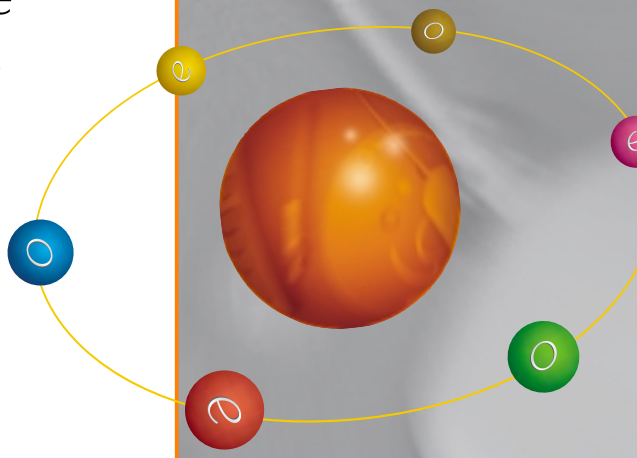
Objecteering/UML

Objecteering/Documentation User Guide

Objecteering/Document Template Editor User Guide

Version 5.2.2



Objecteering Software

www.objecteering.com

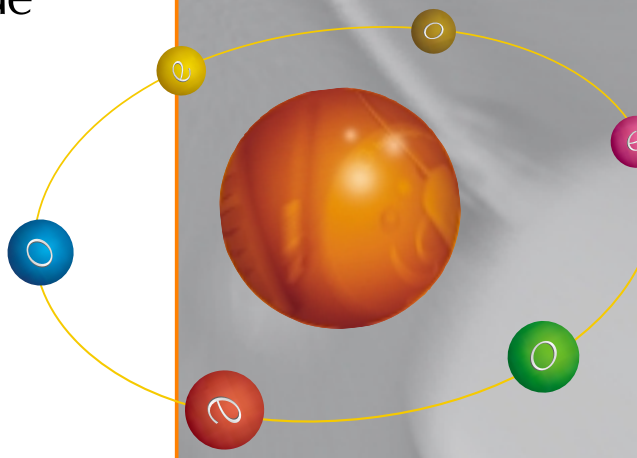
Taking object development one step further

Objecteering/UML version 5.2.2 - CODOBJ 001/001

Objecteering/UML

Objecteering/Documentation User Guide

Version 5.2.2



Objecteering

Software

www.objecteering.com

Taking object development one step further

Contents

Chapter 1: Overview

Introduction

Overview

Welcome to the Objecteering/Documentation user guide!

The *Objecteering/Documentation* module is used to automatically generate all documentation resulting from a model developed in Objecteering/UML. Documents are accessible from the most widely used text editors.

Predefined document templates

Objecteering/UML version 5.2.2 is delivered with two document templates:

- a *specification document template* which is used to obtain an analysis document
- a *design document template* which is used to obtain a design document

Generated documentation formats

Objecteering/UML offers the following default generation formats:

- *Postscript*: this format can only be generated in UNIX. It offers "*clean*" documents, even without a word processing tool. UNIX has many Postscript visualization tools.
- *RTF*: the most widely used word processor in the world, in the Windows environment.
- *HTML*: a "*universal*" format, UNIX and Windows, providing very useful hypertext features used to obtain distributed and interactive documents. For printing on paper, the *Postscript* and *Word* formats are more appropriate.
- *Ascii*: this format can be generated and visualized in UNIX and Windows.

Parameterizing documentation

Predefined document templates allow you to define parameterization options.

The document template editor can be used to construct all sorts of other documents.

Parameterizing documentation generation

Input information

The generation of documentation is based on:

- modeling information
- annotation through tagged values
- notes (text zones), which can be freely entered
- a document template which customizes the desired documentation

Generated documentation can, therefore, be of any kind (specification document, design document, quality control documents, etc.), depending on the parameters set by the user.

Producing formatted documents

The formatting of documentation is the last step in generation (see Figure 1-1). A specific processor called "*tpf*" generates formats adapted to different text formats (*Postscript*, *ascii*, *Rtf*, etc.) selected by the user. You can therefore apply several different processors to the same document, depending on the desired format.

Formatting is transparent to the user. However, it is interesting to note that documentation generation generates a document in a common format, before converting it to the desired format (*HTML*, *.rtf*, etc).

