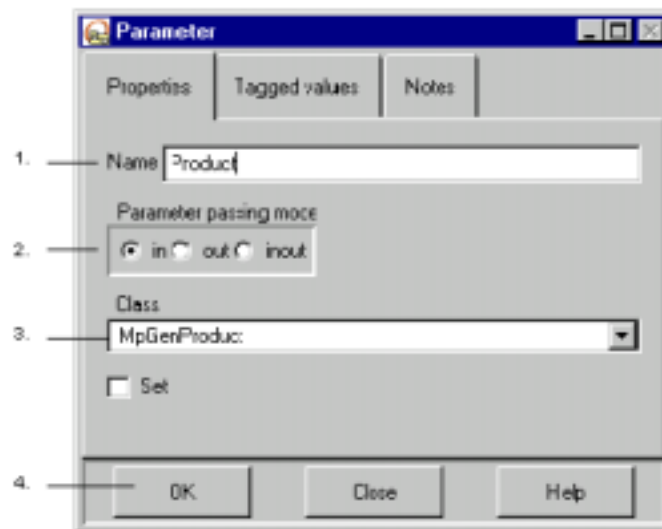


Figure 3-8. Adding a parameter to a J method

Steps:

- 1 - Enter the parameter name.
- 2 - Select a passing mode.
- 3 - Choose the class.
- 4 - Confirm.

Creating the visualization J methods

The ... J method	on the ... metaclass	is used to ...
visualize ()	JavaProduct	visualize the file generated by an internal editor.
edit ()	JavaProduct	edit the file generated by an external editor.
getIdLineComment () return String	JavaProduct	return the characters placed in front of the identifiers and kept in the generated file.

Creating J methods for managing the module

The ... J method	on the ... metaclass	is called ...
moduleInstall ()	Object	when the module is installed.
moduleUninstall ()	Object	when the module is uninstalled.

Implementing J methods

Run the "Edit the J code" command on the Java UML profile as shown in Figure 3-9.

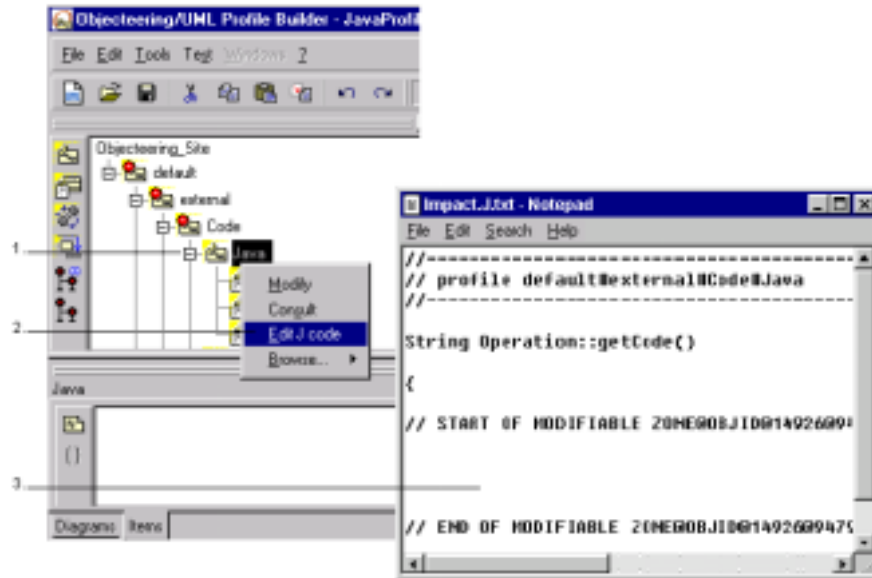


Figure 3-9. Editing the J code

Steps:

- 1 - Click on the "Java" UML profile using the right mouse-button.
- 2 - Select the "Edit J code" option from the context menu which appears.
- 3 - Enter the zone that can be modified.

The "JavaProduct::generate ()" method

This method allows the generation of Java code from the generation work product. It consists of generating the modeling element code referenced by the work product (calling of the generate method). All open visualizers are updated after generation.

```
Void JavaProduct::generate ()
{
    // Java code generation
    OriginModelElement.<generate();

    // updating of all the visualizers
    updateAllEditors();
} // method generate
```

The "Package::generate ()" method

This method allows the generation of Java code from a package. It consists of executing the "generate" method on packages and their packages.

```
Void Package::generate ()
{
    // displaying a message in the console
    StdOut.write ("Generation of the package ", Name, NL);

    // generation of the packages of the current package.
    OwnedElementPackage.<generate();

    // generation of the classes of the current package.
    OwnedElementClass.<generate();
}
```

The "Class::generate ()" method

This method allows the generation of Java code from a class. It consists of generating the class header, attributes and methods, saving the generated files and allowing it be administrated by the work product referenced by the class.

```

Void Class::generate ()
{
    String content;
    MpGenProduct genProduct;
    String fileName;

    // displaying a message in the console
    StdOut.write (Name, " : ");

    // generation of a comment at the beginning of the class
    // this part of code cannot be modified
    content.strcat (idGen ());
    content.strcat ("// -----", NL);
    content.strcat ("// Class ", Name, NL);
    content.strcat ("// -----", NL);
    content.strcat (idEnd ());

    // generation of the class
    // this part of code can be modified
    // thanks to a dialog box
    content.strcat (idBox ());
    content.strcat ("class ", Name, NL);
    content.strcat (idEnd ());

    // generation of the opening bracket
    // this part of code cannot be modified
    content.strcat (idGen ());
    content.strcat ("{" , NL);
    content.strcat (idEnd ());

    // generation of the class attributes
    PartAttribute
    {
        content.strcat(generate());
    }

    -- generation of the class methods
    PartOperation
    {
        content.strcat(generate());
    }
}

```


Chapter 3: First Steps: Building a generator

```
// generation of the closing bracket
// this part of code cannot be modified
content.strcat (idGen ());
content.strcat ("}", NL);
content.strcat (idEnd ());

// displaying a message in the console
StdOut.write ("generate", NL);

// recapturing a class generation work product
genProduct = getAnyProduct();

if (notVoid (genProduct))
  { // creation of the generated file path
    fileName.strcat (genProduct.getAttributeVal("path"),
                    "/",
                    Name,
                    ".",
                    genProduct.getAttributeVal("suffix"));

    // letting the file be managed by
    // the class work product
    genProduct.mngFile (fileName, content);
  }
else
  {StdOut.write ("The class ~", Name, "~" has no Java
work product", NL);}
} // method generate
```

The "Operation::generate ()" method

This method is used to generate a method if the operation is not annotated "nocode" and if it has no in/out parameters or return parameters. It consists of adding the keyword "synchronized" if the method has the {synchronized} tagged value, and retrieving the text to be entered in the method body.

```
String Operation::generate ()
{
    if (isTaggedValue ("nocode") == false) {
        // only methods without parameters should be generated
        if ((IOPParameter.card() == 0) && !(notVoid
(ReturnParameter)))
        {
            // generation of the method
            // this part of code can be modified
            // thanks to a dialog box
            return.strcat (idBox ());

            return.strcat (Tab);

            // adding the word "synchronized" if the tagged
            // value is positioned
            if (isTaggedValue ("synchronized")) {
                return.strcat ("synchronized ");
            }

            return.strcat ("void ", Name, "()", NL);
            return.strcat (idEnd ());

            // adding of the opening bracket
            // this part of code cannot be modified
            return.strcat (idGen ());
            return.strcat ("{" , NL);
            return.strcat (idEnd ());

            // generation of the method implementation
            return.strcat(getCode ());

            // adding of the closing bracket
            // this part of code cannot be modified
            return.strcat (idGen ());
            return.strcat ("} ", NL);
            return.strcat (idEnd ());
        }
    }
} // method generate
```

The "Operation::getCode ()" method

This method allows you to retrieve method body code. It returns the concatenation of all the "JavaCode" text types defined on the current method.

```
String Operation::getCode ()
{
    // recapturing the content of all
    // the "JavaCode" notes
    DescriptorNote.<select (ModelNoteType.Name == "JavaCode")
    {
        // generation of the content
        // this part of code can be modified
        // by the external editor
        return.strcat (idTxt ());
        return.strcat (Tab, Tab, Content, NL);
        return.strcat (idEnd ());
    }

    // If there is no "JavaCode" note
    // on the Operation, then markers are inserted
    // allowing the automatic creation this type of
    // text after external edition of the generated file
    if (return == "") {
        return.strcat (marker ("Descriptor", "JavaCode"));
    }
} // method getCode
```

The "Attribute::generate ()" method

This method is used to generate an attribute. It returns a string containing the attribute's type and name.

```
String Attribute::generate ()
{
    // generation of the current attribute
    // this part of code can be modified thanks to a dialog
    box
    return.strcat (idBox ());
    return.strcat (Tab, getType (), " ", Name, ";", NL);
    return.strcat (idEnd ());
} // method generate
```

The "Attribute::getType ()" method

This method allows you to retrieve an attribute's type. It returns a string according to the type defined in the model.

```
String Attribute::getType ()
{
    // returns the type of a Java attribute according to
    // the modeled type
    TypeGeneralClass
    {
        if (Name == "integer")
            return = "int";
        else if (Name == "real")
            return = "float";
        else if (Name == "String")
            return = "String";
        else
            return = Name;
    }
} // method getType
```

The "JavaProduct::initProduct ()" method

This method allows a new work product to be updated in relation to its parent. It consists of defining the name, suffix and path of the current work product according to the values of the "parent" work product. This method will be called on the Java generation work product of a package, with the package that contains it when generation is run.

```
Void JavaProduct::initProduct (in MpGenProduct Product)
{
    String ProductName;
    String Suffix;
    String Path;

    // start of a session
    sessionBegin ("Propagate", true);

    if (notVoid (Product)) {
        // getting back the values of the father work product.
        ProductName = Product.Name;
        Path        = Product.getAttributeVal ("path");
        Suffix      = Product.getAttributeVal ("suffix");

        // initialization of the current work product
        setName (ProductName);
        setAttributeVal ("path", Path);
        setAttributeVal ("suffix", Suffix);
    }

    // end of the "Propagate" session
    sessionEnd ();
} // method initProduct
```

The "JavaProduct::update ()" method

This method is used to update a work product in relation to its parent. It consists of destroying all the files generated by the work product. This method will be called on the work product associated to a package when the package that contains it, or the work product itself, is modified.

```
Void JavaProduct::update (in MpGenProduct Product)
{
    String ProductName;
    String Suffix;
    String Path;

    // start of a session
    sessionBegin ("Propagate", true);

    if (notVoid (Product)) {
        // getting the values of the parent work product.
        ProductName = Product.Name;
        Path        = Product.getAttributeVal ("path");
        Suffix      = Product.getAttributeVal ("suffix");

        // initialization of the current work product
        setName (ProductName);
        setAttributeVal ("path", Path);
        setAttributeVal ("suffix", Suffix);
    }

    // deletion of all the files managed by the work product
    deleteAllFiles ();

    // end of the "Propagate" session
    sessionEnd ();
} // method update
```

The "JavaProduct::mustPropagate ()" method

This method is used to indicate how the work product must propagate the child model elements. It returns the "true" value, whatever the modeling element linked to the current work product. The work product therefore propagates to all the packages and classes contained in the modeling element linked to the work product.

```
Boolean JavaProduct::mustPropagate ()
{
    // the propagation is carried out for any modeling
    element
    // that is associated to the current work product
    // A similar work product will therefore be built for all
    // the packages && classes
    return = true;
} // method mustPropagate
```

The "JavaProduct::isPresent" method

This method is used to filter the work product's propagation to the child model elements. It consists of preventing a "*JavaProduct*" work product from being created on a model element that already has one.

```
boolean JavaProduct::isPresent (in MpGenProduct Product)
{
    // prevents from having a work product of the same type
    // on a modeling element that would be of
    // package or class type
    if (Product.ClassOf == ClassOf)
        return = true;
    else
        return = false;
} // method isPresent
```

The "JavaProduct::visualize " method

This method is used to display a file managed by the current work product through a visualizer.

```
Void JavaProduct::visualize ()
{
    String fileName;

    // construction of the complete path
    fileName.strcat (getAttributeVal ("path"),
                    "\",
                    OriginModelElement.Name,
                    ".",
                    getAttributeVal ("suffix"));

    // message display in the console
    StdOut.write ("Visualization of the file ", fileName,
NL);

    // internal visualization of the generated file
    intVisuFileName (fileName);
} // method visualize
```

Note: Please note that for Linux and Unix, you should use "/" instead of "\" (for this method and the following method).

The "JavaProduct::edit" method

This method is used to edit a file managed by the current work product in an external editor.

```
Void JavaProduct::edit ()
{
    String fileName;

    // construction of the complete path
    fileName.strcat (getAttributeVal("path"),
                    "\",
                    OriginModelElement.Name,
                    ".",
                    getAttributeVal("suffix"));

    // message display in the console
    StdOut.write ("File edition", fileName, NL);

    // external edit of the generated file
    extEditFileName(fileName);
} // method edit
```

The "JavaProduct::getIdLineComment " method

This method returns the characters generated before the markers.

```
String JavaProduct::getIdLineComment ()
{
    return = "// ";
} // method getIdLineComment
```

The "Object::moduleInstall" method

```
void Object::moduleInstall ()
{
    // message display in the console
    StdOut.write ("Installation of Java module", NL);
} // methode moduleInstall
```

The "Object::moduleUninstall" method

```
void Object::moduleUninstall ()
{
    // message display in the console
    StdOut.write ("Uninstalling the Java module ", Name, NL);
} // method moduleUninstall
```

Creating a module

Creating the Java module

In the "*JavaProfilingProject*" UML profiling project, create the "Java" module. This module must reference the "*default#external#Code#Java*" UML profile. (Parent UML profiles are automatically referenced).

The ... field	has the ... value
Name	"ModuleJava"
Label	"Java code generator"
Working directory	This field is left blank. (If the " <i>Working directory</i> " field is left blank, then the working directory is \$OBJING_PATH/modules/<ModuleName>/<Version>. Otherwise, the user may specify the working directory of his choice.)
Number of major version	"1"
Number of minor version	"0"
Release information	This field is left blank.
Minimum binary version compatibility	This field is left blank.
Mask parents	"FALSE"

Creating commands

Overview

In the Java module, we are going to create the following three commands:

The ... command	is used to ...
Generate	run the generation of the Java code.
Visualize the class	visualize the generated file.
Edit the class	edit the generated file.

The "Generate" command

To create the "Generate" command, enter the following values:

- ◆ In the "Name" field, enter "generate".
- ◆ In the "Label" field, enter "Generate".
- ◆ In the "Can be activated in edition mode" field, enter "TRUE".
- ◆ In the "Can be activated in consulting mode" field, enter "TRUE".
- ◆ In the "Running UML profile" field, enter "default#external#Code#Java".
- ◆ In the "J method" field, enter "JavaProduct:default#external#Code#Java#generate".

The "Visualize the class" command

To create the "Visualize the class" command, enter the following values:

- ◆ In the "Name" field, enter "visualize".
- ◆ In the "Label" field, enter "Visualize the class".
- ◆ In the "Can be activated in edition mode" field, enter "TRUE".
- ◆ In the "Can be activated in consulting mode" field, enter "TRUE".
- ◆ In the "Running UML profile" field, enter "default#external#Code#Java".
- ◆ In the "J method" field, enter "JavaProduct:default#external#Code#Java#visualize".

The "Edit the class" command

To create the "Edit the class" command, enter the following values:

- ◆ In the "Name" field, enter "edit".
 - ◆ In the "Label" field, enter "Edit the class".
 - ◆ In the "Can be activated in edition mode" field, enter "TRUE".
 - ◆ In the "Can be activated in consulting mode" field, enter "TRUE".
 - ◆ In the "Running UML profile" field, enter "default#external#Code#Java".
 - ◆ In the "J method" field, enter "JavaProduct:default#external#Code#Java#edit".
-

Configuring a module

Default values

We are now going to change the configuration of your UML profiling project. For the "*JavaModule*" module, modify the default values of the module parameters.

The ... parameter	of the ... group	has the ... value
Generate the identifiers	External edition	"FALSE"
Command for invoking external editor	External edition	"vi" for unix "notepad" for windows
Java generation path	Java product parameters	"\$(GenRoot)/work" for Unix "\$(GenRoot)\work" for Windows
Java generation suffix	Java product parameters	"java"

Testing a module

Testing the UML profiling project

Objectteering/UML Profile Builder allows you to test the modifications made to the UML profiling project in real time, without having to start Objectteering/UML again.

For this, tests are carried out from a test project explorer.

Creating a test project

To create a test project to test the module you have just developed, carry out the steps illustrated in Figure 3-10 below.

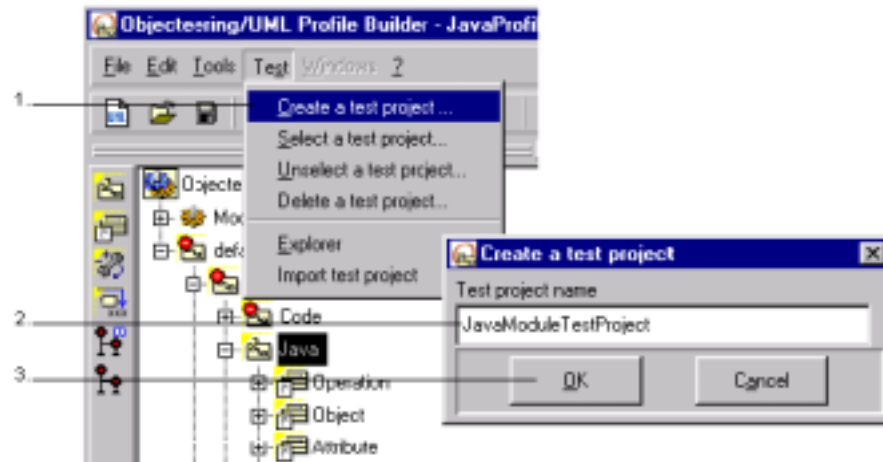


Figure 3-10. Creating a test project for your newly created Java generator

Steps:

- 1 - Click on the "Test" menu and run the "Create a test project..." command. The "Create a test project" window then appears.
- 2 - In the "Test project name" field, enter the name of your new test project.
- 3 - Click on "OK" to confirm.

The newly created test project then appears in an explorer. The "ModuleJava" module you have just created is automatically selected for this test project.

Inside this test project, create a simple model (packages, classes, operations, and so on).

For further information on test projects, please refer to chapter 8, "Test projects", of this user guide.

Generating code

To generate Java code on your new test project using the "ModuleJava" module you have just created, you must first create a generation work product, as shown in Figure 3-11.

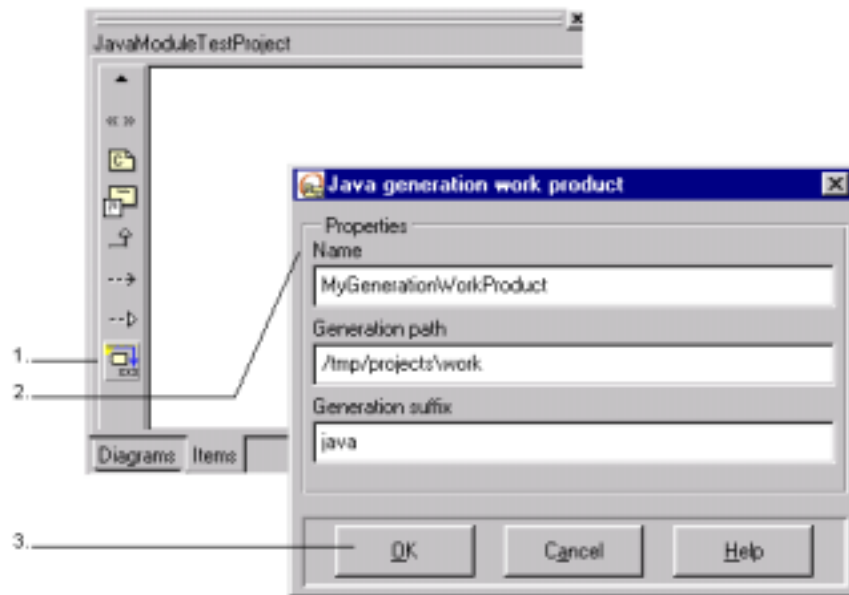


Figure 3-11. Testing generation work product creation in your test project

Steps:

- 1 - In the test project explorer, select the element for which you wish to create a generation work product, and then click on the generation work product icon in the "Items" tab of the properties editor.
- 2 - Enter the necessary information.
- 3 - Confirm by clicking on "OK".

After confirmation, your new generation work product appears in the "Items" tab of the properties editor.

To launch Java code generation, simply carry out the steps shown in Figure 3-12.

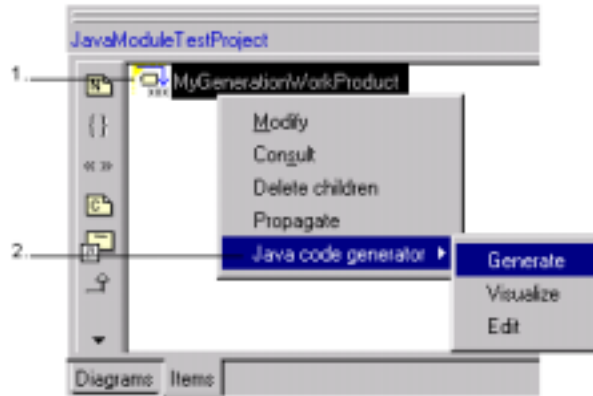


Figure 3-12. Testing Java code generation on your test project

Steps:

- 1 - Select the newly created work product in the "*Items*" tab of the properties editor by right-clicking.
- 2 - Run the "*Java code generator/Generate*" command.

Java code generation is then run.

Visualizing and editing generated code

Complete these first steps by testing the "*Visualize*" and "*Edit*" commands on the generated work product.

Chapter 4: Using Objecteering/UML Profile Builder

Launching Objecteering/UML Profile Builder

Launching the UML Profile Builder tool

The *Objecteering/UML Profile Builder* tool can be launched:

- ◆ by clicking on the UML Profile Builder icon in your desktop
- ◆ by selecting "*Start/Programs/Objecteering/Objecteering UML Profile Builder*"
- ◆ by typing the "*objing -profiling*" command or "*objingprofile*" in UNIX

Note: Please note that when UML Profile Builder is launched, the user cannot work on UML modeling projects, and when the UML Modeler is launched, the user cannot work on UML profiling projects.

Creating or opening a UML profiling project

Creating a UML profiling project

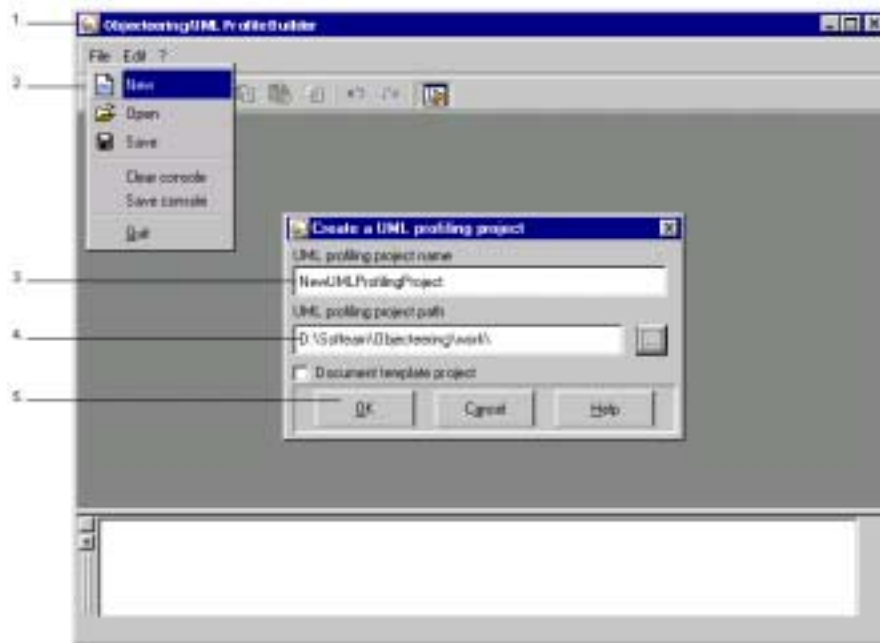




Figure 4-1. Creating a UML profiling project

Steps:

- 1 - Click on the  *Objecteering/UML Profile Builder* icon in your desktop. The window shown in Figure 4-1 will then appear.
- 2 - Click on the "File/New" menu. The "Create a UML profiling project" window will then open.
- 3 - In the "UML profiling project name" field, enter the name of the UML profiling project which is to be created.
- 4 - In the "UML profiling project path" field, enter the path of the directory where the new UML profiling project is to be created. You may also use the  icon to open a file browser through which you can select your UML profiling project path.
- 5 - Confirm by clicking on the "OK" button.

Note: Document templates are created by checking the "*Document template project*" tickbox.

Opening an existing UML profiling project

The procedure for opening an existing UML profiling project is very similar to the procedure for creating a new UML profiling project. Simply carry out the following steps:

- 1 - Click on the *Objectteering/UML Profile Builder* icon in your desktop. The window shown in Figure 4-1 will then appear.
- 2 - Click on the "File/Open" menu. The "Open an existing UML profiling project" window will then open.
- 3 - Double-click on the UML profiling project you wish to open. It will then automatically open.

Note 1: It is also possible to open an existing UML profiling project simply by double-clicking on it in the explorer. This launches Objectteering/UML and opens the UML profiling project you have selected.

Note: UML profiling projects may only be opened using the *Objectteering/UML Profile Builder* tool, whilst UML modeling projects may only be opened using the *Objectteering/UML Modeler* tool.

Receiving or upgrading UML profiling projects

As we have just seen, UML profiling projects can be opened simply by double-clicking on the UML profiling project file either in the Windows explorer or in the "Open an existing UML profiling project" window.

However, if you wish to work on a UML profiling project created on another user site and/or using an earlier version of Objectteering/UML, the UML profiling project has to be received and/or upgraded. For further information, please refer to the "Receiving and upgrading UML profiling projects" section in the current chapter of this user guide.

Saving your model context

It is possible to save your model context, which can be particularly useful when opening an existing UML profiling project. For further details, please refer to the "Saving your model context" section in chapter 3 of the *Objectteering/UML Modeler* user guide.

Receiving and upgrading UML profiling projects

Introduction

Objectteering/UML features a simplified UML profiling project reception and upgrade procedure, thus ensuring backward compatibility and making Objectteering/UML even easier for you to use.

If you wish to work on a UML profiling project either created on a different user site or using an earlier version of Objectteering/UML, you can launch the reception and/or upgrade procedures simply by:

- ◆ double-clicking on the UML profiling project in question in the Windows explorer
- ◆ double-clicking on the UML profiling project in question in a file browser available via the "*Open an existing UML profiling project*" window

Receiving a UML profiling project

There are four possible case scenarios with regard to the reception of UML profiling projects:

- ◆ the reception of a database which is not known to your site, but which exists in the same version of Objectteering/UML. In this case, you can choose to simply receive the database in question on your site.
- ◆ the reception of a database which is not known to your site, and which exists in an earlier version, for example, the 4.3.2 version of Objectteering/UML. In this case, you can choose to receive the database in question on your site, and upgrade it from the previous version to the current version of Objectteering/UML.
- ◆ the reception of a database which is not known to your site, but which has the same name as an existing database and which exists in the same version of Objectteering/UML. In this case, you can choose to receive the database in question on your site and rename it.
- ◆ the reception of a database which is not known to your site, but which has the same name as an existing database and which exists in an earlier version, for example, the 4.3.2 version of Objectteering/UML. In this case, you can choose to receive the database in question on your site, rename it and upgrade it from the previous version to the current version of Objectteering/UML.

Chapter 4: Using Objecteering/UML Profile Builder

In the example shown below (Figure 4-2), a simple reception operation is demonstrated.

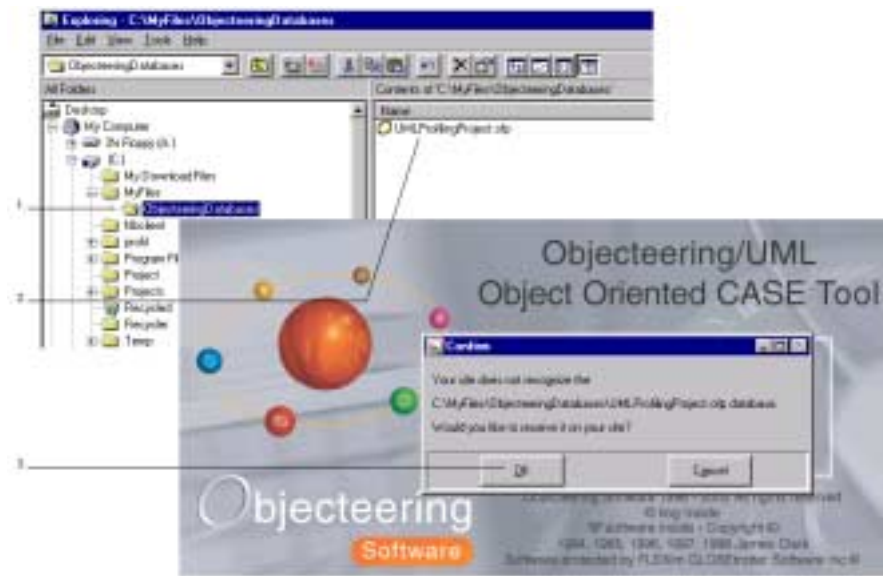


Figure 4-2. Receiving the "ProfilingProject" UML profiling project by double-clicking in the Windows explorer

Steps:

- 1 - In the Windows explorer, position yourself in the directory containing the UML profiling project you wish to use.
- 2 - Double-click on this UML profiling project file. A confirmation dialog box then appears, asking you if you wish to receive the database of the UML profiling project on your site.
- 3 - Click on the "OK" button to confirm the reception of the UML profiling project.

The reception procedure is then launched, and you can follow its progress in the console. This process can take a few minutes. Once completed, the UML profiling project is opened and is ready for use.

Upgrading a UML profiling project

To upgrade a UML profiling project created on your site using, say, the 5.1.1 version of Objecteering/UML, simply carry out the steps below (shown in Figure 4-3).

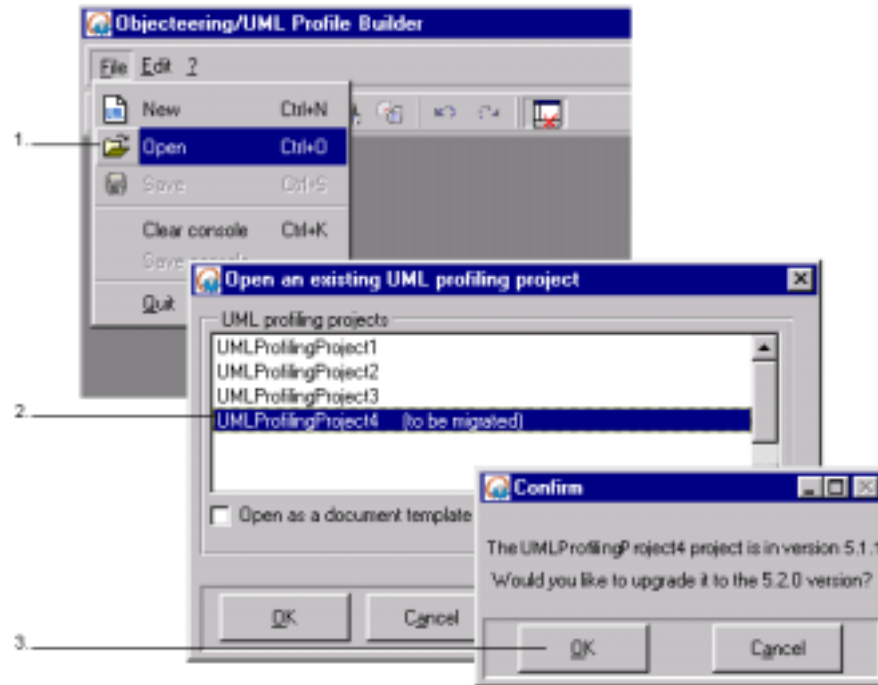


Figure 4-3. Upgrading a UML profiling project via the "Open an existing UML profiling project" window

Chapter 4: Using Objecteering/UML Profile Builder

Steps:

- 1 - Click on "*File/Open*". The "*Open an existing UML profiling project*" window will then appear.
- 2 - Double-click on the UML profiling project you wish to upgrade. Please note that UML profiling projects which have not yet been upgraded are listed in this window with the message "*to be migrated*" shown after their name.
- 3 - A dialog box then appears, telling you which version the UML profiling project is currently in, and asking you if you want to upgrade it. Click on the "*OK*" button.

The upgrade procedure is then automatically launched, and you can follow its progress in the console. This process can take a few minutes. Once completed, the UML profiling project is opened and is ready for use.

Note: When a UML profiling project is upgraded from a previous version of Objecteering/UML to the current version, the modules contained therein are not upgraded. To upgrade modules, the module upgrade procedure must be launched explicitly.

The main window

Main window

After creating the new UML profiling project (as shown in Figure 4-1), the main window (Figure 4-4) appears. The main window provides general services such as saving a UML profiling project, importing models from other UML profiling projects or managing other windows.

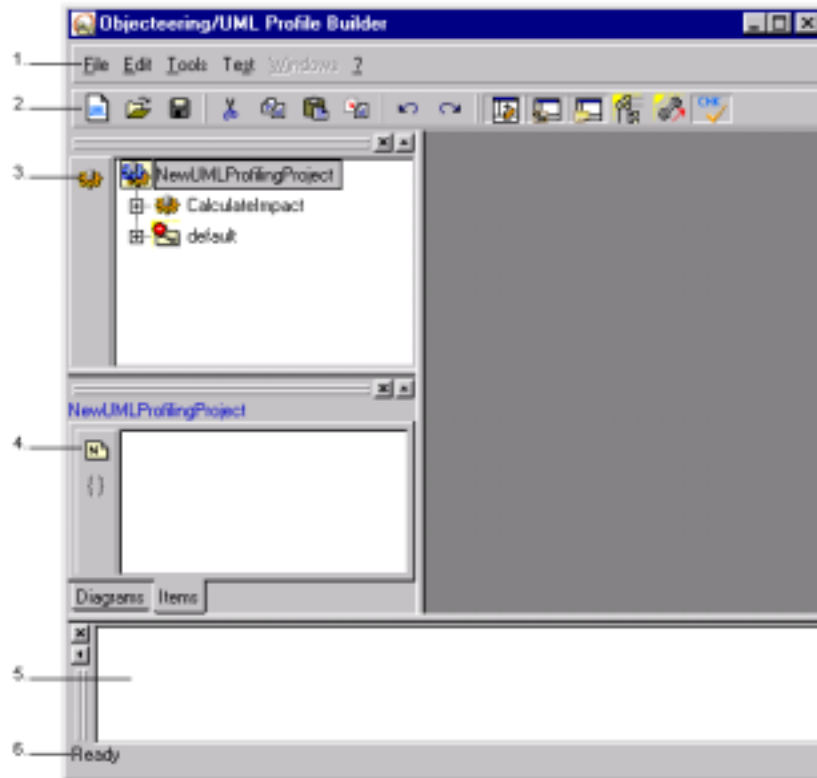


Figure 4-4. Main window for editing a UML profiling project

Chapter 4: Using Objecteering/UML Profile Builder

Description:

- 1 - Menu bar: this contains the "*File*", "*Tools*", "*Test*", "*Views*", "*Edit*" and "*Windows*" menus.
 - 2 - Tool bar: icons associated to certain elements in the menu bar appear here.
 - 3 - Meta-explorer: the meta-explorer contains the UML profiling project, and is used to browse the model and create and edit model elements.
 - 4 - Properties editor: this box contains a number of tabs, each containing information specific to a certain domain.
 - 5 - Console: this contains operation traces, error messages and warnings.
 - 6 - Status bar: this provides information complementary to that displayed by the bubbles which appear over the tool's icons.
-

The meta-explorer

Overview of the meta-explorer

The meta-explorer is a tool for editing:

- ◆ modules
- ◆ UML profiles
- ◆ J methods and J attributes
- ◆ types of notes, types of tagged values and stereotypes
- ◆ commands
- ◆ module parameters
- ◆ types of work products
- ◆ generation or document templates

Several meta-explorers can be open at the same time, which can be useful for browsing the different parts of a UML profiling project for example.

Launching a meta-explorer

Meta-explorers are launched from the "Tools" menu in the main window. A meta-explorer is automatically launched when the *UML Profile Builder* is started.

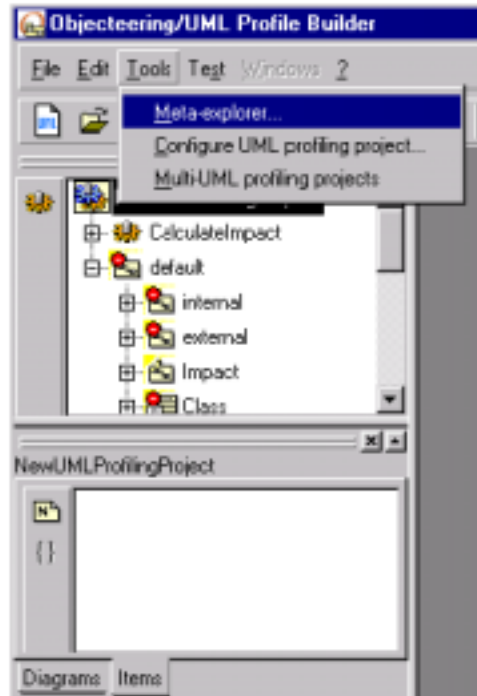


Figure 4-5. Opening a meta-explorer

Steps:

- 1 - Select the "Meta-explorer..." option in the "Tools" menu.

The meta-explorer

The icons which appear in the meta-explorer allow the creation of the different structural elements of a UML profiling project, according to the selected element. Here (Figure 4-6), only the creation of modules is possible. If you select a UML profile, other elements can be created.

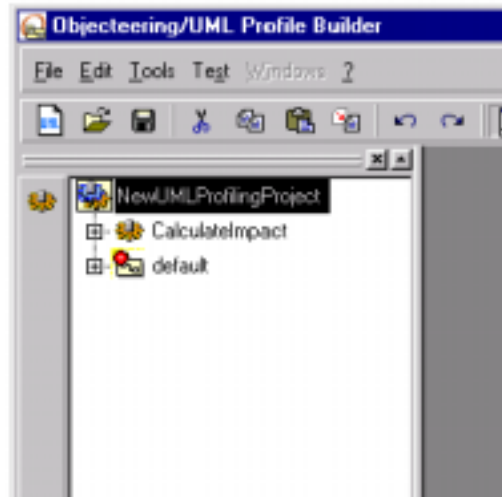





















Figure 4-6. The meta-explorer

Structural element creation icons in the meta-explorer

The icons shown in the table below are used to create structural elements in a UML profiling project.

The ... button	icon	is used to ...	in ...
Create a module		create a new module	a UML profiling project
Create a command		create a command	a module
Create a child UML Profile		create a child UML Profile	a UML profile
Create a metaclass reference		create a metaclass reference	a UML profile
Create a parameter		create a module parameter	a UML profile
Create a work product		create a product	a UML profile
Create a document template		create a document template	a UML profile
Create a generation template		create a generation template	a UML profile
Create a document item		create a document item	a document template
Create a generation item		create a generation item	a generation template
Create a J method		create a J method	a metaclass reference
Redefine a J method		add to a referenced metaclass, a method which redefines an operation of a parent UML profile and /or a parent metaclass	a metaclass reference

The ... button	icon	is used to ...	in ...
Create a J attribute		create a class attribute	a metaclass reference, a work product
Create a tagged value type		add a type of tagged value	a metaclass reference, a stereotype
Create a type of note		create a note type	a metaclass reference, a stereotype
Create a stereotype		create a stereotype	a metaclass reference
Add a parameter		add a parameter	a J method
Add a return parameter		add a return parameter	a J method
Add a constraint		add a constraint	a stereotype

The properties editor

Overview of the properties editor

The properties editor is a tool which contains a number of tabs, and which is used, in *Objecteering/UML Profile Builder*, to:

- ◆ annotate structural elements, through tagged values and notes
- ◆ specialize modules
- ◆ reference UML profiles
- ◆ use UML profiles
- ◆ select UML installation profiles
- ◆ create and redefine J methods

The properties editor

The icons which appear in the "Items" tab of the properties editor for a UML profiling project allow the creation of the different terminal elements of a UML profiling project, according to the selected element (as shown in Figure 4-7).

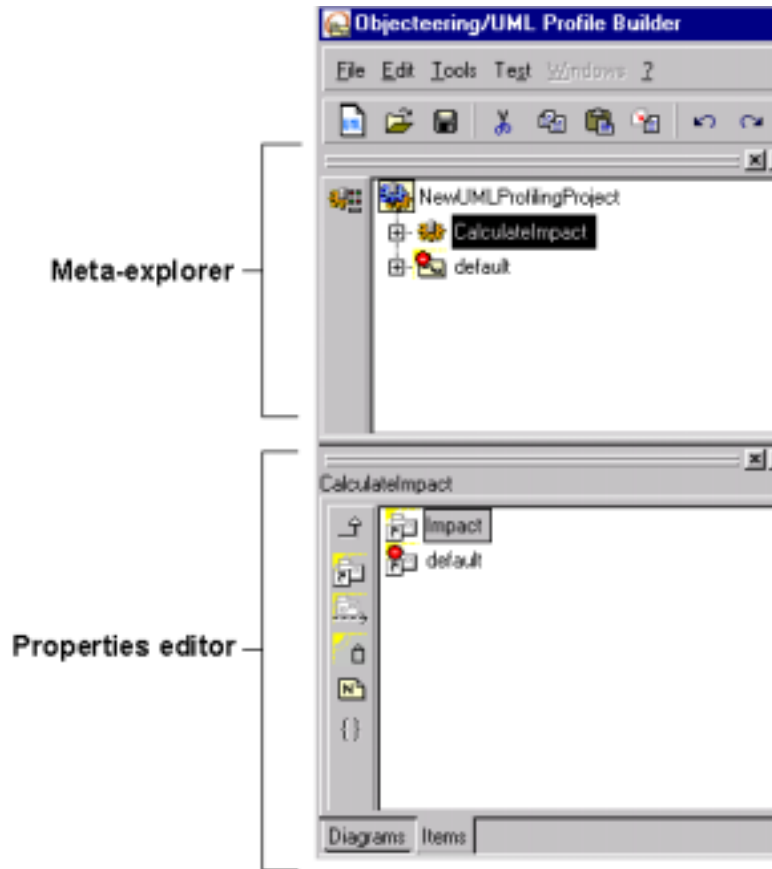









Figure 4-7. The properties editor for a module

Terminal element creation icons in the properties editor

The icons shown in the table below are used to create terminal elements in a UML profiling project.

The ... button	icon	is used to ...	in...
Add a note		add a note	all elements
Associate a tagged value		associate a tagged value	all elements
Specialize a module		specialize a module	a module
Reference a UML Profile		reference UML Profile of the current UML profiling project	a module
Use a UML profile		use a UML profile	a module
Installing a UML profile		select the installation profile	a module
Redefine a J method		add a parent method to a method or replace the existing one.	an operation

Chapter 5: Elements customizing UML and Objecteering/UML

Modules

Overview



Modules are functional subsets, which are selected at UML modeling project level. For example, a user can decide to use the C++ generation, Oracle generation, and documentation generation modules. In this way, his UML modeling project is configured to handle these specific targets:

- ◆ by providing the menus needed for each generator (notion of command)
- ◆ by providing a tagged value system on the model, in order to specify the characteristics of each part of the model (virtual method in C++, index for Oracle, etc.)
- ◆ by providing notes for each type of model element concerned by the target. For example, a "*description*" note is provided for classes, to generate documentation, and a C++ note is available for operations, to generate C++ code.
- ◆ by making it possible to create work products in the properties editor. Examples of work products are C++ code or documentation associated to a class or a package.
- ◆ by providing document templates, either for documentation or for code

UML encompasses the notion of extensibility mechanisms, which is supported here.

Using modules

Overview

Modules are delivered as .prof files and are located in the \$OBJING_PATH/modules, which is created during installation.

During the installation procedure, the user can select which modules he wishes to have installed on his site, simply by electing to carry out a "Custom" installation (please see Figure 2-8 and 2-9 in the "Single station installation in Windows" section in chapter 2 of the *Objecteering/Introduction* user guide) and by checking the tickboxes of the relevant modules. Once Objecteering/UML installation is complete, he simply has to select the module for the current UML modeling project in the "Modules" window (shown in Figure 5-1).

Modules developed by users can also be externalized in a directory, using the "Package" command, in order to be diffused to other sites. They must be installed in a site, before being accessible to user UML modeling projects.

Selecting modules in a UML modeling project

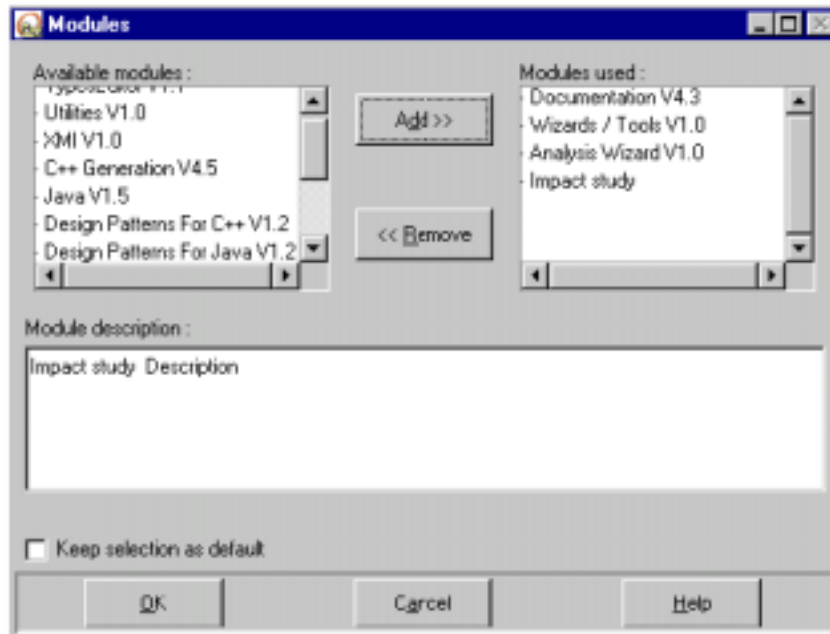


Figure 5-1. Window for selecting modules in Objectteering/UML

Modules are selected for a UML modeling project in the window shown above. They may also be selected through the "*Tools/Modules*" menu in the menu bar.

Note: For payable modules which use floating licenses, licenses are necessary at this stage.

Configuring modules

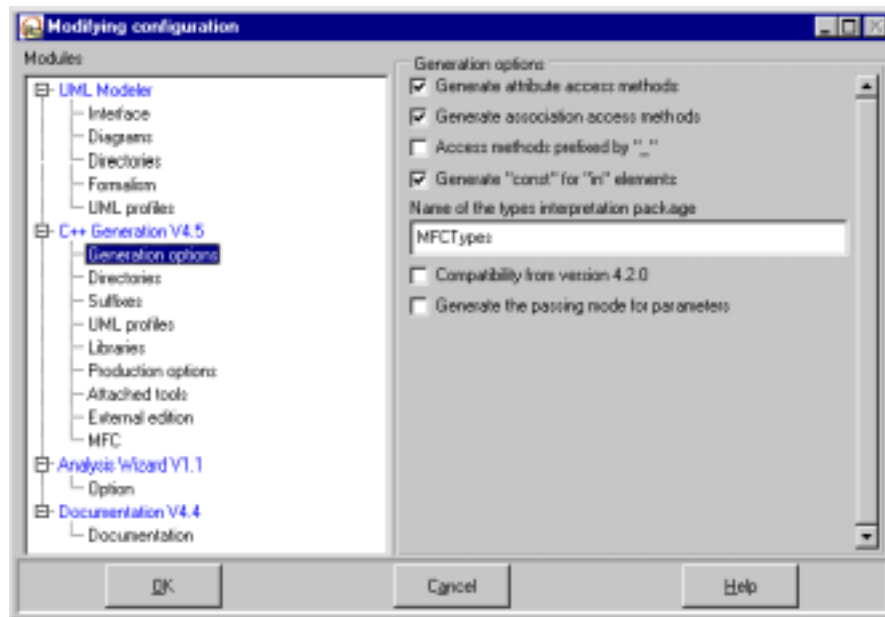


Figure 5-2. The "Modify configuration" window

In the Objecteering/UML Modeler tool, the "Modify configuration" menu allows the user to modify the module's parameters, which will define important options for the whole UML modeling project.

Objecteering/UML Profile Builder has been used to define these parameters, as well as their default values.

Access from the J language

Module parameters have a value which can be accessed in J using the `getCurrentModuleParameterValue` method.

Commands

Overview



Each module selected for a UML modeling project comes with a set of commands, each represented by a pop-up menu item available for the elements concerned (class, attribute, work product, etc). The selection of one of these menu items activates a J method defined in *Objecteering/UML Profile Builder*.

Example

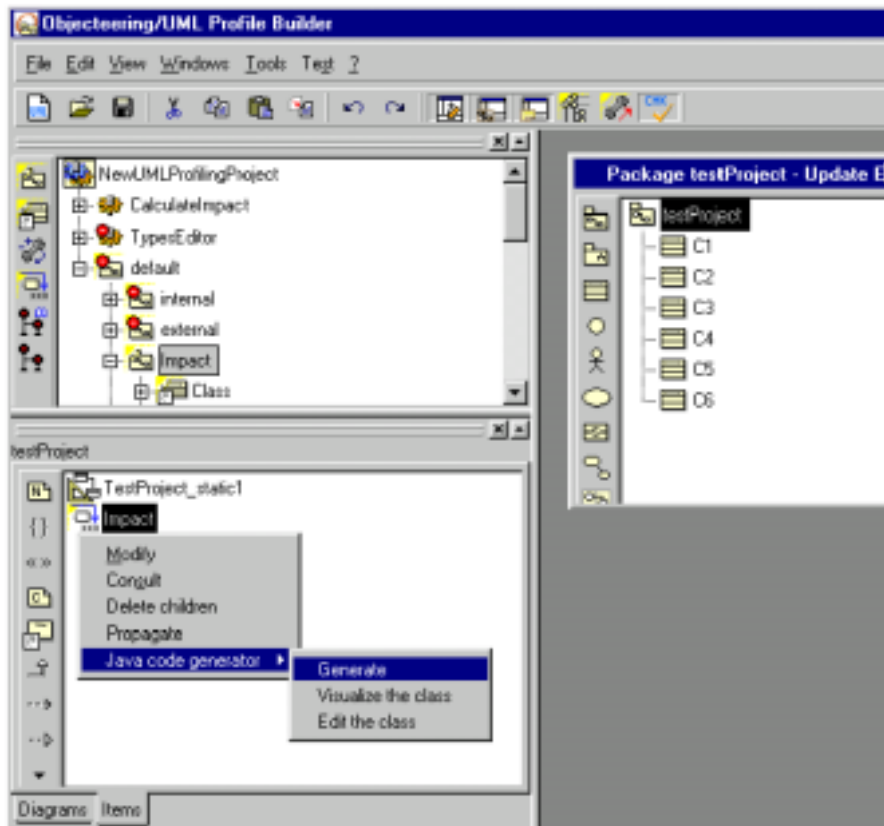


Figure 5-3. Commands associated with the "Impact analysis module" on an impact relation


When the "Generate" context menu item is selected, the impact report generation process is activated.

Access from the J language

In *Objecteering/UML Profile Builder*, each command is associated with a J method. This J method has to be public, and cannot have parameters. The metaclass it belongs to corresponds to the type of element on which the menu is available (in Figure 5-3, the element concerned is "*Class*").

Tagged values

Overview

 Any model element can be annotated using tagged values. The purpose of these tagged values is to bring a specific additional meaning to a model element. For example, Objecteering/UML provides as standard the *{primitive}* tagged value, used to designate primitive classes.

Objecteering/UML Profile Builder allows the definition of new tagged values, in order to give model elements a meaning that is specific to a module. For example, the *{virtual}* tagged value on an operation has a meaning for the C++ generator, whereas the *{persistence}* tagged value is used by the Oracle generator.

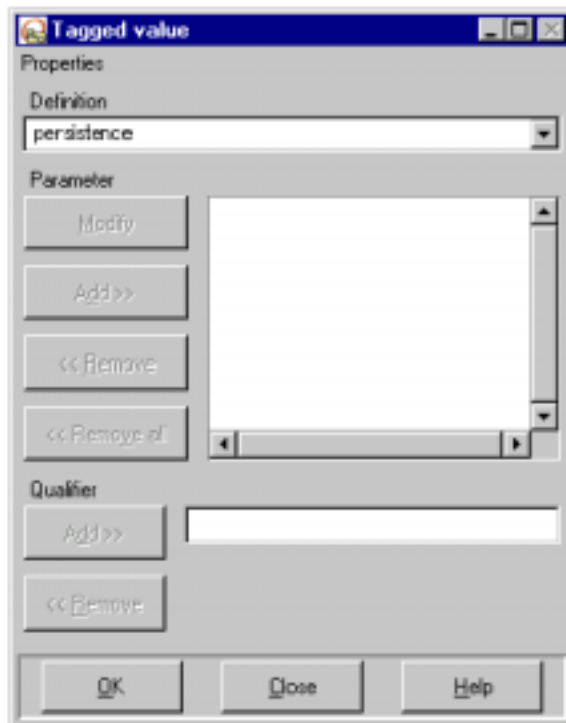
Example

Figure 5-4. Defining and visualizing tagged values (for example *{persistence}*)

Tagged values are visible in the "Item" tab of the properties editor and in editors with the *{tagged value}* notation, and can be entered from graphic editors or the properties editor.


Access from the J language

The Objecteering/UML metaclasses describe precisely how to access element tagged values (see the *Objecteering/Metamodel User Guide*, more specifically the "TaggedValue" and "TagType" metaclasses). In this way, for any element with a tagged value ("ModelElement" metaclass), the following J example lists the names of the associated tagged values:

```
TagTaggedValue
{
  StdOut.write(DefinitionTagType.Name, NL);
}
```

Notes

Overview

 Models entered in Objectteering/UML can be completed by text descriptions, which have a specific meaning for a specific module. These texts are, for example, descriptive texts used for documentation ("*description*") or C++ code used to implement a method ("C++"). *Objectteering/UML Profile Builder* allows you to define note types, aimed at providing new families of descriptions for new modules (for example, Java code, review descriptions, etc.).

Example

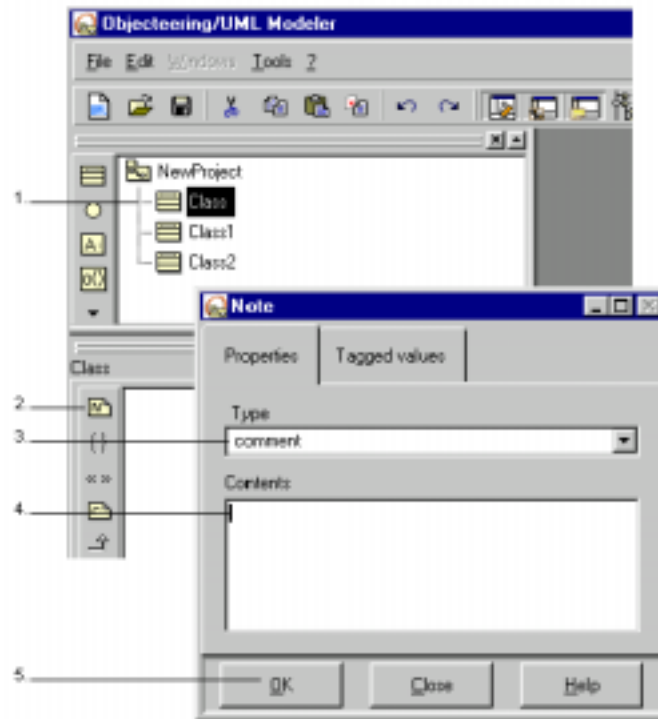


Figure 5-5. Example of entering a note for a class

Steps:

- 1 - Select the "Class" class.
- 2 - Create a note in this class in the "Items" tab of the properties editor.
- 3 - Choose the "comment" type from the scrolling list.
- 4 - Enter the text in the field.
- 5 - Confirm by clicking on the "OK" button.


Access from the J language

The Objecteering/UML metamodel describes precisely how to access the descriptions of an element. In this way, for any element with a note ("*ModelElement*" metaclass), the following J example displays the content of the note named "description":

```
DescriptorNote.<select(ModelNoteType.Name = "description")
{
  StdOut.write(Content, NL) ;
}
```

Stereotypes

Overview

 The UML defines stereotypes, which are used to extend the semantics of UML model elements. Stereotypes can have associated icons and are annotated <<Stereotype name>>. *Objectteering/UML Profile Builder* allows the creation of new stereotypes in UML profiles, related to metaclasses, which have a name and an associated icon.

Example

In this example, the <<interface>> stereotype is associated to the "Class" class.

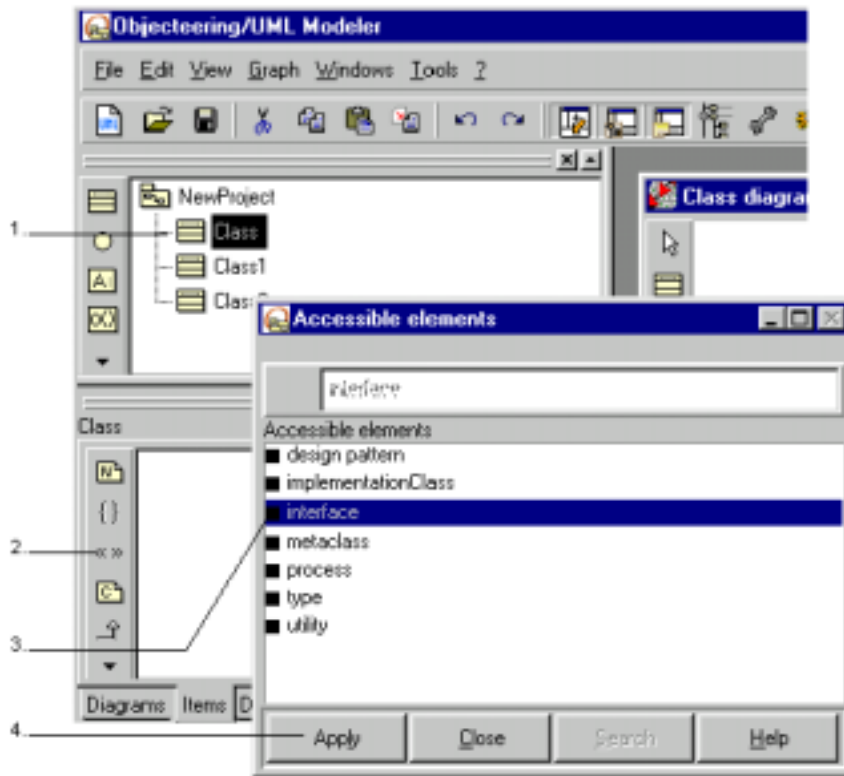



Figure 5-6. Creating a stereotype on a class

Chapter 5: Elements customizing UML and Objecteering/UML

Steps:

- 1 - Select the "C1" class in the explorer.
- 2 - Click on the  "Associate a stereotype" icon in the "Item" tab of the properties editor.
- 3 - Select a stereotype from the list.
- 4 - click on the "Apply" button.

The stereotype then appears in the "Item" tab.

Access from the J language

The Objecteering/UML metamodel describes exactly how to access an element's stereotypes. In this way, for any element with a stereotype ("*ModelElement*" metaclass), the following J example displays the name of this stereotype.

```
ExtensionStereotype
{
  StdOut.write (Name,NL);
}
```

Document and generation templates

Overview



("Create a document template")



("Create a generation template")

Document templates and generation templates are high level mechanisms used to describe the structure of a "target" (documentation, source code). They are made up of a hierarchy of document items or generation items. A specific editor is provided for templates, and mechanisms are provided to allow code to be easily generated.

Example

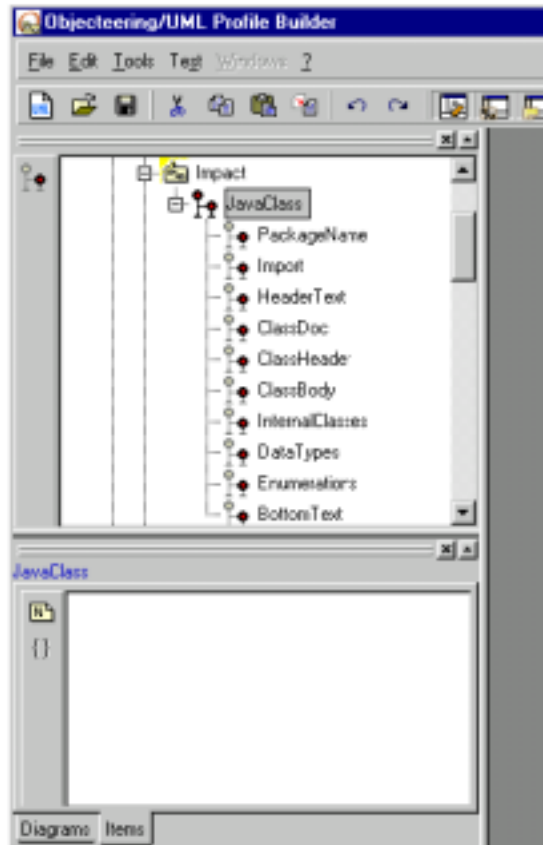


Figure 5-7. Java generation template

Java sources have a predefined structure (class declaration, method declaration, method implementation, etc), expressed in detail by this template. Java code generation is customized simply by editing this template.

Access from the J language

Document items and generation items provide connections to J methods, from which J processing can begin. Document templates are supported by a "template engine", which follows the template structure and connects to the related J methods.

- ◆ Connect a generation template to a generation work product ("*MyTemplate*"):

```
String name="MyTemplate";  
initTemplate (name);
```

- ◆ Launch generation of a generation work product driven by a template:

```
String result;  
result=generateWithTemplate ();
```

This code is valid in the context of generation work products, in a J method which includes a generation work product.

Work products

Overview



Work products represent external elements, generated by modules. Typically, these external elements are files, for example, C++ generated sources, production processes, binaries, documentation, and so on. Work products are represented by specific creation icons. Each work product has its own dialog box.

Objecteering/UML Profile Builder can be used to define new types of work products, and to associate icons, commands and attributes present in the work product's dialog box.

Example: "Java" work product

In this example, the user has created a Java work product in the "Java" tab of the properties editor. This work product represents Java sources generated for a class or a package.

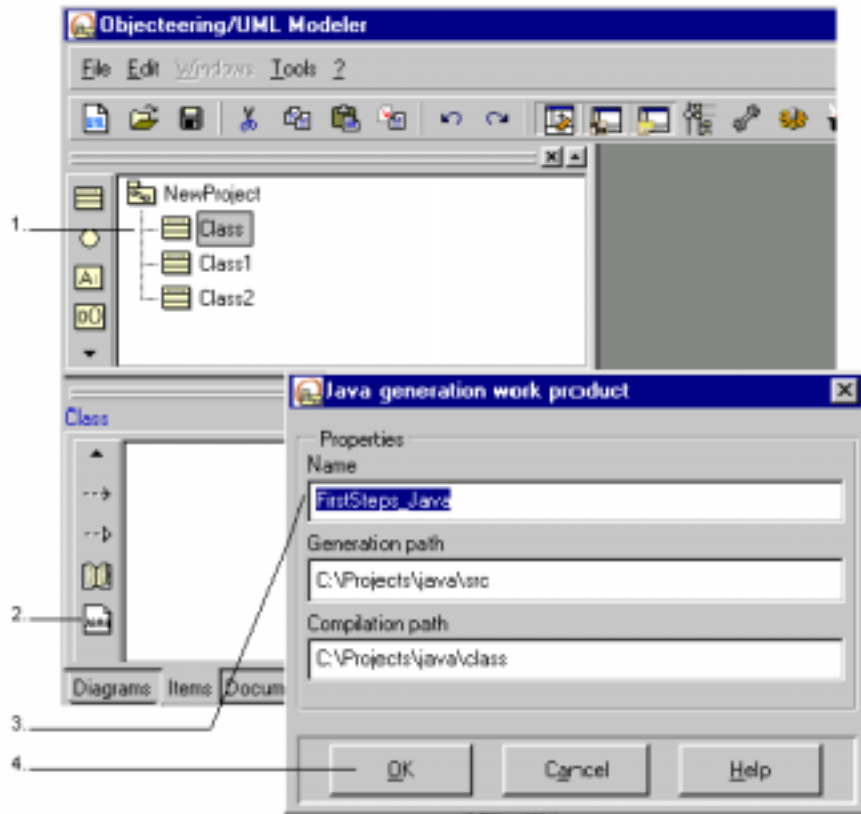



Figure 5-8. Creating a "Java" generation work product

Chapter 5: Elements customizing UML and Objecteering/UML

Steps:

- 1 - Select the "Class" class in the explorer.
- 2 - Click on the  "Create a Java work product" button in the "Items" tab of the properties editor.
- 3 - Enter the name of the work product.
- 4 - Confirm.

Access from the J language

The *Objecteering/The J Language* user guide supplies a set of primitives which allow you to use and manage work products ("MPGenProduct" metaclass).

Other customizable services

Overview

The J language supplies a set of tools to create additional services linked to modules. These consist, for example, of dialog boxes, external text editors added to the model, model transformation services, undo/redo operations on persistent transformation services, and so on.

External text entry: Example

The following example shows the editing of J method bodies from the "Impact" UML profile in *Objecteering/UML Profile Builder*.

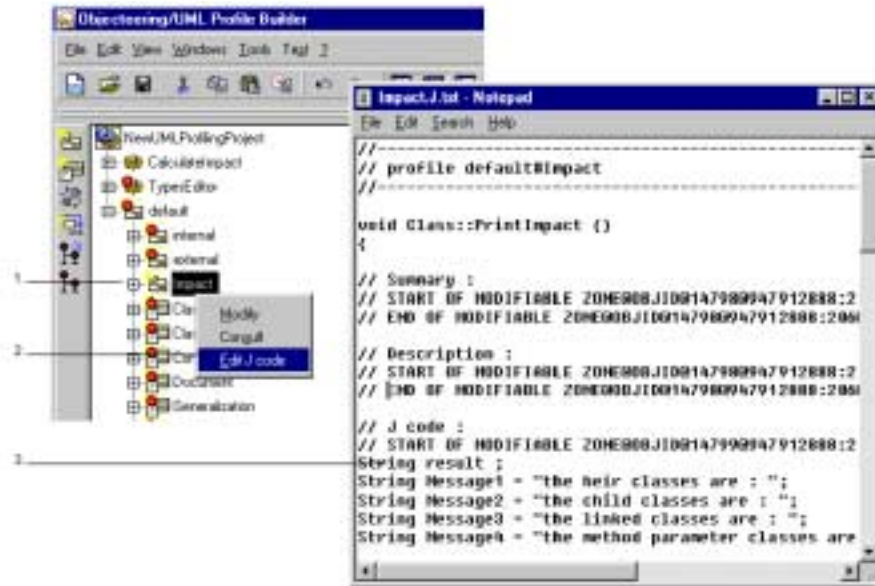


Figure 5-9. External text entry

Steps:

- 1 - Click on the "Impact" UML profile with the right mouse-button.
- 2 - Select the "Edit J code" option.
- 3 - Enter the text in the "Impact" window.

Note: The command for external edition can be customized. You can select "emacs", "notepad", or another editor.

J dialog box: Example

J provides primitives to create simple dialog boxes, such as the one presented in Figure 5-10. For further information on the creation of these dialog boxes, please refer to chapter 6, "*Dynamic dialog boxes*", of the *Objecteering/J Libraries User Guide*.

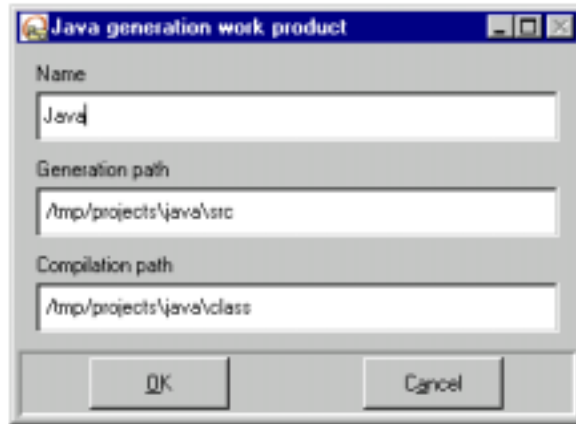


Figure 5-10. *Java generation work product* dialog box

Chapter 6: Defining UML profiles

Referencing a metaclass

Metaclass definition



A metaclass is a metamodel element, such as "*Operation*", "*Attribute*", "*State*", etc. On a metaclass, J methods, J attributes, note types, tagged value types and stereotypes are defined. The same metaclass can be referenced by several UML profiles.

Metaclasses are predefined in the Objectteering/UML metamodel (see the *Objectteering/Metamodel* user guide).

The "*Help*" service provided on metaclasses displays the Objectteering/UML metamodel (and the corresponding metaclass), which is an invaluable aid to J programmers.

Referencing a metaclass

By referencing a metamodel metaclass, only the metaclasses we wish to work on will be displayed in a UML profile. We are then able to define the J methods, the J attributes, the note types, the tagged value types and the stereotypes (see Figure 6-1).

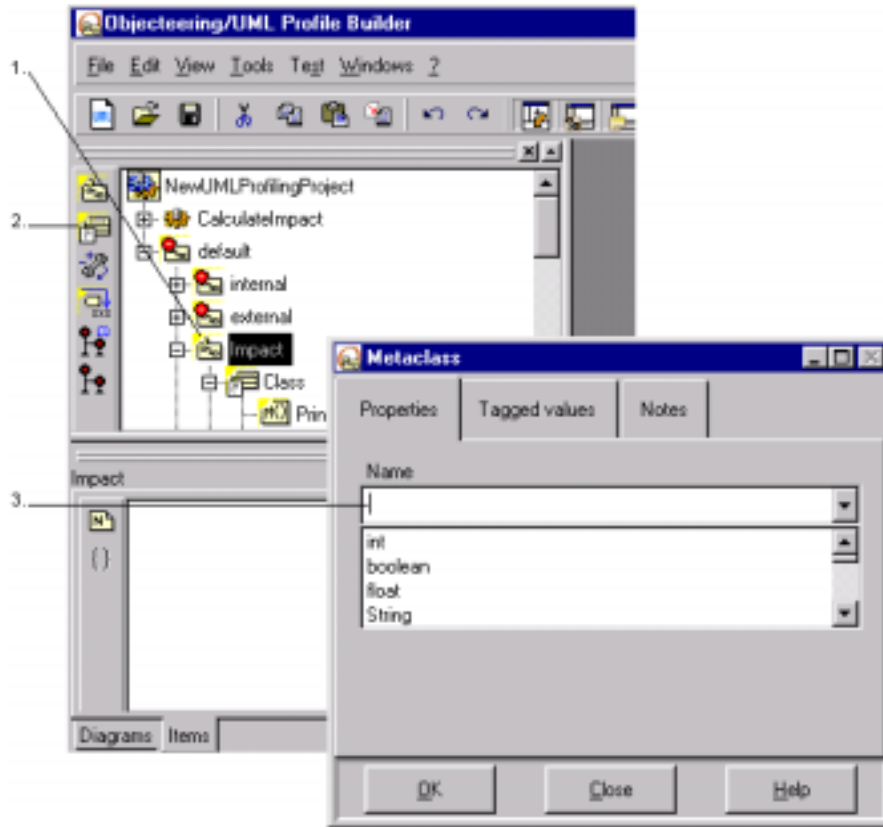



Figure 6-1. Referencing a metaclass

Steps:

- 1 - Select a UML profile.
 - 2 - Click on the  "Reference a metaclass" button.
 - 3 - Select the name in the combobox.
 - 4 - Confirm.
-

Creating a type of tagged value

Entering a type of tagged value



The *Objecteering/UML Profile Builder* tool can be used to define tagged value types on metaclasses. This defines possible tagged values for the representatives of this metaclass in a given model.

For example, due to the operations executed in Figure 6-2, all the UML modeling projects which use the "Impact" UML profile will be able to use the {persistent} tagged value on their model's classes.

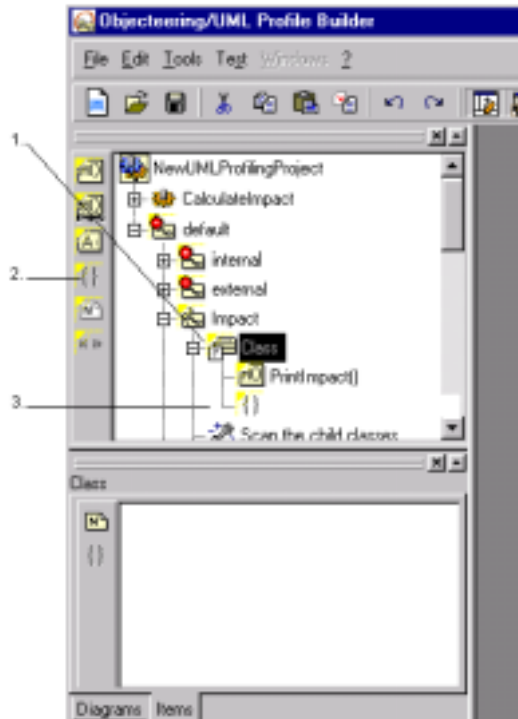



Figure 6-2. Entering a type of tagged value in the "Class" metaclass

Steps:

- 1 - Select the metaclass.
- 2 - Click on the  "Create a tagged value" button.
- 3 - Enter the name directly in the explorer and confirm by left-clicking.

Warning:

Please note that we strongly recommend against using the names of Objectteering Software modules as prefixes for your new tagged values, as this could cause name repetition problems. For example, the `{JavaName}`, `{JavaImport}` and `{JavaStatic}` tagged values are tagged values specific to the *Objectteering/Java* module.

Note: If you press the "Return" key on your keyboard, a new UML profile will be created (through the continuous entry creation mode).

The "Tagged value type" dialog box

When you select the "Modify" menu (using the right mouse button) on the "Tagged value type", the "Tagged value type" dialog box (shown in Figure 6-3) is opened.

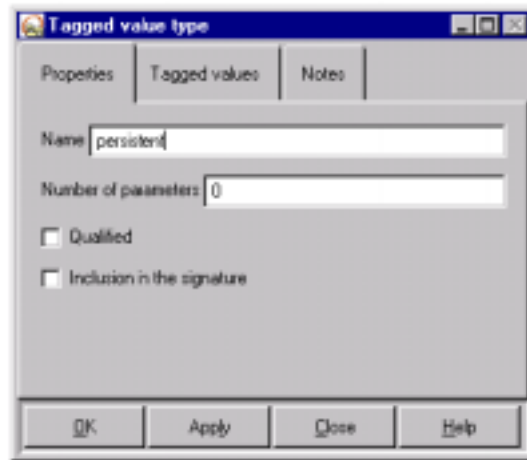



Figure 6-3. The "Tagged value type" dialog box

The ... field	represents ...
Name	the name of the corresponding tagged values.
Number of parameters	the number of parameters that the corresponding tagged values will have. This will be checked during a tagged value entry. In Objecteering/UML, tagged values can have parameters and a qualifier, which is a privileged parameter.
Qualified	whether or not the tagged value has a qualifier.
Inclusion in the signature	the taking into account of the tagged value by the Operation signature comparison algorithm. For code generation purposes, typically for C++, the presence of certain tagged values must be taken into account by the Operation signature comparison algorithm.

Creating a type of note

Entering a type of note

 With *Objectteering/UML Profile Builder*, you can also add note types to metaclasses. This allows you to create notes with these types on models.

For example, through the operations shown in figure 6-4, any UML modeling project which uses the "Impact" UML profile can use a text zone called "definition" for each class.

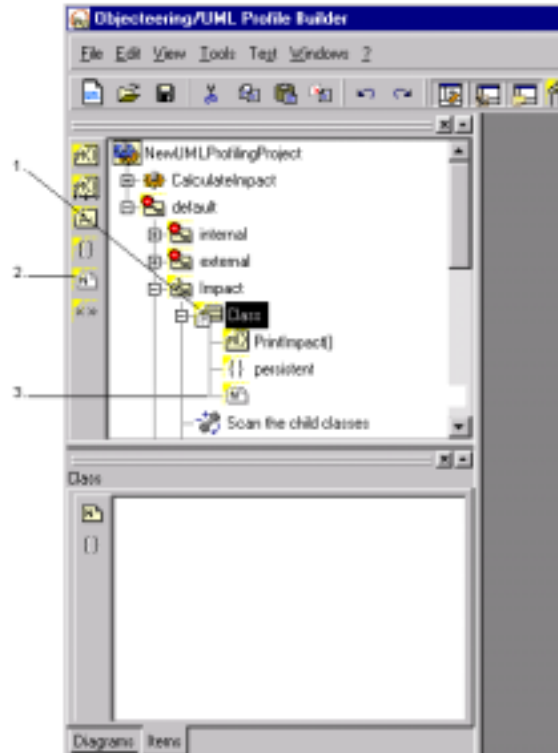



Figure 6-4. Entering a note type in the "Class" metaclass.

Chapter 6: Defining UML profiles

Steps:

- 1 - Select the metaclass.
- 2 - Click on the  "Create a type of note" button.
- 3 - Enter the "definition" name directly in the explorer and confirm by left-clicking.

If you press the "Return" key on your keyboard, a new UML profile will be created (through the continuous entry mode).

Note: A note type can only be entered in a metaclass reference.

The "Note type" dialog box

When you select the "Modify" menu (using the right mouse button) on the note type, the "Note type" dialog box is displayed (as shown in Figure 6-5).

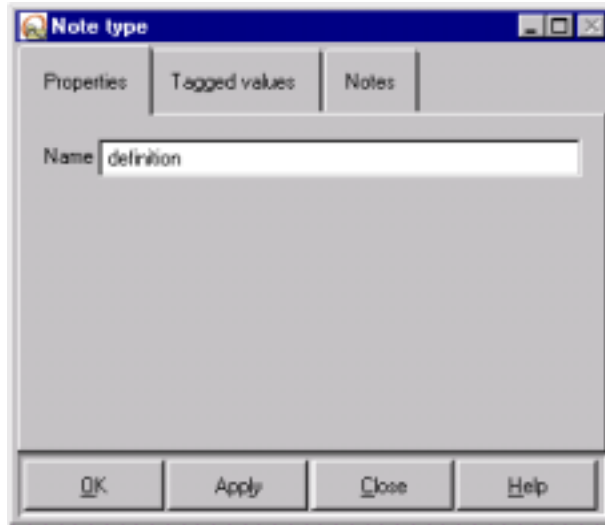



Figure 6-5. The "Note type" dialog box

The ... field	represents ...
Name	the name of the corresponding note.

Creating a stereotype

Entering a stereotype

 With the *Objectteering/UML Profile Builder* tool, you can also add stereotypes to metaclasses. This allows you to create stereotypes on models (see Figure 6-6).

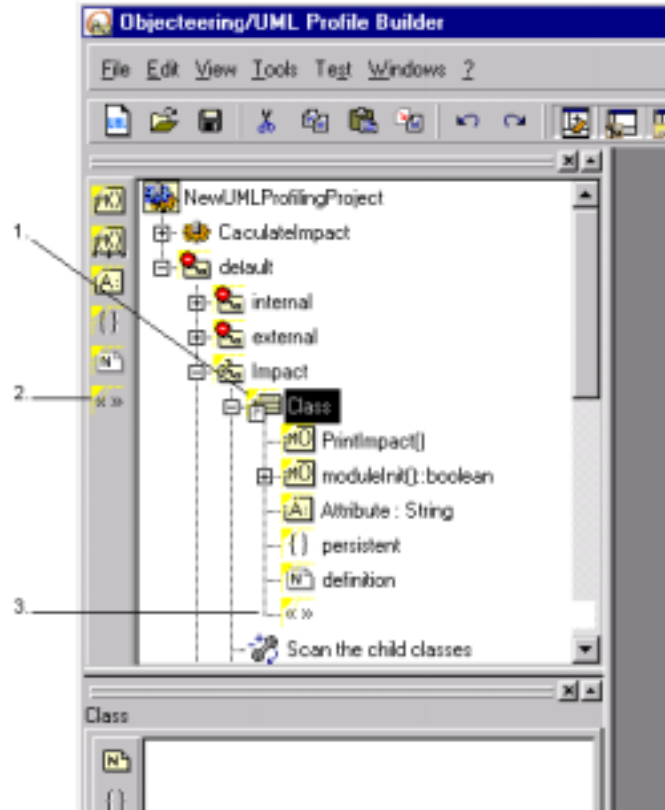



Figure 6-6. Entering a stereotype in the "Class" metaclass.

Steps:

- 1 - Select the embedding object.
- 2 - Click on the  "Create a stereotype" button.
- 3 - Enter the name directly in the explorer and confirm by left-clicking.

Note: If you press the "Return" key on your keyboard, a new UML Profile will be created (through the continuous entry creation mode).

The "Stereotype" dialog box

By selecting the "Modify" menu (using the right mouse button) on the stereotype, the "Stereotype" dialog box (shown in Figure 6-7) is opened.

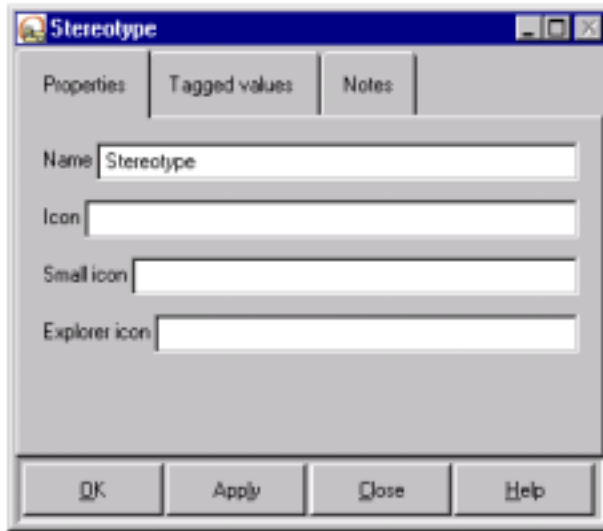



Figure 6-7 The "Stereotype" dialog box

The ... field	represents ...
Name	the name of the corresponding stereotype.
Icon	the name of the file containing the bitmap of the stereotype. It must be the full path name or a "relative" path name. In this case, the path will be calculated from the \$OBJING_PATH installation path. The format can be .gif or .bmp. When the icon is defined, it is this bitmap which appears in the graphic editors, to represent the stereotyped model element.
Small icon	the name of the file containing the small icon bitmap of the stereotype. When the small icon is defined, it is this bitmap which appears in the corner of the box representing the stereotyped model element.
Explorer icon	the name of the file containing the explorer icon bitmap of the stereotype. When the explorer icon is defined, it is this bitmap which appears in the explorer hierarchy, to represent the stereotyped model element.

Note: Please note that the icon, small icon and explorer icon features have been designed to help make stereotype visibility clearer.

Creating a constraint

Entering a constraint

 With the *Objecteering/UML Profile Builder* tool, you can also add constraints to metaclasses. This allows you to create constraints on models (see Figure 6-8).

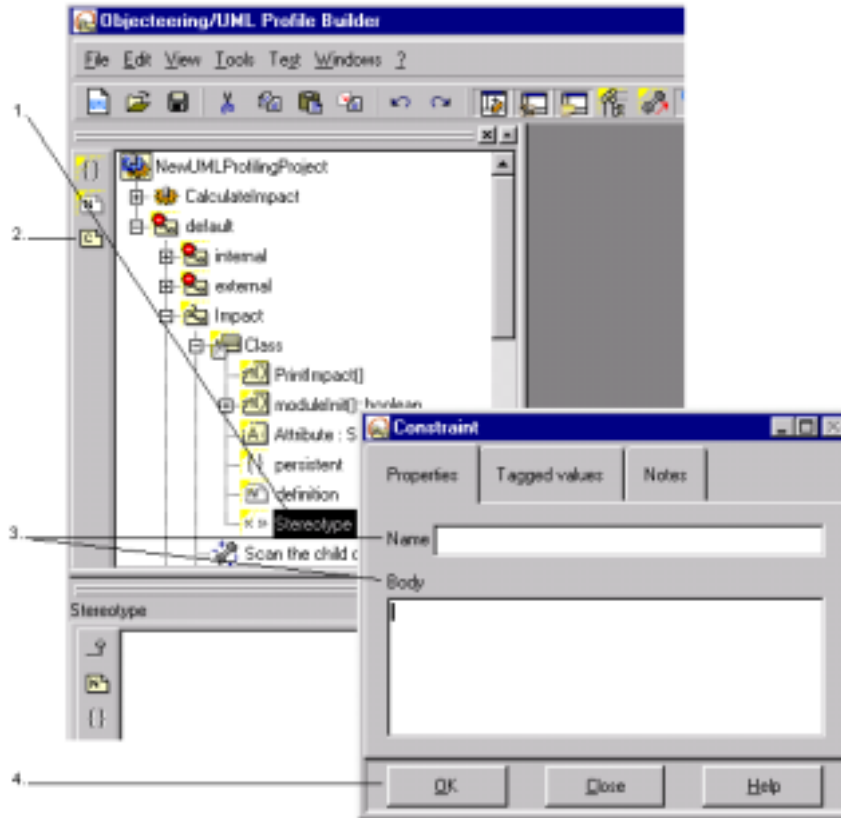



Figure 6-8. Entering a constraint in the "Stereotype" stereotype.

Steps:

- 1 - Select the embedding element.
- 2 - Click on the  "Create a constraint" button.
- 3 - Enter the name and the body text.
- 4 - Confirm by clicking on "OK".

The "Type of constraint" dialog box

The ... field	represents ...
Name	the name of the corresponding constraint.
Body	the body text associated with the constraint.

Creating a J class attribute

Definition



In this version, only "class" meta-attributes can be created. Instance meta-attributes are predefined by the Objecteering/UML metamodel. Class meta-attributes are used by the J methods that are defined on the same metaclass. They are not persistent (their values that can be modified in J are not stored in a model).

Procedure

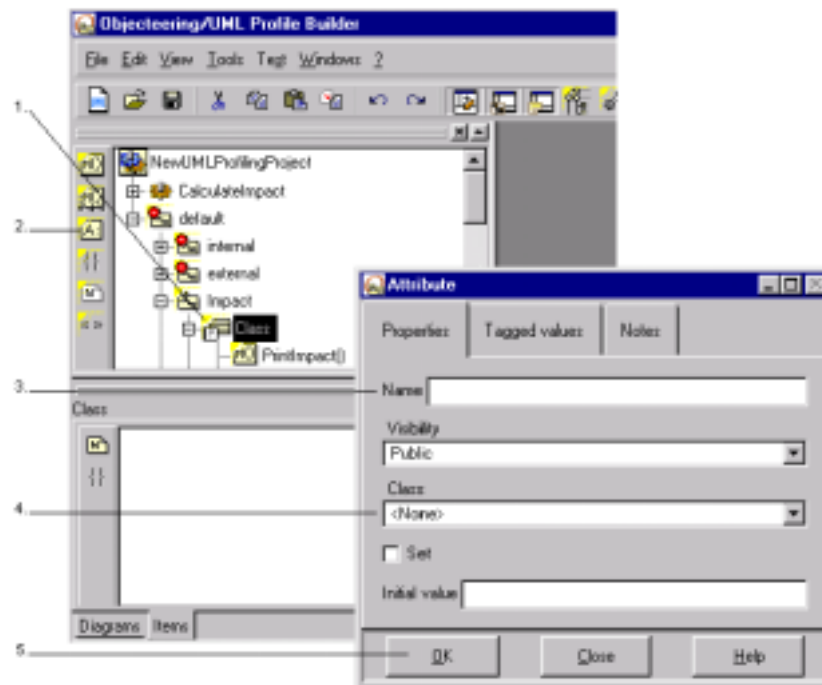



Figure 6-9. Creating a meta-attribute

Steps:

- 1 - Select the metaclass.
- 2 - Click on the  "Create a J attribute" button.
- 3 - Enter the name.
- 4 - Select the metaclass that gives the attribute type.
- 5 - Confirm.

The "Attribute" dialog box

The ... field	represents ...
Name	the attribute name.
Visibility	the attribute visibility.
Class	the attribute type.
Set	whether or not the attribute is a set.
Initial value	This field is left blank.

Creating a J method

Definition



The J methods defined in *Objectteering/UML Profile Builder* are defined:

- ◆ in a UML profile
- ◆ on a metaclass

J methods are written using the J language (for further information, please refer to the *Objectteering/The J Language* user guide).

Procedure

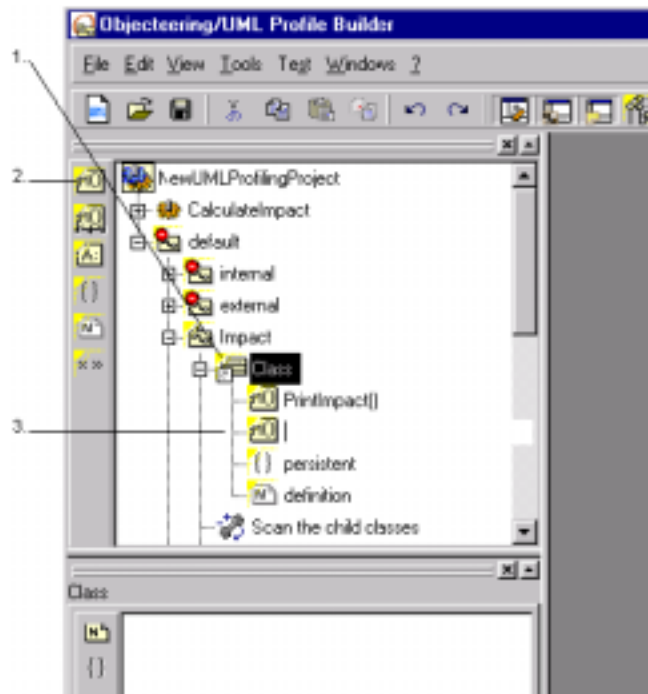



Figure 6-10. Creating a J method

Steps:

1 - Select a metaclass reference.

2 - Click on the  "Create a J method" button.

3 - Enter the name directly in the meta-explorer and confirm by left-clicking.

Note: If you press the "Return" key on your keyboard, a new UML profile will be created (through the continuous entry creation mode).

The "J Method" dialog box

By selecting the "Modify" menu (using the right mouse button) on the J method, the "J Method" dialog box is displayed (Figure 6-11).

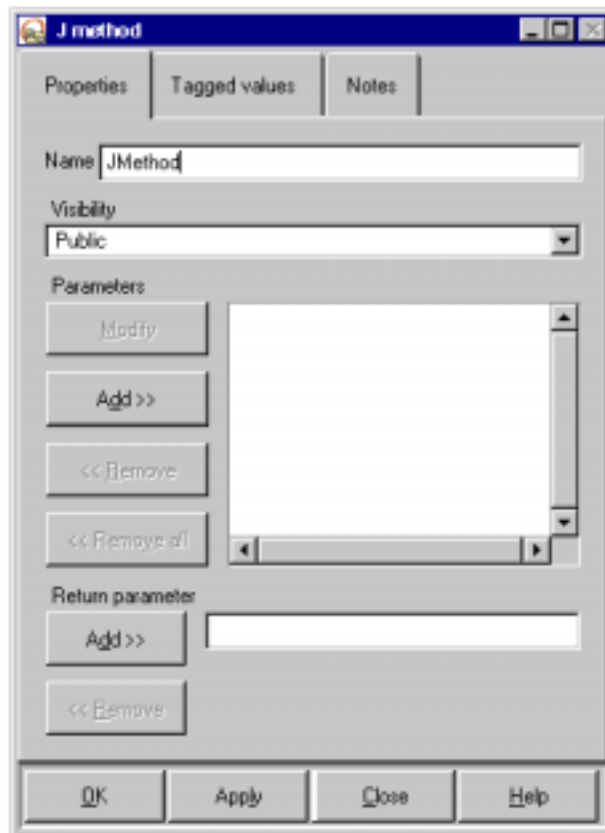


Figure 6-11. The "J Method" dialog box

The ... field or button	is used to ...
Name	enter the command name.
Visibility	select the visibility of the method (public, protected, private, none). A public method without parameters can be referenced by a command. A private method cannot be redefined in a sub UML Profile.
Parameters	create the method's parameters.
Return parameter	create the method's return parameter.

Textually editing a J method

Overview

J code is entered in "*J code*" type notes which are associated to J methods. It is possible to edit the code, either by double-clicking on the text zone, or using the external editors, by selecting the "*Edit the J code*" menu on a UML profile.

Editing a J method

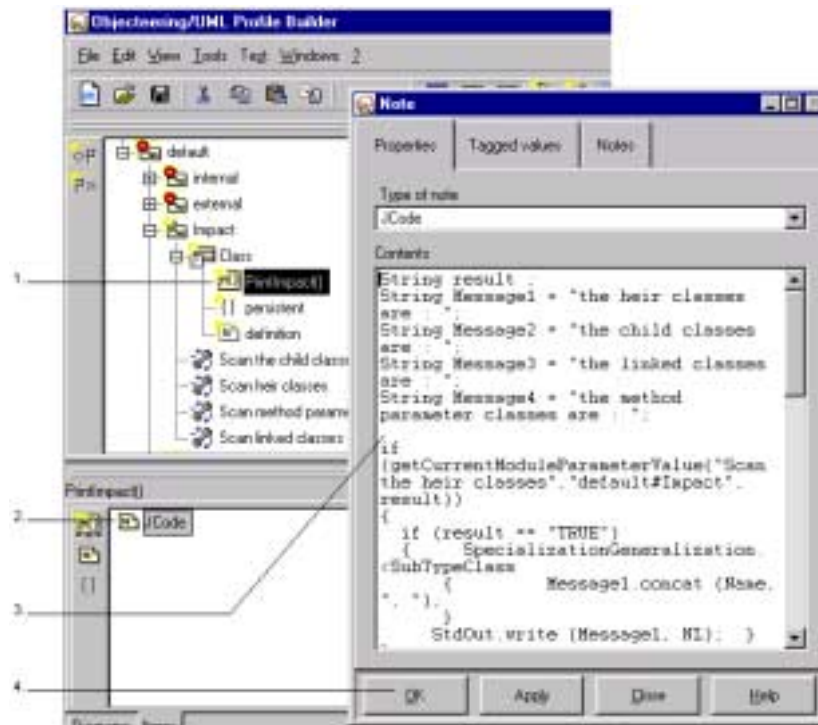


Figure 6-12. Editing the "moduleInit()boolean" J method

Steps:

- 1 - Select the J method.
- 2 - Double-click on the note.
- 3 - Enter the text in the dialog box which appears.
- 4 - Confirm.

External edition

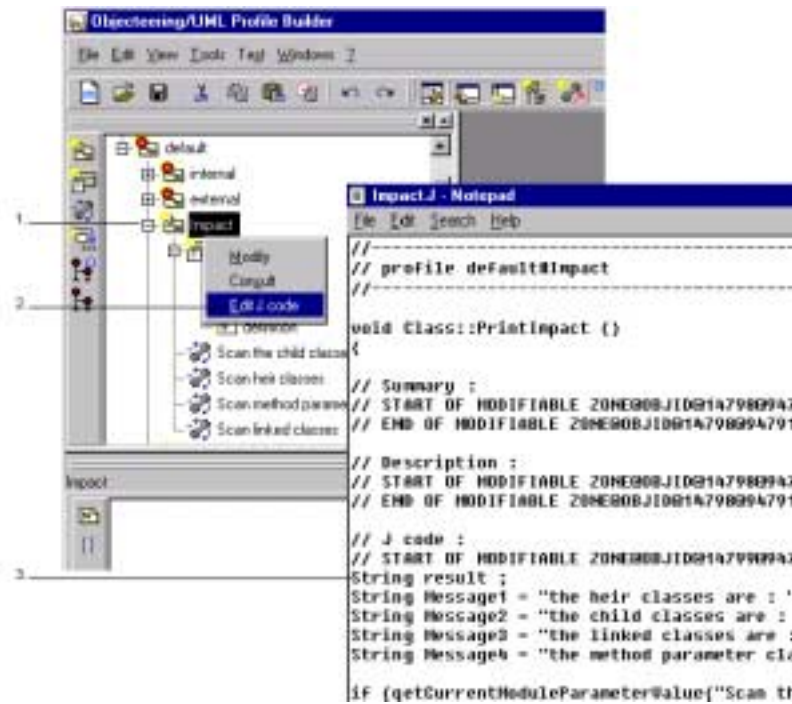


Figure 6-13. Editing the "PrintImpact()Boolean" J method


Steps:

- 1 - Select the UML profile containing the J method using the right mouse-button.
- 2 - Select "Edit J code".
- 3 - Enter the code in the zone that can be modified.

Note: When the editor is closed, only the text located between a start marker and its corresponding end marker will be taken into account.

Redefining a J method

Definition

The  "Redefine a J method" icon allows the redefinition of a J method coming from the metaclass or from a parent metaclass in a parent UML profile. A help list then appears, allowing you to select the method to be redefined. Only public or protected methods are displayed.

Procedure

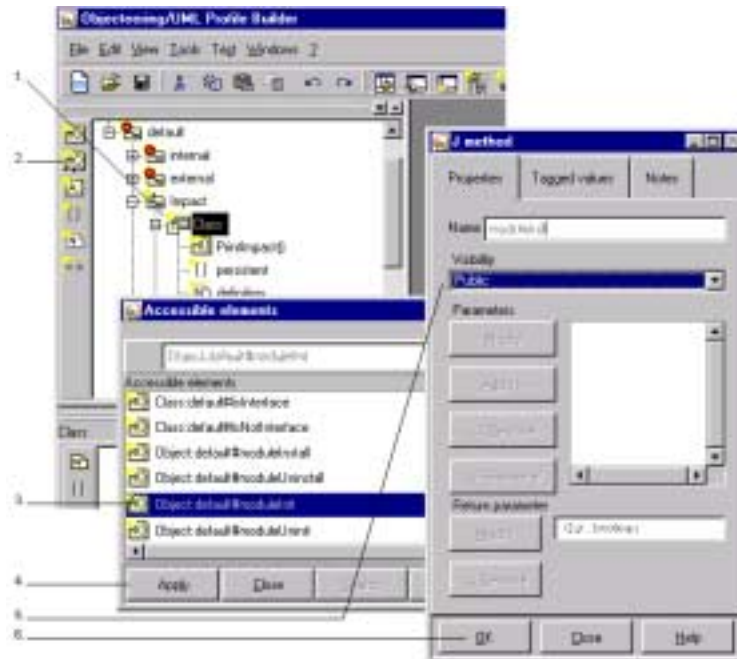



Figure 6-14. Redefining a parent J method

Steps:

- 1 - Select a metaclass reference created in a child UML profile.
- 2 - Click on the  "Redefine a J method" button.
- 3 - Select a method.
- 4 - Click on "Apply".
- 5 - Change the visibility of the operation in the dialog box which appears.
- 6 - Confirm.

Result

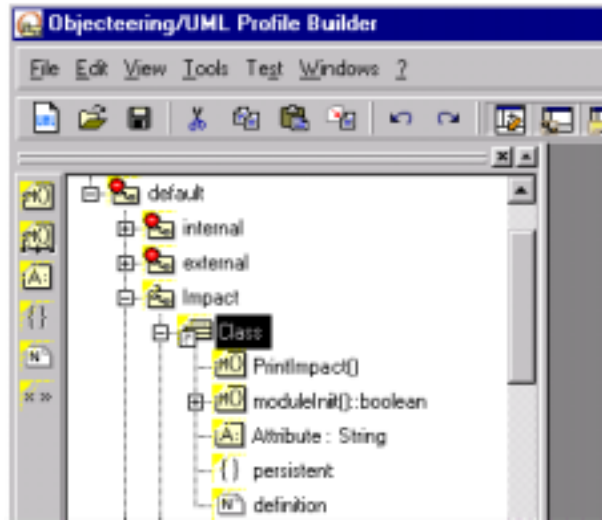


Figure 6-15. Result of the redefinition of the "moduleInit()Boolean" parent J method

Defining a module parameter

Procedure



The end user of a UML modeling project can access module parameters that have already been defined in the "Module Configuration" window, and select options for the module. Their values can be accessed through J rules with the "getCurrentModuleParameterValue" operation.

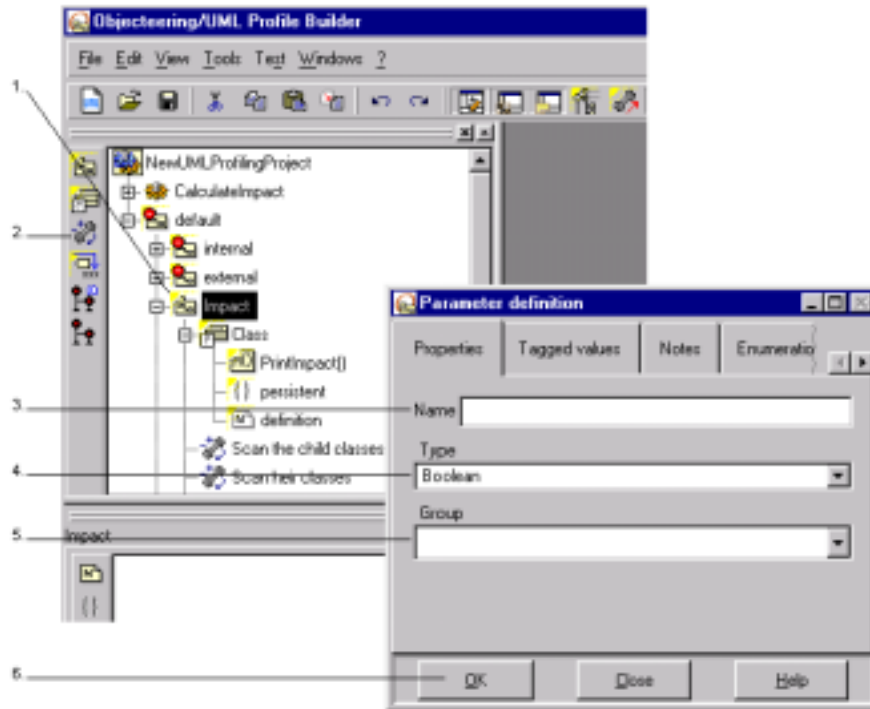



Figure 6-16. Creating a parameter




Chapter 6: Defining UML profiles

Steps:

- 1 - Select the UML profile.
- 2 - Click on the  "Create a parameter" button.
- 3 - Enter the name.
- 4 - Choose the type.
- 5 - Choose the group.
- 6 - Confirm.

Note: You can either select an existing group or type a new group name.

The "Parameter" dialog box - "Properties" tab

The ... field	is used to ...
Name	enter the parameter name.
Type	<p>select the parameter type. Available parameter types are:</p> <ul style="list-style-type: none">- Boolean, to create a tickbox parameter- String, to create a text field parameter.- Enumeration, to create a dropdown .enumerate selection box parameter- File open, to create a text field .parameter, accompanied by the  icon, used to display the "Open" window- File save, to create a text field parameter, accompanied by the  icon, used to display the "Save" window- Directory, to create a text field .parameter, accompanied by the  icon, used to open a file browser- Password, to create a text field parameter, in which .characters entered are represented by asterisks, thereby remaining hidden
Group	select the group in which the parameter appears in the "Edit the Configuration" window. This window can be accessed from the "Configure the UML profiling project" item in the "Tools" menu.

The "Parameter" dialog box - "Enumeration Values" tab

When the type is "Enumeration", this tab is used to enter the field's literal values.



Figure 6-17. "Enumeration Values" tab

Creating a generation or document template

Overview



("Create a generation template")



("Create a document template")

Document templates are used to describe target generation, by defining their text structure in a hierarchy. This kind of generator definition reduces the volume of J programming necessary. Any target which has a text format (C++, SQL, Java, makefiles, XMI etc.) can thus be described.

The parameterization of the makefiles generation, described in *the Objecteering/C++* user guide, provides a significant example of the use of generation document templates.

Documentation is a special case of generation. It is similar to the other generation operations, but its formatting constraints (RTF, HTML, etc.) and graphical formats make it different. A specific document template exists for documentation generation. This document template is described in the *Objecteering/Document Template Editor* user guide.

Creating a generation template

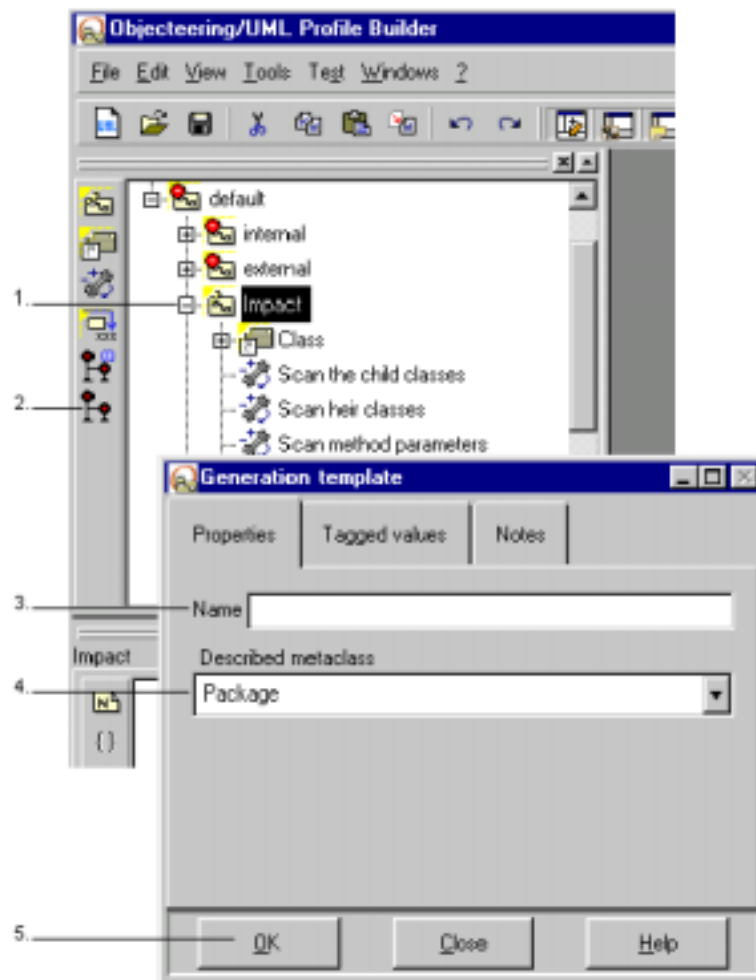



Figure 6-18. Creating a generation document template

Chapter 6: Defining UML profiles

Steps:

- 1 - Select a UML profile.
- 2 - Click on the  "Create a *generation template*" button.
- 3 - Enter the name of the document template.
- 4 - Enter the metaclass (the field corresponds to the type of root objects of a generation that uses this document template).
- 5 - Confirm.

Creating a generation item



Generation items are elements of the document template tree. Each represents a file zone to be produced. An item can have sub-items, that correspond to the organization of its zone into sub-zones.

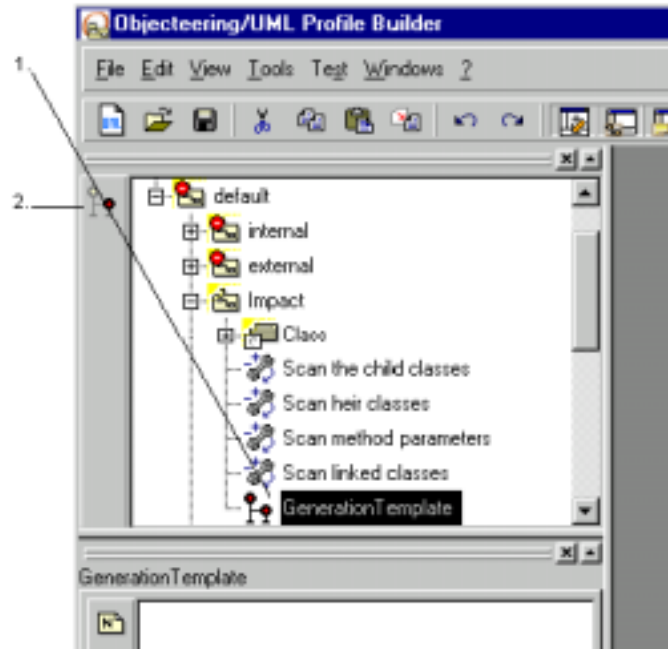



Figure 6-19. Creating a generation item

Steps:

- 1 - Select the generation document template.
- 2 - Click on the  "Create a generation item" button.

The "Generation item" dialog box

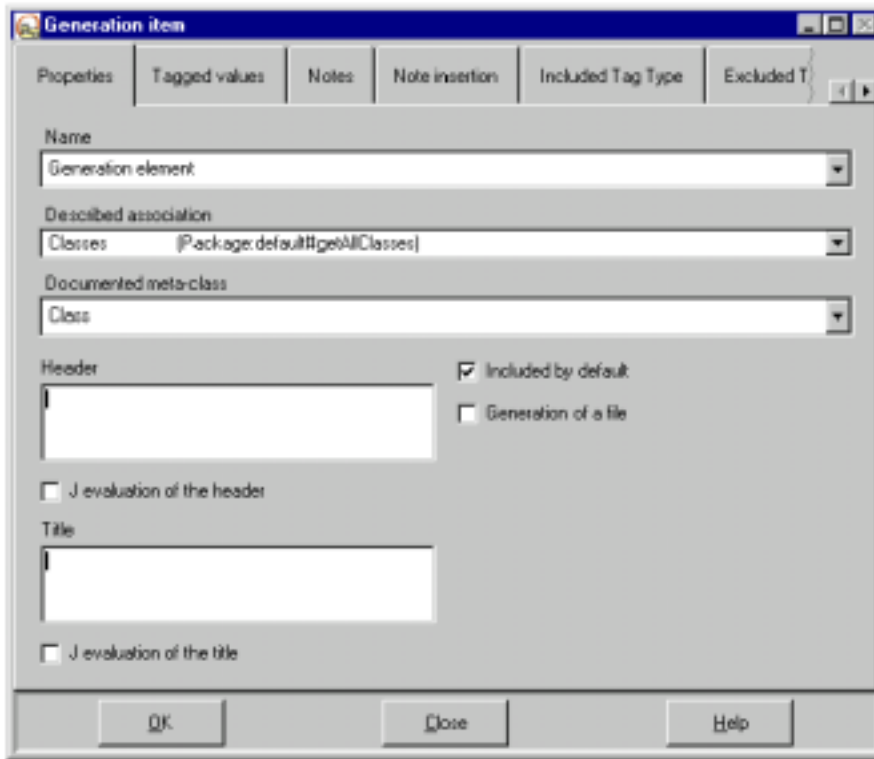



Figure 6-20. Generation item dialog box

The ... field	corresponds to ...
Name	the name of the item.
Described association	a journey through a model, for example going to the methods from a class.
Documented meta-class	the metaclass which the element concerns. A default value is calculated. You may modify it and choose a parent metaclass to generalize your item.
Header	the text inserted before the generation of all the modeling elements the element browses.
J evaluation of the header	the header. The header can be a J expression to be evaluated dynamically, instead of immediate text.
Title	the text inserted before the generation of all the modeling elements the element browses.
J evaluation of the title	the fact that the title can be a J expression, to be dynamically evaluated, instead of immediate text.
Included by default	By default, all the elements will be included. If not checked, only those that have the tagged values selected in the "Inclusion" tab will be included.
Generation of a file	If checked, a separate file will be generated for every included element..

Creating a document template

 ("Create a document template") This button is used to edit the document template dialog box, through which document templates are created.

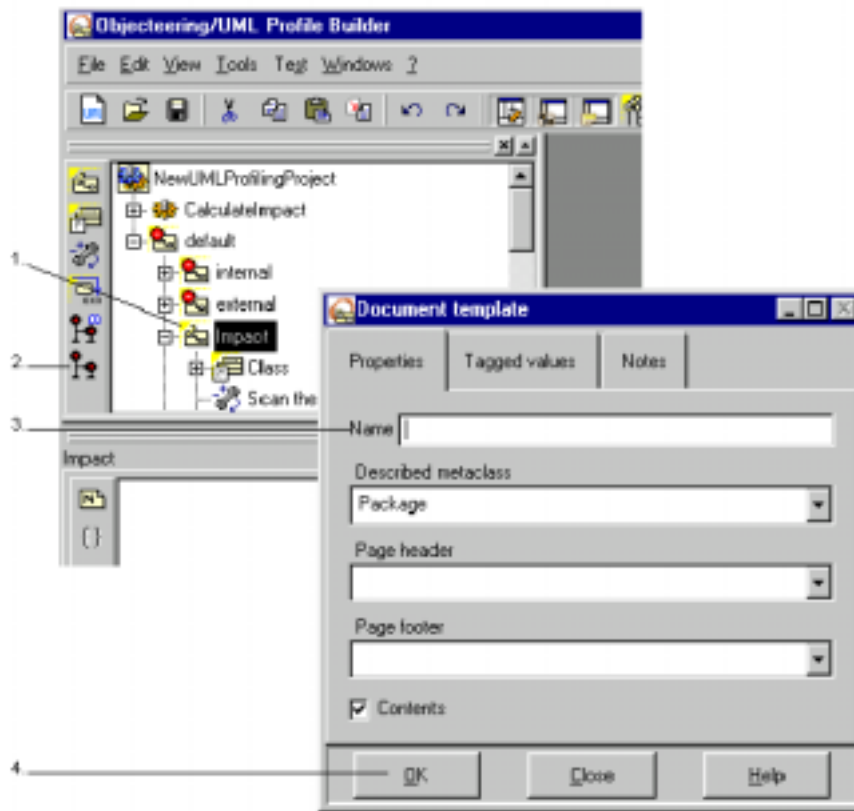



Figure 6-21. Creating a document template

Steps:

- 1 - Select a child UML profile.
- 2 - Click on the  "Create a document template " button.
- 3 - Enter the fields (see table below).
- 4 - Confirm.

The ... field	corresponds to ...
Name	the name of the element
Described metaclass	a journey through a model, for example going to the methods from a class
Page header	the text at the top of the page
Page footer	the text at the bottom of the page

Creating a document item



Document items are the components of the document template. They describe information that must be found in the generation target. Document items can be:

- ◆ a summed-up description of operations on a class
- ◆ the detailed description of the operation

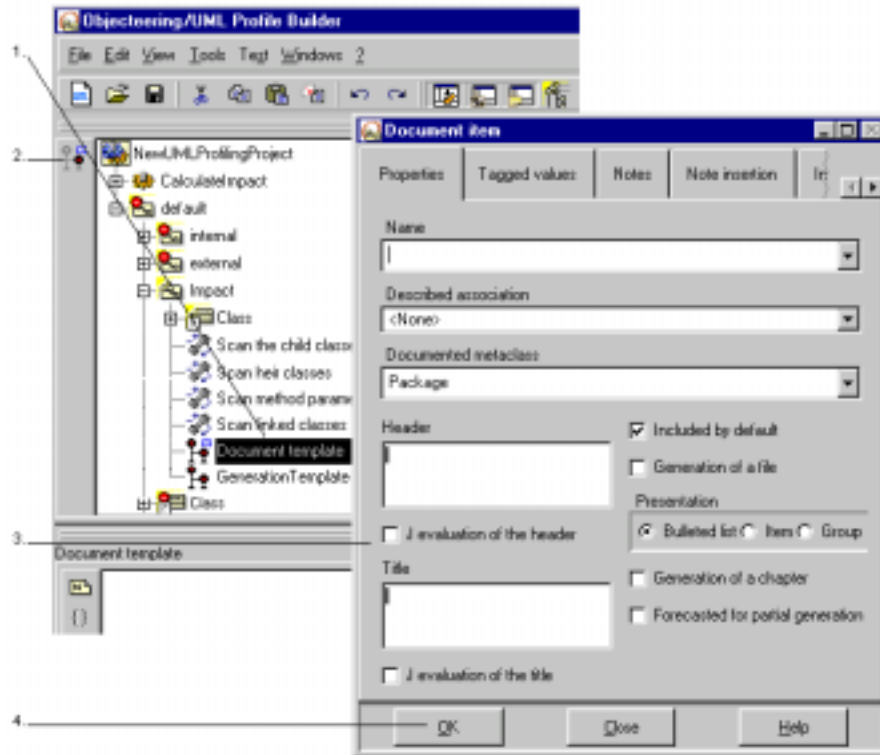



Figure 6-22. Creating a document item

Steps:

- 1 - Select the document template.
- 2 - Click on the  "Create a *document item*" button.
- 3 - Enter the fields.
- 4 - Confirm.

Possible actions on a module



From a module, it is possible to:

- ◆ reference a UML profile
- ◆ use a UML profile
- ◆ select the installation UML profile
- ◆ create a command
- ◆ inherit from a module
- ◆ package the module

The default configuration of a module can be entered through the "*Configure the UML profiling project*" item in the "*Tools*" menu.

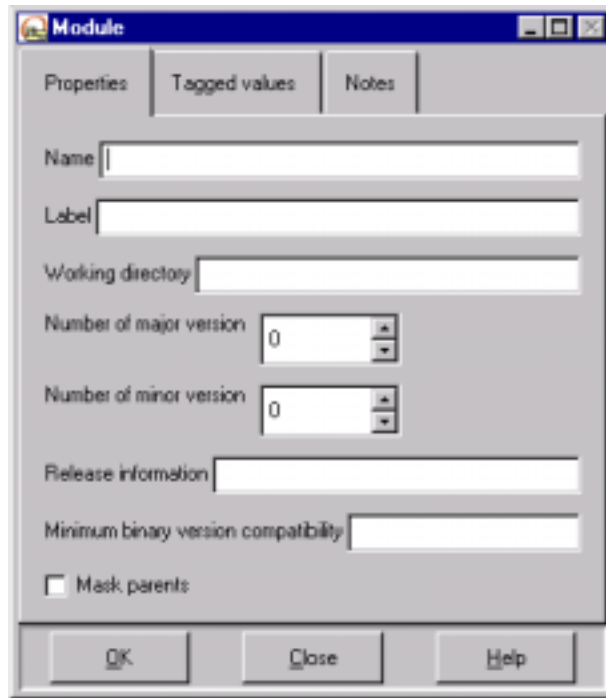


Figure 6-23. "Module" dialog box

The ... field or button	is used to ...
Name	enter the module name, used internally (in the meta-explorer)
Label	enter the module name, in the interfaces
Working directory	enter the packaging target directory and the directory containing the module's external resources. If the " <i>Working directory</i> " field is left blank, then the working directory is \$OBJING_PATH/modules/<ModuleName>/<Version>. Otherwise, the user may specify the working directory of his choice.
Major version number	indicate the first number in the version number (for example, if the complete version number is "4.6", then the major version number is 4).
Minor version number	indicate the second number in the version number (for example, if the complete version number is "4.6", then the minor version number is 6).
Release information	indicate the release number for the module (for example, "a").
Minimum binary version compatibility	indicate Objectteering/UML binary requirements for the module.
Mask parents	mask the parameters and menus of the parent of the module which has been selected by the UML modeling project. If this box is not checked, the module's parameters and menus will cohabit with those of its parents. This box has no effect on modules which do not have parents.

Defining a new kind of work product

Definition



Objectteering/UML Profile Builder allows you to define new kinds of work product. A meta-product corresponds to a new metaclass created dynamically (contrary to other metaclasses, predefined by the *Objectteering/Metamodel*), with the following characteristics:

- ◆ attribute types are constraints (string and boolean)
- ◆ the meta-product specializes "*MpGenProduct*"
- ◆ an icon can be associated to it
- ◆ one or more metaclasses can be associated to it, on which the work products can be created.

Work products represent the external result of generation in the *Objectteering/UML Modeler* workshop. For example, they can represent generated C++ sources, makefiles or documentation. Work products are visible in the explorer, and are associated to a model element. A creation icon for this element will appear, and its attribute dialog box will be automatically defined.

When a product is defined in a UML profile, it is possible to create a meta-class reference on the product in the UML profile and its child. We can then create J methods and then commands on the modules which reference the UML profile, where the work product or one of its children is defined.

Creation procedure

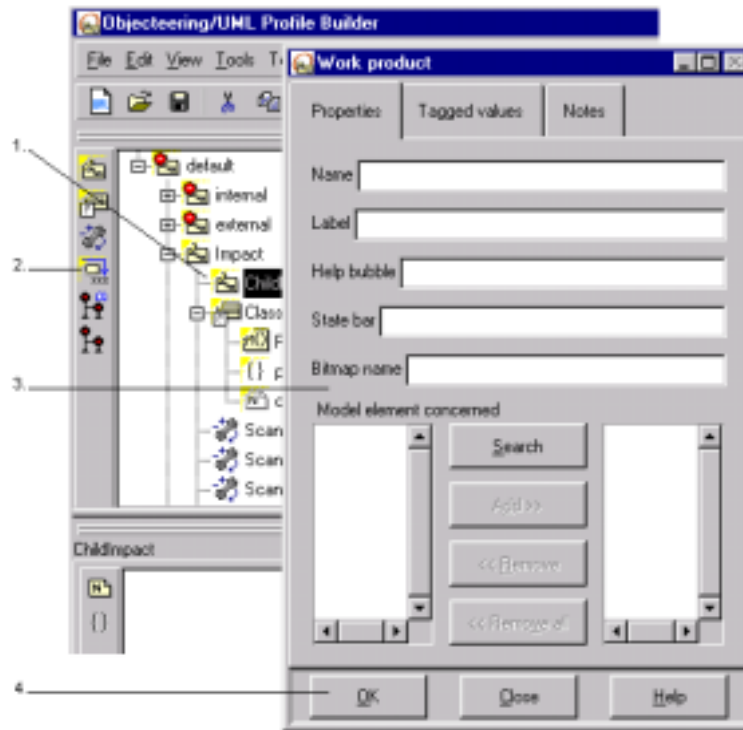



Figure 6-24. Creation procedure

Steps:

- 1 - Select a UML profile and create a child UML profile within it.
- 2 - Click on the  "Create a work product" button.
- 3 - Enter the fields (see table below).
- 4 - Confirm.

Text fields

The ... field	represents ...
Name	the name of the work product.
Label	the title of the work product dialog box
Help bubble	the description displayed in the work product's help bubble
Status bar	the description displayed in the work product's status bar
Bitmap name	the bitmap name displayed on the work product's button. The bitmap with the BMP format must be 19 pixels high and 21 pixels wide, stored in the \$OBJING_PATH/res/bmp directory and taken into account by the tool. The name entered must not contain the .bmp extension.
Model element concerned	model element on which the product button must appear

Associating a metaclass reference

At present, the "meta-product" can be accessed like any other Objecteering/UML metaclass. To associate J methods (necessary for creating commands on the work product) or J class attributes, a "metaclass reference" must be associated to it.

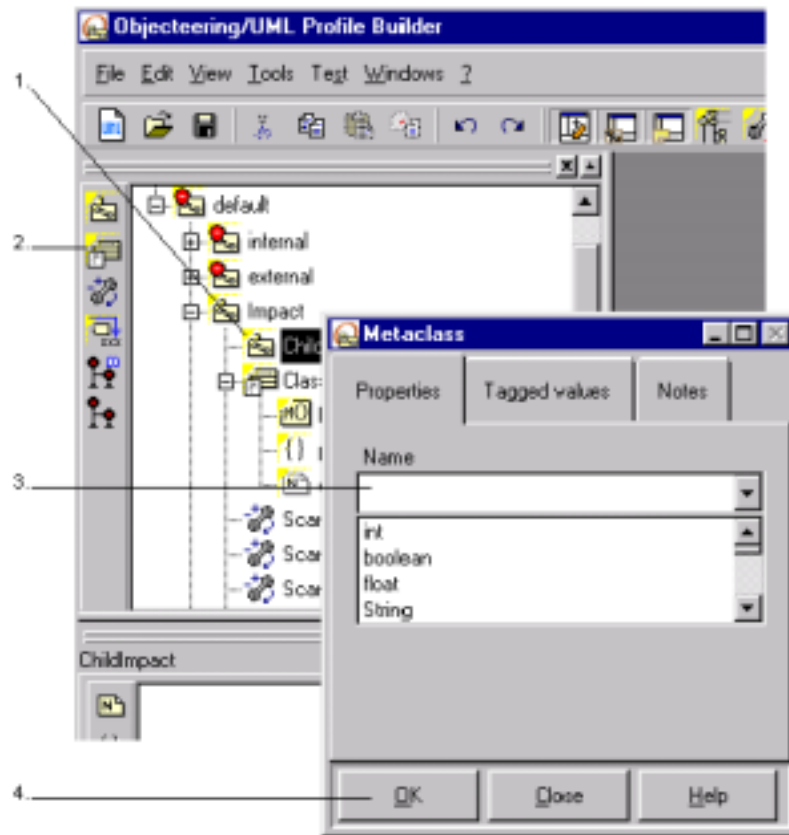



Figure 6-25. Referencing a metaclass for a meta-product

Steps:

- 1 - Select a child UML profile.
 - 2 - Click on the  "Create a *metaclass reference*" button.
 - 3 - Select or enter the meta-work product name.
 - 4 - Confirm.
-

Chapter 7: Defining modules

Overview of module definition

Module utility



A module is the functional packaging of a set of UML profiles. It is the unit which will be transferred between models, made available in *Objectteering/UML Modeler* UML modeling projects and utilized by the end user.

Module properties

In a module, the following must be defined:

- ◆ specialized modules
- ◆ the UML profiles used
- ◆ the UML profiles referenced
- ◆ the UML installation profile
- ◆ module commands
- ◆ the default values of module parameters

It is then possible for a module to be packaged, in order to allow it to be delivered and used on other sites.

Creating a module

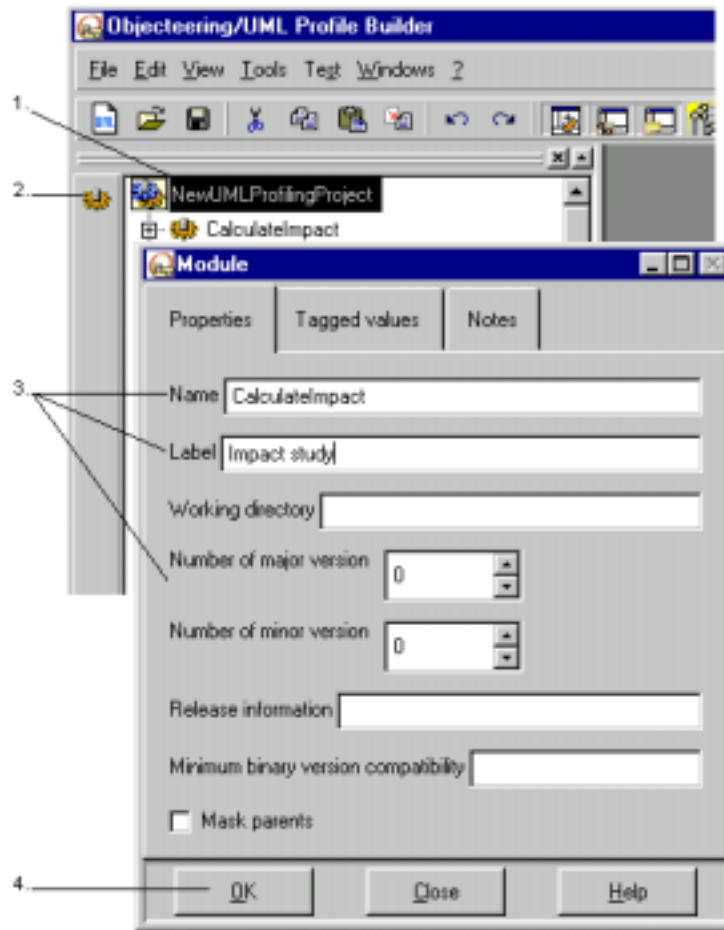



Figure 7-1. Creating a module in a UML profiling project

Steps:

- 1 - Select the UML profiling project.
- 2 - Click on the  "Create a *module*" button.
- 3 - Enter the necessary information in the data entry fields.
- 4 - Confirm.

Note: If the "*Working directory*" field is left blank, then the working directory is \$OBJING_PATH/modules/<ModuleName>/<Version>. Otherwise, the user may specify the working directory of his choice.

Referencing UML Profiles

Procedure

By referencing UML profiles, all their components may be accessed. Through these UML profiles, the module can access:

- ◆ module parameters defined in the UML profiles
- ◆ tagged values, note types and stereotypes
- ◆ public J methods defined in UML profile metaclasses, in order to link them to commands
- ◆ document templates and generation templates, defined in UML profiles
- ◆ generation work products, defined in UML profiles

The referencing of UML profiles allows you to use a profile as a UML installation profile.

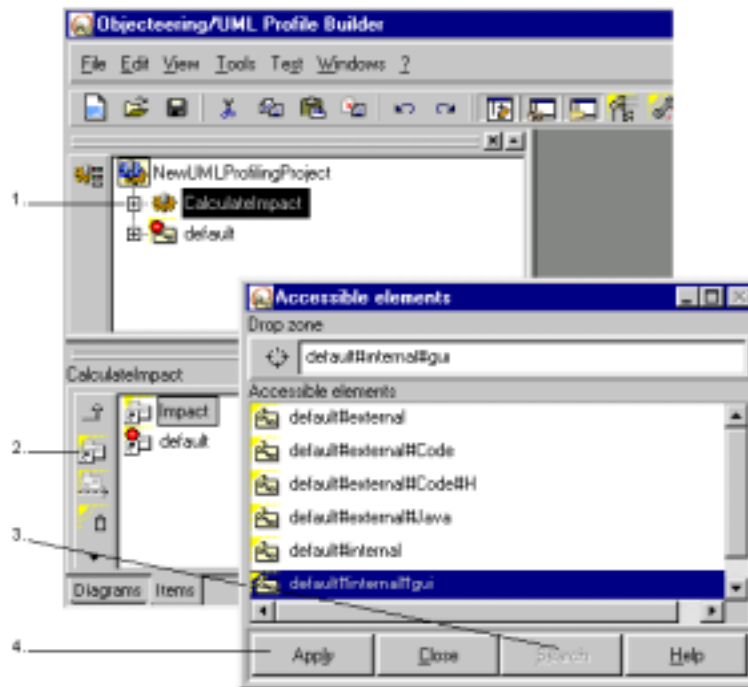



Figure 7-2. Referencing the "Impact" UML profile

Steps:

- 1 - Select the module in the meta-explorer.
- 2 - Click on the  "Reference a UML profile" icon in the "Items" tab of the properties editor.
- 3 - Select the UML profile(s) to be referenced by clicking on the "Search" button to display the list of accessible elements, and then by selecting the element in question. The element chosen will then appear in the "Drop zone" field.
- 4 - Confirm by clicking on the "Apply" button.

Using UML Profiles

Procedure

UML profiles should be used when the J code and stereotypes, tagged value types and note types contained therein are necessary to the module, but where work products, generation templates and parameters must be hidden. If you forget the essential "use" dependencies, then a J error (such as "method not found") will occur. If you use referencing instead of "use", then inappropriate work products or parameters will appear. UML profiles which are being used cannot be used as installation profiles.

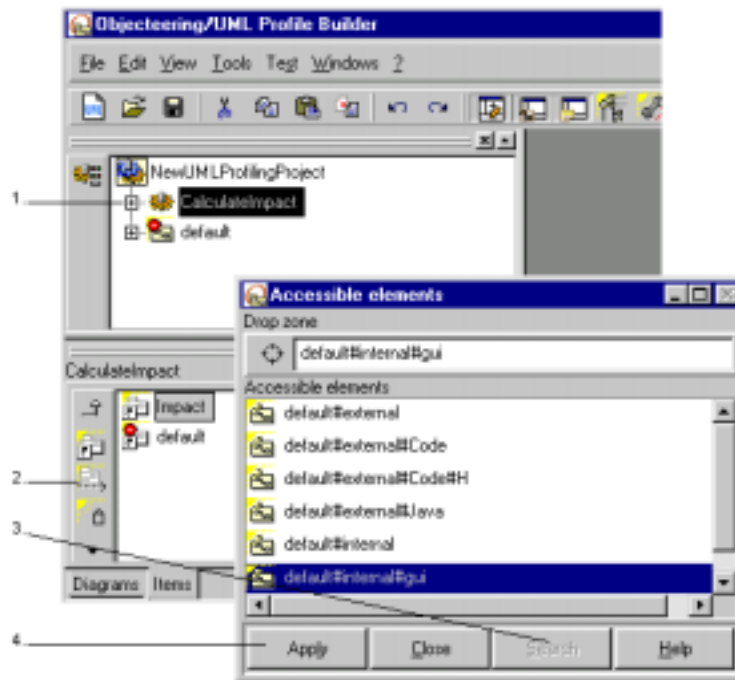



Figure 7-3. Using the "Impact" UML profile

Steps:

- 1 - Select the module in the meta-explorer.
 - 2 - Click on the  "Use a UML profile" button in the "Items" tab of the properties editor.
 - 3 - Select the UML profile(s) to be referenced by clicking on the "Search" button to display a list of accessible elements, and then by selecting the element in question, which will then appear in the "Drop zone" field.
 - 4 - Confirm by clicking on the "Apply" button.
-

Defining a UML installation profile

Procedure

Installation profiles are used to call additional actions associated with a specific operation. They are created automatically in the "*Object*" metaclass when the UML profile is selected as the installation profile, and the user may then further define them. The J methods which can be defined on installation profiles are as follows:

- ◆ the reception of a module
- ◆ the installation of a module
- ◆ the uninstallation of a module
- ◆ the selection of a module
- ◆ the unselection of a module

The actions called are as follows:

- ◆ For the reception of a module, `moduleInit`
- ◆ For the installation of a module, `moduleInstall`
- ◆ For the uninstallation of a module, `moduleUninstall`
- ◆ For the selection of a module, `moduleSelect`
- ◆ For the unselection of a module, `moduleUnselect`

(For further information, please refer to the "*Module management services*" section in chapter 5 of the *Objectteering/J Libraries* user guide).

Figure 7-4 illustrates the definition of a UML installation profile.

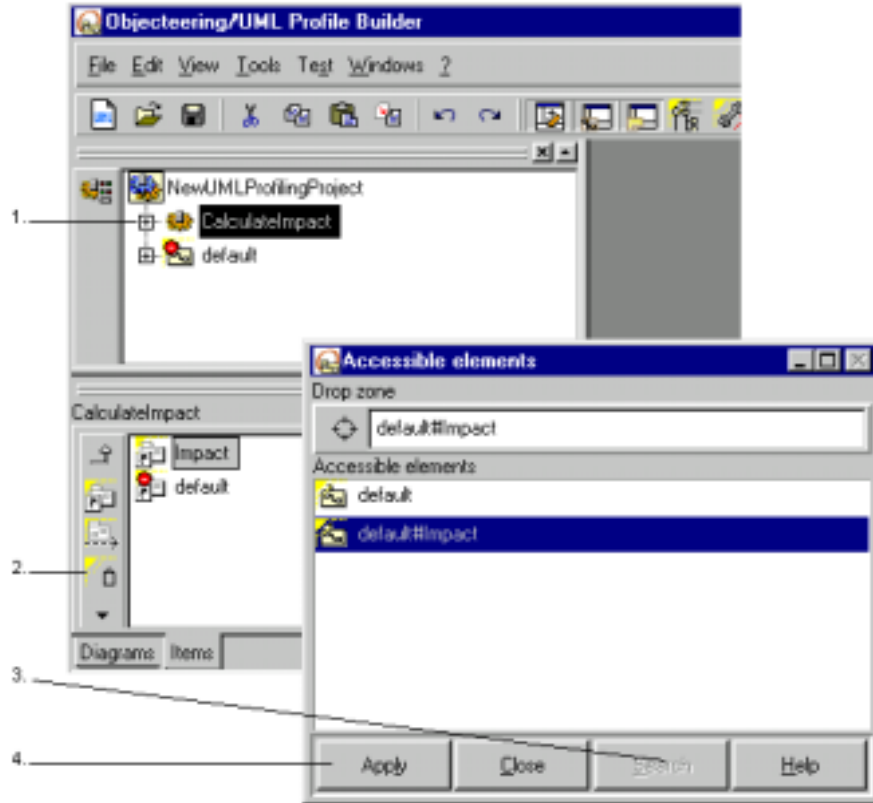



Figure 7-4. Defining a UML installation profile

Chapter 7: Defining modules

Steps:

- 1 - Select a module in the meta-explorer.
 - 2 - Click on a  "Installation UML profile" icon in the "Items" tab of the properties editor.
 - 3 - Click on the "Search" button to display a list of accessible elements, and then selecting the element in question, which will then appear in the "Drop zone" field.
 - 4 - Confirm by clicking on the "Apply" button.
-

Creating commands

Procedure

J rule entry points, which may be accessed by users in *Objecteering/UML Modeler*, are defined by commands. A command is included in the pop-up menu on the model elements concerned, and is associated to one of the metaclass J methods. The associated J method must be public and must have no parameters.

For example, in figure 7-5, the creation of the command introduces a pop-up menu item on a model's classes. This element will be called generation and trigger the "PrintImpact" J method of the "Impact" UML profile.

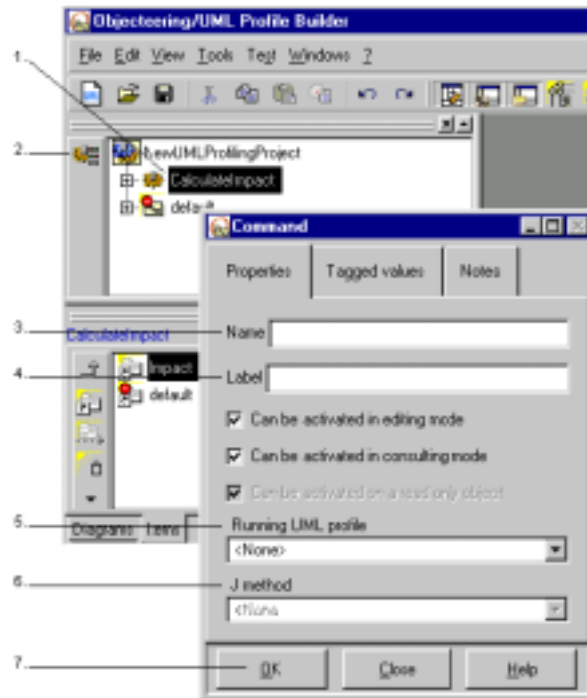



Figure 7-5. Creating a command

Chapter 7: Defining modules

Steps:

- 1 - Select the module.
- 2 - Click on the  "Create a command" button.
- 3 - Enter the name.
- 4 - Enter the identifier name.
- 5 - Select a UML profile.
- 6 - Select a J method.
- 7 - Confirm.

The "Command" dialog box

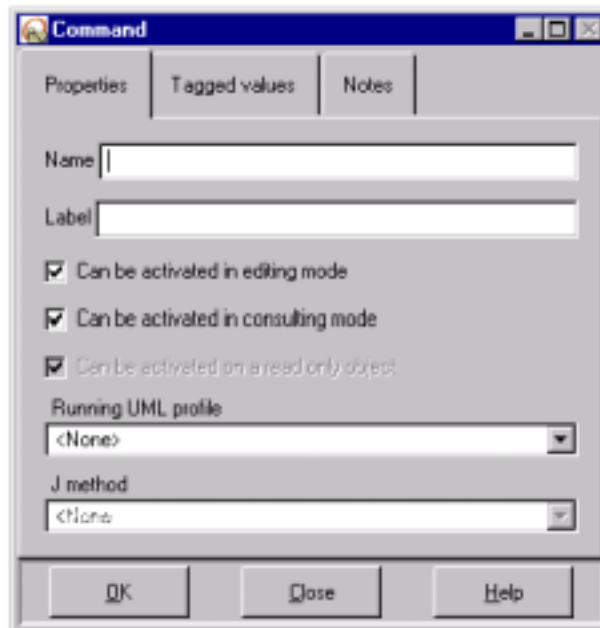


Figure 7-6. "Command" dialog box

The ... field or button	is used to ...
Name	enter the command name.
Label	enter the name of the menu item associated to this command
Can be activated in editing mode	specify whether or not the menu item associated to the command can be activated on an editor object in editing mode
Can be activated in consulting mode	specify whether or not the menu item associated to the command can be activated on an object in consulting mode
Can be activated on a read only object	specify whether or not the menu item associated with the command can be activated on an object in read only mode
Running UML Profile	select the context for activating the method
J method	select the J method to be invoked. It must be public and without parameters.

Note: When the "*Can be activated in consulting mode*" state is checked, the "*Can be activated on a read only object*" state is grayed out (since it serves no purpose).

Changing the default values of parameters

UML profiling project configuration

The user can modify the default values of module parameters. This is done in the "Modifying configuration" window launched from Objectteering/UML's main window (Figure 7-7) via the "Tools/Configure UML profiling project" menu.

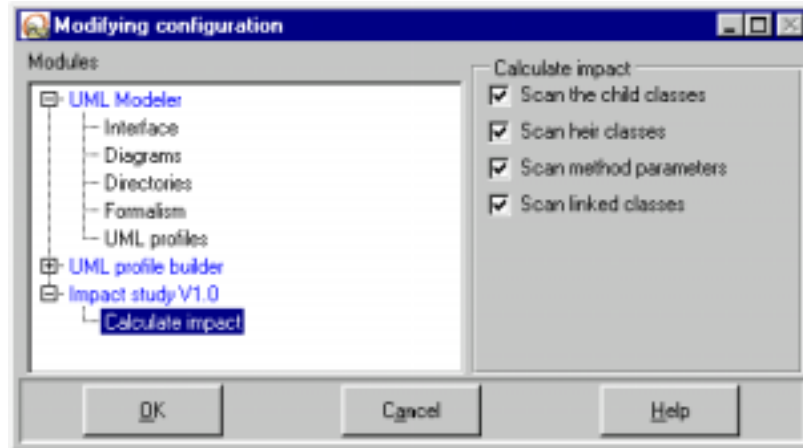


Figure 7-7. Editing the configuration

Specializing a module

Procedure

A module can specialize one or more UML profiling project modules. Cycles are forbidden. Specializing a module allows access to the parent module's parameters and UML profiles. Inherited parameter values are initialized with the value of the parent module and can then be modified.

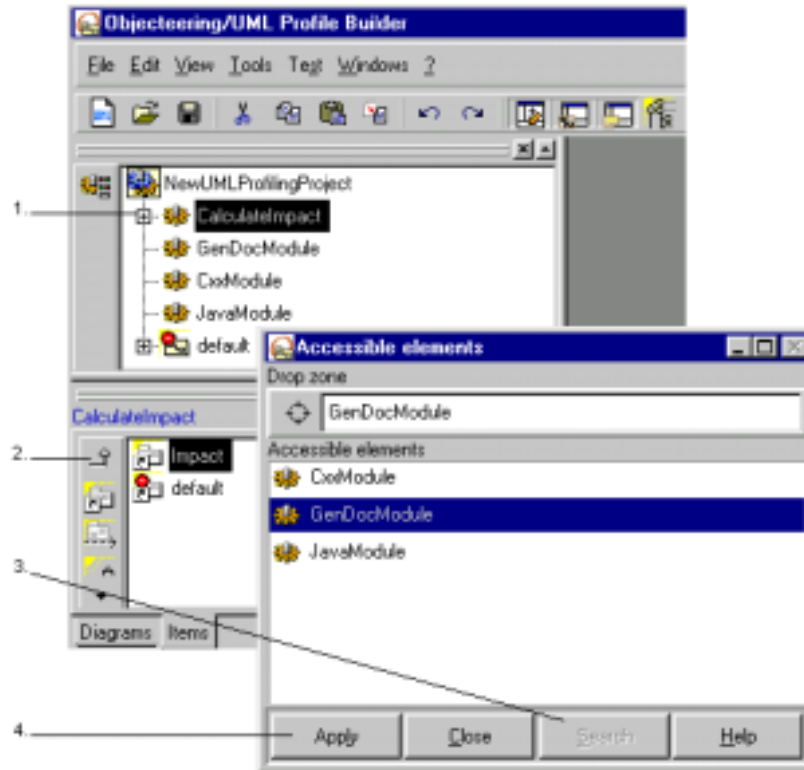



Figure 7-8. Specializing a module

Steps:

- 1 - Select the module.
- 2 - Click on the  "Specialize" button in the "Items" tab of the properties editor.
- 3 - Select the module by clicking on the "Search" button to display a list of accessible elements, and then by selecting the element in question, which will then appear in the "Drop zone" field.
- 4 - Confirm by clicking on the "Apply" button.

Note: If a module, M1, inherits from another module, M2, which owns a parameter, P, and M1 already references the UML profile where P is defined, or already inherits from a module which references this UML profile, M1 keeps the value P already has. This value can be different from M2's value.

Packaging a module

Procedure

Packaging a module allows you to obtain a module in the form of a file, in order to internalize it in another site. (For further information on installation, please refer to the "Overview of the Configuration menu" section of chapter 2 of the *Objectteering/Administrating Objectteering Sites* user guide.) The packaging mechanism is the correct way of packaging and delivering modules to other sites, and is the method used for standard Objectteering modules (*Objectteering/C++*, *Objectteering/Java*, *Objectteering/Documentation*, etc).

The packaged module (a .prof file) will be stored in the directory you indicate in the window shown in Figure 7-10. If no directory is specified, the module will be stored by default in the "\$OBJJING_PATH/modules/<ModuleName>/<Version>" directory. Module packaging may include resources, files, binaries, scripts, etc, found in the module work directory.

Note: \$OBJJING_PATH is a variable which represents the Objectteering/UML installation directory path.

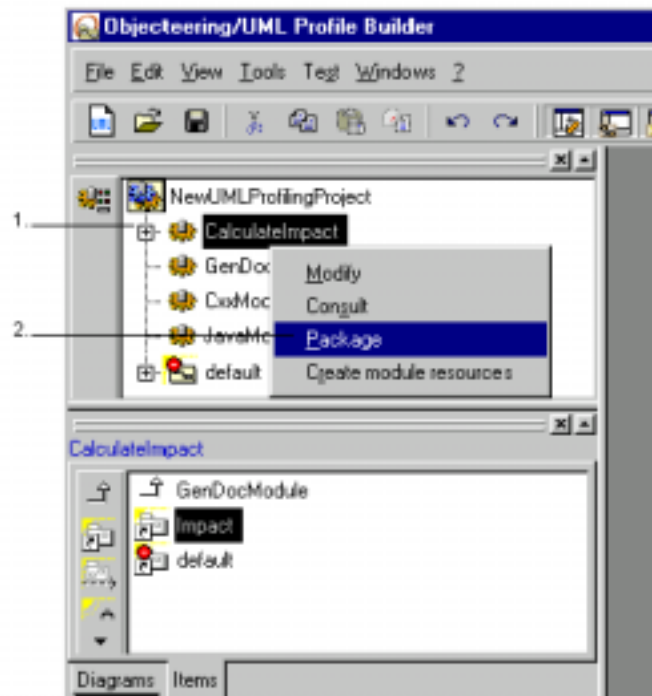


Figure 7-9. Packaging a module

Steps:

- 1 - Select the module using the right mouse-button.
- 2 - Choose the "Package" option from the context menu which appears.

Chapter 7: Defining modules

The following window (figure 7-10) then appears.

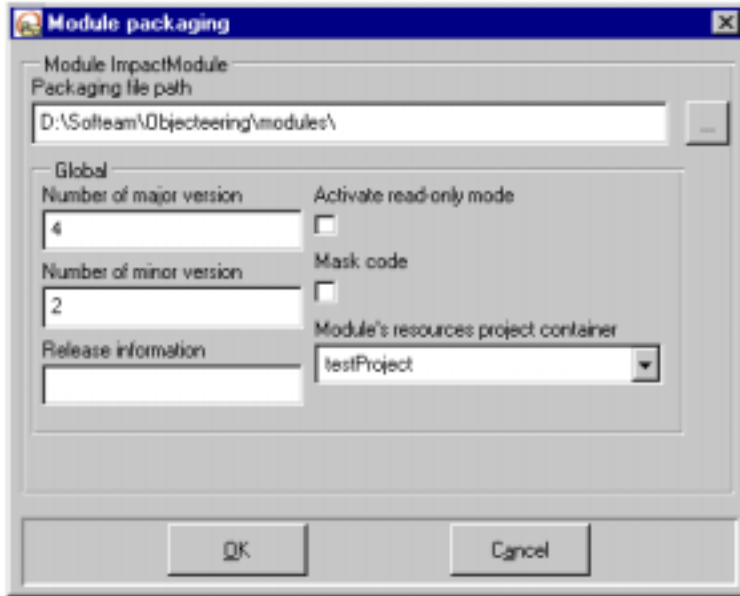


Figure 7-10. The "Module packaging" window

Key:

- ◆ "*Packaging file path*": this field is used to indicate where the module's packaged file is stored.
- ◆ "*Number of major version*": this field is used to indicate the first number in the version number (for example, if the complete version number is "4.2", then the major version number is 4).
- ◆ "*Number of minor version*": this field is used to indicate the second number in the version number (for example, if the complete version number is "4.2", then the minor version number is 2).
- ◆ "*Release information*": this field is used to indicate the release number for the module (for example, "a").
- ◆ "*Activate read only mode*": this field is used to indicate whether or not code should be in read only mode. It should be noted that even where this option is activated, the module itself is in read/write mode.
- ◆ "*Mask code*": this field is used to indicate whether or not methods will be visible to users.
- ◆ "*Module's resources project container*": this field is used to indicate in which UML modeling projects the module's resources are located (for example, the First Steps package, the types package, etc.).

Creating module resources

The "Create module resources" command is used to create a test project (if one does not already exist) and to internalize a types package template.

The types package template is used to parameterize a module without going via UML Profile Builder. This means that during generation phases, it is possible to parameterize generated J code with regard to a predefined type which is used in the modeled application.

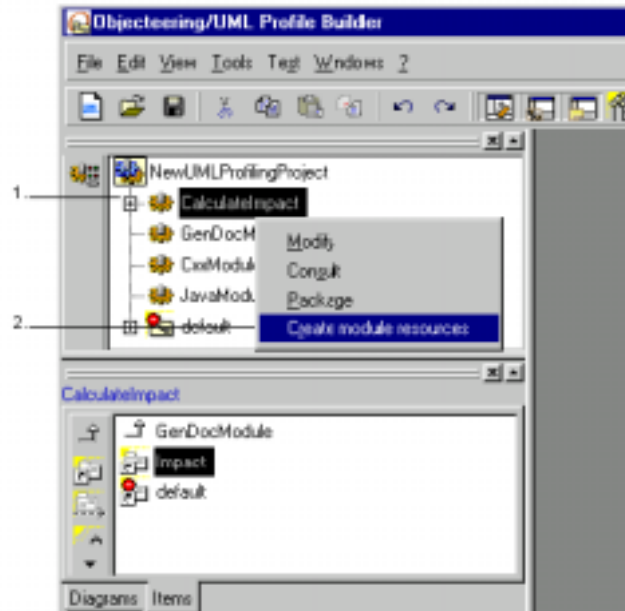


Figure 7-11. Running the "Create module resources" command

Steps:

- 1 - Select the module using the right-mouse button.
- 2 - Choose the "Create module resources" option from the context menu which appears.

Figure 7-12 shows the result of the "Create module resources" command in a test project and a meta-explorer.



Figure 7-12. The result of running the "Create module resources" command on a module

Key:

- 1 - A "TypesEditor" module has been added in the meta-explorer. This module is needed if the user wishes to work with the types package.
- 2 - Three packages ("TypesPackageContainer", "FirstStepsContainer" and "ModelsContainer") have been added in the test project explorer. The "TypesPackageContainer" package is created with "TypesPackage_Template", whilst the other two packages are created empty. These packages are optional, if your module does not need them.

Chapter 7: Defining modules

The "*TypesPackageContainer*" package should be used if your module uses types during generation processing.

The "*FirstStepsContainer*" package should be used if you wish to deliver a UML modeling project, which will be used as the module's First Steps project.

The "*ModelsContainer*" package should be used if you choose to deliver certain module parts which are needed to show your module's functionalities.

Note: For each first steps package or model part you wish to create, a container package is created ("*FirstStepsContainer*" package for module first steps and the "*ModelsContainer*" package for model parts). In this container package, you should then create a package for the first steps or the model parts concerned (for example "*FS1*").

Chapter 8: Test projects

Test project definition

The test project

The test project is a UML modeling project specifically associated to a UML profiling project. Any modification of the UML profiling project automatically affects the test project. This means that any modification made to J code, tagged values definition or the document template can be immediately tested on the test project. There is no module installation procedure to implement.

A test project is firstly created like a UML modeling project, and is exclusively dedicated to one UML profiling project.

Selecting a test project

Before being able to test your developments, a test project must be selected by the user (see Figure 8-1).

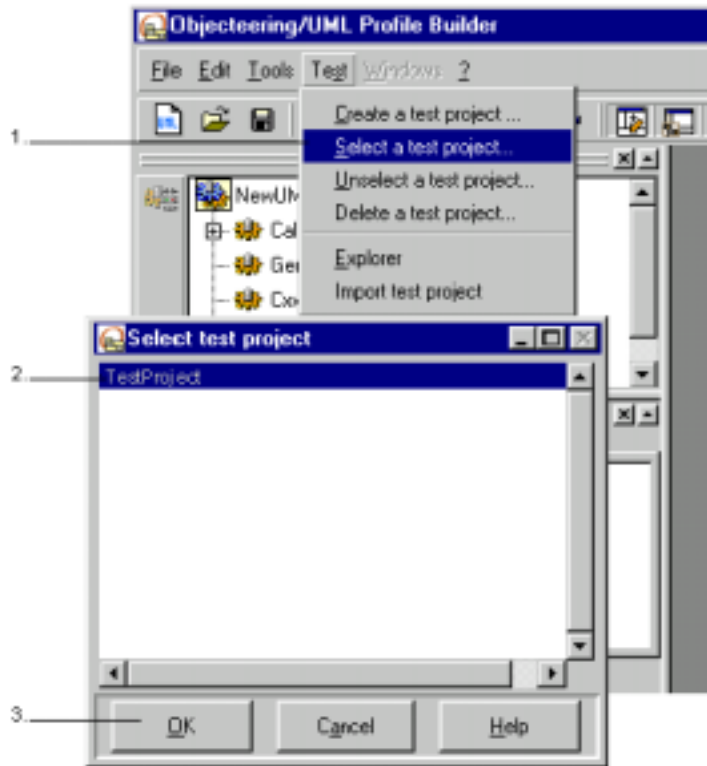


Figure 8-1. Selecting a test project

Steps:

- 1 - Choose the "*Select a test project...*" menu from the "*Test*" menu.
- 2 - Choose the test project you wish to use. Test projects are UML modeling projects in their own right.
- 3 - Confirm.

An explorer on the test project is then automatically started.

If no test projects are available for selection, you can create a test project by selecting the "*Create a test project...*" command from the "*Test*" menu. A window like the one shown in Figure 8-2 then appears.

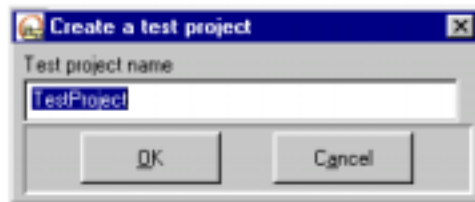


Figure 8-2. Creating a test project

Simply give a name to the test project you wish to create and then confirm by clicking on the "OK" button. An explorer is then automatically launched on the newly created test project.

Unselecting a test project

A test project can be unselected simply by running the "*Unselect a test project...*" command from the "*Test*" menu.

Note: A test project can only be changed once in the course of a modeling session. To change test project more than once, simply save your work and then use the "*File/New*" or "*File/Open*" commands to reload the UML profiling project in question.

Deleting a test project

A test project can be unselected simply by running the "*Delete a test project...*" command from the "*Test*" menu.

Importing a test project

It is possible with Objectteering/UML to import the contents of an existing UML modeling project into a UML Profile Builder test project. To do this simply carry out the steps shown in Figure 8-3.

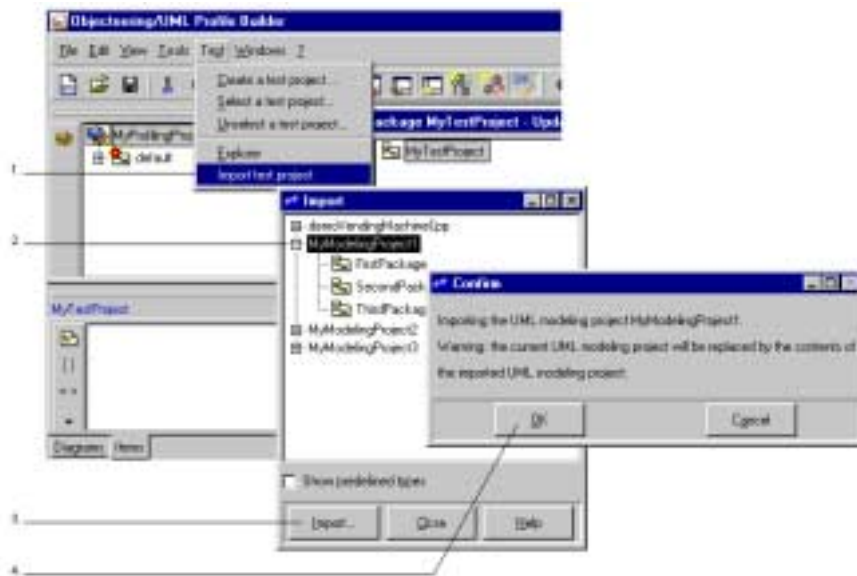


Figure 8-3. Importing the contents of the "MyModelingProject1" UML modeling project into the "MyTestProject" test project

Chapter 8: Test projects

Steps:

- 1 - Click on the "*Test/Import a test project...*" menu. The "*Import*" window then appears.
 - 2 - In the "*Import*" window, select the UML modeling project whose contents you wish to import. As you can see, you have access to all the UML modeling projects you have developed in the UML Modeler tool, and you can choose either to import the entire contents of one of these UML modeling projects or simply to import some of its components. In our example, we are going to import the entire contents of the "*MyModelingProject1*" UML modeling project.
 - 3 - Click on the "*Import*" button.
 - 4 - A confirmation dialog box then appears, informing you that the current contents of your test project will be overwritten with the contents of the UML modeling project you are importing. Confirm by clicking "*OK*".
-

Testing J methods

Procedure

When a command is created in a module, the elements of the test project whose metaclass is, or inherits from, the one to which the J method referenced by the command belongs to, have an item in their context menu which is used to launch this command. Therefore, if you wish to test the "*Generate*" method on the "*Class*" metaclass, create a command which refers to "*Generate*", and run it on any class of the test project.

Creating work products

When a module references a UML profile in which a work product is created, it is possible to create instances of this work product in the test project. A button appears in the explorer for all the elements which fulfill the following condition: the metaclass which these elements belong to is part of the list of metaclasses for which the work product has been defined, or which the work product specializes.

Index

- .prof file 7-20
- {nocode} tagged value 3-11
- {persistence} tagged value 5-10
- {persistent} tagged value 1-10, 6-6
- {primitive} tagged value 5-10
- {synchronized} tagged value 3-11, 3-27
- {virtual} tagged value 5-10
- Analysis phase 1-3
- Associating a metaclass reference 6-50
- Attributes 1-6, 5-7, 5-22
- Backward compatibility 4-7
- Boolean 6-32
- C++ 5-13
- C++ code generation 1-7
- C++ generation 5-3
- C++ generator 5-10
- Changing the default value of parameters 7-17
- Characteristics of Java code generation 3-3
- Checking rules 1-3
- Class 1-6
- Classes 3-3, 5-7, 5-10, 5-23
- Code generation templates 1-5
- Code generators 1-4
- Command 1-8
 - Creation 2-20
- Commands 1-5, 1-6, 1-7, 1-9, 2-18, 2-20, 3-37, 4-13, 5-3, 5-7, 5-22, 6-47, 7-3, 7-6, 7-13, 8-9
 - Access from the J language 5-9
 - Overview 5-7
- Configuring a module 2-22
 - Default values 3-39
- Configuring modules 5-6
- Configuring the UML profiling project 7-17
- Consistency checks 1-4
- Console 4-10, 4-12
- Constraint
 - Creation 6-16
- Constraints 1-3, 6-16
- Continuous entry creation mode 2-6, 2-12, 3-7, 3-10, 6-7, 6-10, 6-13, 6-22
- Creating a child UML profile 1-7
- Creating a command 1-8, 2-20
- Creating a constraint 6-16
- Creating a document item 6-42
- Creating a document template 1-7, 6-34, 6-40
- Creating a generation document template 1-7
- Creating a generation item 6-37
- Creating a generation template 6-34
- Creating a generator 3-3
- Creating a J class attribute 6-18
- Creating a J method 2-11, 6-21
- Creating a module 2-16, 3-36, 7-4
- Creating a parameter 1-7
- Creating a reference to a metaclass 1-7
- Creating a stereotype 6-12
- Creating a tagged value type 3-11
- Creating a test project 3-41, 8-5
- Creating a type of work product 1-7
- Creating a UML profiling project 2-4, 3-4
- Creating attributes 3-16
- Creating commands 3-37, 7-13
- Creating J attributes 3-12

- Creating module resources 7-24
- Creating new parameters 3-8
- Creating note types 3-9
- Creating parameters 3-15
- Creating work products 3-14, 8-9
- Creation buttons 4-16
- Default UML profiles 1-8
- Defining a module parameter 6-31
- Defining a UML installation profile 7-10
- Defining a work product 6-47
- Defining and visualizing tagged values 5-11
- Deleting a test project 8-6
- Design Patterns 1-4
- Design phase 1-3
- Directory 6-32
- Document and generation templates
 - Access from the J language 5-21
 - Overview 5-19
- Document items 1-9, 5-19, 5-21
- Document template 1-7, 7-6
- Document template project 1-9
- Document templates 1-4, 1-5, 1-6, 4-5, 4-13, 5-3, 5-19, 6-34, 8-3
- Documentation generation 5-3
- Drag and drop function 2-19
- Entering a J method 2-13
- Entering a type of note 6-9
- Entering a type of tagged value 6-6
- Enumeration 6-32
- Explorer 2-19, 6-47
- Extensibility mechanisms 5-3
- External text editors 5-25
- Externalizing modules 5-4
- File open 6-32
- File save 6-32
- Functions 1-5
- Functions of the UML Profile Builder tool 1-3
- Generation document template 1-7
- Generation items 1-9, 5-19, 5-21, 6-37
- Generation rules* 1-3
- Generation template project 1-9
- Generation templates 1-4, 4-13, 5-19, 6-34, 7-6, 7-8
- Generation work product 5-21
- Generation work products 1-5, 1-6, 3-4, 3-14, 3-16, 3-17, 3-24
- getCurrentModuleParameterValue 5-6
- Implementing J methods 3-23
- Installation 5-4
- J attributes 1-9, 4-13, 6-3
- J language 1-4, 1-9, 5-6, 5-9, 5-12, 5-15, 5-25
- J method
 - Creation 2-11
 - Entry procedure 2-13
 - moduleInstall 3-22
- J methods 1-5, 1-6, 1-7, 1-8, 1-9, 1-10, 2-9, 2-11, 2-13, 3-9, 3-18, 4-13, 5-7, 5-9, 5-21, 5-26, 6-3, 6-18, 6-21, 6-25, 6-28, 6-47, 7-6, 7-10, 7-13, 8-9
 - Creating J methods 3-18
 - Creating J methods for managing the module 3-22
 - Creating the visualization J methods 3-22
- edit 3-22
- generate 3-18
- getCode 3-18
- getLineComment 3-22
- getType 3-18

- initProduct 3-20
- isPresent 3-20
- J methods for managing work products 3-20
- moduleInit 7-10
- moduleInstall 7-10
- moduleSelect 7-10
- moduleUninstall 3-22, 7-10
- moduleUnselect 7-10
- mustPropagate 3-20
- Protected 6-28
- Public 6-28
- Testing J methods 8-9
- update 3-20
- visualize 3-22
- J rules 1-7
- J services 1-5
 - getCurrentModuleParameterValue 5-6, 6-31
- Java 1-4
- Java generation template 5-20
- Java work product 5-23
- Java-like syntax 1-4
- Licenses 5-5
- Loading modules
 - Access from the J language 5-6
 - Configuring modules 5-6
- Major version 7-23
- Makefile generation 1-7
- Markers 6-27
- Mask code 7-23
- Menu bar 4-12
- Metaclass 1-10
- Metaclass definition 6-3
- Metaclasses 1-6, 1-7, 1-9, 2-9, 2-11, 5-9, 5-12, 5-16, 6-3, 6-6, 6-12, 6-16, 6-21, 6-28, 6-36, 6-47
- Referencing procedure 2-9
- Meta-explorer 4-12, 4-13, 4-16, 7-25
 - Description 4-15
- Metamodel 1-6, 1-9
- Metamodel classes 1-8
- Minor version 7-23
- Model transformation 5-25
- Model transformation services 5-25
- Model-driven development 1-3
- ModelElement metaclass 5-12, 5-15, 5-18
- Module 1-6, 1-8, 7-8
 - Commands 7-3
 - Configuration 2-22
 - Creation 2-16
 - Definition 7-3
 - Parameters 7-3
- Module configuration
 - Module configuration window 2-23
- Module configuration window 2-22
- Module parameters 1-9
 - Creation 2-7
- Module resources 7-23
- Module transformation 1-9
- moduleInit 7-10
- moduleInstall 7-10
- Modules 1-5, 1-6, 1-7, 1-9, 2-3, 2-16, 2-22, 2-23, 3-16, 4-13, 4-15, 5-3, 5-4, 5-7, 5-10, 5-13, 5-22, 5-25, 6-44, 8-9
 - Creating a module 3-36
 - Overview 5-3
- moduleSelect 7-10
- moduleUninstall 7-10
- moduleUnselect 7-10
- MpGenProduct metaclass 5-24
- Note type 1-9

- Note types 2-9, 3-4, 4-13, 5-13, 6-3, 7-6, 7-8
 - Creating a note type 3-9
- Notes 1-5, 1-6, 1-7, 2-9, 3-3, 4-13, 5-3, 5-13, 7-6, 7-8
 - Access from the J language 5-15
 - Overview 5-13
- Objecteering/Administrating
 - Objecteering Sites 7-20
- Objecteering/C++ 1-5, 1-6, 5-3, 6-34, 7-20
- Objecteering/Document Template Editor 6-34
- Objecteering/Documentation 1-5, 5-3, 7-20
- Objecteering/Introduction 1-5
- Objecteering/J Libraries 1-5, 7-10
- Objecteering/Java 7-20
- Objecteering/Metamodel 1-5, 1-6, 5-15, 6-3, 6-47
- Objecteering/Metamodel User Guide 5-12
- Objecteering/Model Dialog Boxes 5-22
- Objecteering/The J Language 1-5, 1-6, 1-9, 5-24, 6-21
- Objecteering/UML installation procedure 5-4
- Objecteering/UML metamodel 5-15, 5-18, 6-18
- Objecteering/UML Modeler 1-4, 1-5, 1-6, 1-7, 1-9, 2-3, 2-5, 3-5, 4-5, 4-6, 6-47, 7-3, 7-13
- Objecteering/UML Profile Builder 1-10, 4-6
- Opening a meta-explorer 4-14
- Opening an existing UML profiling project 4-6
- Operation 1-6
- Operations on a module
 - Creating a command 6-44
 - Inheriting from a module 6-44
 - Packaging the module 6-44
 - Referencing a UML profile 6-44
 - Selecting the installation UML profile 6-44
 - Using a UML profile 6-44
- Oracle generation 5-3
- Oracle generator 5-10
- Other customizable services
 - Overview 5-25
- Package 5-23
- Packages 3-3
- Packaging a module 7-20
- Packaging file path 7-23
- Parameter types
 - Boolean 6-32
 - Directory 6-32
 - Enumeration 6-32
 - File open 6-32
 - File save 6-32
 - Password 6-32
 - String 6-32
- Parameters 1-5, 1-6, 1-7, 2-7, 2-22, 2-23, 3-3, 3-8, 3-16, 3-21, 3-27, 3-39, 4-13, 5-6, 5-9, 6-31, 7-3, 7-6, 7-8, 7-13, 7-17, 7-18
 - Creating new parameters 3-8
- Password 6-32
- Process-driven development* 1-3
- Properties editor 4-12, 4-13, 5-11, 5-23, 7-7
 - Annotating structural elements 4-18
 - Creating J methods 4-18
- Items tab 2-26, 4-19

- Redefining J methods 4-18
- Referencing UML profiles 4-18
- Selecting UML installation profiles 4-18
- Specializing modules 4-18
- Terminal element creation icons 4-20
- Using UML profiles 4-18
- RDB generation 1-7
- Read only mode 7-23
- Read/write mode 7-23
- Receiving a UML profiling project 4-7
- Receiving and renaming a UML profiling project 4-7
- Receiving and upgrading a UML profiling project 4-7
- Receiving UML profiling projects 4-6
- Receiving, renaming and upgrading a UML profiling project 4-7
- Redefining a J method 6-28
- Referencing a metaclass 2-9, 6-3, 6-4
- Referencing a UML profile for the module 2-18
- Referencing UML profiles 1-8, 7-6, 8-9
- Release information 7-23
- Saving your model context 4-6
- Selecting a test project 2-24, 8-4
- Selecting modules 5-5
- Specialized modules 7-3
- Specializing a module 7-18
- Specializing the UML profiling project's module 1-8
- Status bar 4-12
- Stereotype
 - Creation 6-12
- Stereotype visibility 6-15
- Stereotypes 1-3, 1-5, 1-7, 1-10, 4-13, 5-16, 6-3, 6-12, 7-6, 7-8
 - Access from the J language 5-18
 - Overview 5-16
- String 6-32
- Structural element creation icons in the meta-explorer 4-16
- Structure
 - Modules 1-6
 - UML profiles 1-6
- Tagged value types 1-10, 2-9, 3-4, 3-11, 4-13, 6-3, 6-6, 7-6, 7-8
 - Creation 3-11
- Tagged values 1-3, 1-5, 1-6, 1-7, 1-10, 2-9, 3-3, 3-27, 4-13, 5-3, 5-10, 6-6, 7-6, 7-8, 8-3
 - Access from the J language 5-12
 - Overview 5-10
- TaggedValue metaclass 5-12
- TagType metaclass 5-12
- Terminal element creation icons 4-20
- Test project 1-10, 2-26, 7-25
- Test projects 3-40, 8-3, 8-9
 - Creating a test project 8-5
 - Deleting a test project 8-6
 - Importing a test project 8-7
 - Selecting a test project 8-4
 - Unselecting a test project 8-6
- Testing a UML profiling project 2-24
- Testing code edition 3-43
- Testing code generation 3-42
- Testing code visualization 3-43
- Testing J methods 8-9
 - Creating work products 8-9
- Testing the UML profiling project 3-40

Textually editing a J method 6-25

The "Attribute" dialog box 6-20

The "Attribute::generate ()" method 3-28

The "Attribute::getType ()" method 3-29

The "Class::generate ()" method 3-25

The "Command" dialog box 7-15

The "Document template" dialog box 6-40

The "Generation item" dialog box 6-38

The "J Method" dialog box 6-23

The "JavaProduct::edit" method 3-34

The "JavaProduct::generate ()" method 3-24

The "JavaProduct::getIdLineComment" method 3-34

The "JavaProduct::initProduct ()" method 3-30

The "JavaProduct::isPresent" method 3-32

The "JavaProduct::mustPropagate ()" method 3-32

The "JavaProduct::update ()" method 3-31

The "JavaProduct::visualize" method 3-33

The "Module" dialog box 6-44

The "Note type" dialog box 6-11

The "Object::moduleInstall" method 3-34

The "Object::moduleUninstall" method 3-35

The "Operation::generate ()" method 3-27

The "Operation::getCode ()" method 3-28

The "Package::generate ()" method 3-24

The "Parameter" dialog box 6-32

The "Stereotype" dialog box 6-14

The "Tagged value type" dialog box 6-8

The "Type of constraint" dialog box 6-17

Tool bar 4-12

Transformation rules 1-9

Types package template 7-24

UML 1.3 1-6

UML installation profile 7-3, 7-6, 7-10

UML installation profiles 7-8

UML modeling project 1-7, 2-3, 5-3, 5-7, 6-31, 8-3

UML modeling projects 1-4, 4-3, 4-6, 5-4

UML profile 1-10

UML Profile Builder workspaces 1-4

UML profile generalization 1-7

UML profiles 1-3, 1-6, 1-7, 1-8, 2-3, 2-6, 2-9, 2-19, 2-23, 3-4, 3-6, 3-8, 3-9, 3-11, 3-12, 3-14, 3-18, 3-23, 3-36, 4-13, 4-15, 5-16, 6-3, 6-6, 6-21, 6-25, 6-28, 6-32, 6-36, 6-47, 7-3, 7-6, 7-8, 7-18

Creating a child UML profile 1-7, 2-6

Creating a UML profile 3-6

Definition 1-7

Referencing a UML profile for the module 2-18

UML profiling project 3-4, 4-11, 4-13, 4-16

UML profiling project 1-7, 2-3, 2-16, 2-24, 4-15, 4-19, 4-20, 8-3

Creation 2-3

Test 2-24
UML profiling projects 1-8, 1-10, 2-6,
4-3
 Definition 1-7
UML rules 1-3
Undo/redo operations 5-25
Unselecting a test project 8-6
Upgrading a UML profiling project 4-
9
Upgrading UML profiling projects 4-6
Using modules 5-4

Selecting modules in a UML
modeling project 5-5
Using UML profiles 1-8, 7-8
Validation rules 1-3
Work context 4-6
Work product 1-7, 1-10
Work products 3-14, 3-30, 4-13, 5-3,
5-7, 7-8
 Access from the J language 5-24
 Overview 5-22
Workspaces 1-4