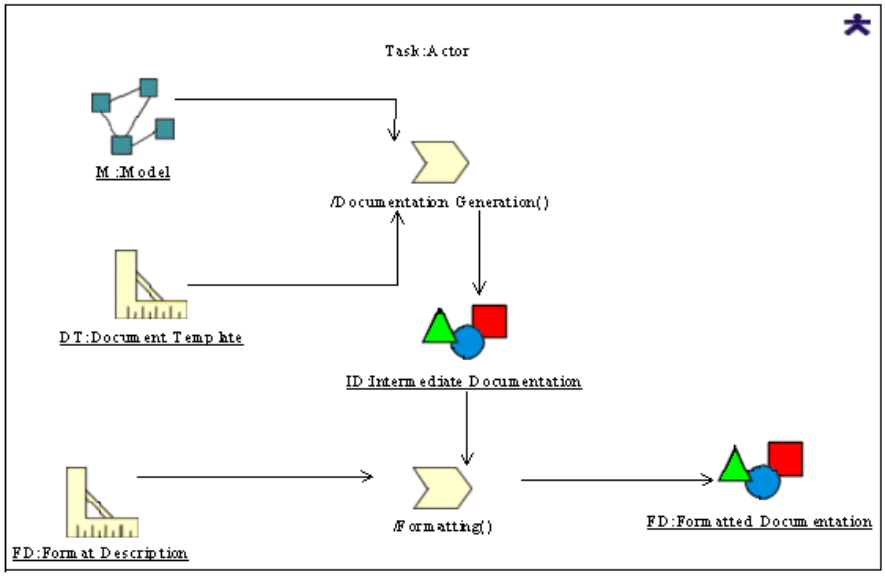


Figure 1-1. Documentation generation process

Entering free text

Complementary free text is entered through Objecteering/UML notes. These can be defined through model dialog boxes (as shown in Figure 1-2) or in the "*Items*" tab of the properties editor.

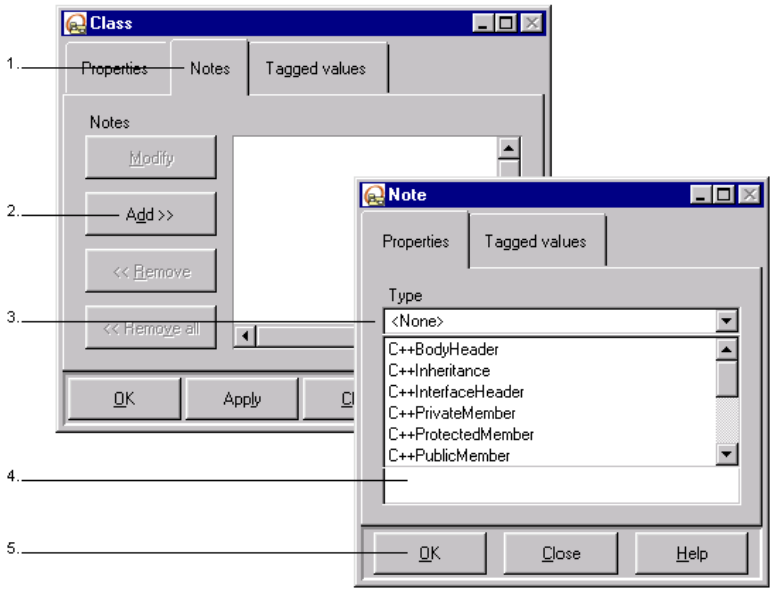


Figure 1-2. Entering the description of a class

Steps:

1 - Select the "*Notes*" tab.

2 - Click on "*Add*".

3 - Select the "*description*" note type.

4 - Enter the contents.

5 - Confirm.

Glossary

Ascii: One of the documentation formats provided, this format can be generated and visualized in UNIX and Windows.

Document package: This references all the elements which should be in the documentation.

Documentation root: Documentation generation always starts from an element in the model, to generate all the elements structured by this element. Most of the time, this is a package, but it can also be a class, an actor, a use case, or another type of element.

Document template: A document template describes how to obtain a document from a model. It entirely conditions the nature of the produced document.

Document work product: This allows you to add specific information related to the documentation in question (title, author), and to generate the information in the desired format.

Generation directory: This is the directory where documentation is generated.

HTML: One of the documentation formats provided, this format can be generated and visualized in UNIX and Windows. HTML documentation can be edited by all web explorers.

Note type: All model elements (class, attribute, association, diagram, etc.) have at least one "*Note*" type to describe them, with the two most common note types being "*summary*" and "*description*". These notes give additional information on the element in question.

Postscript: One of the documentation formats provided, this format can only be generated and visualized in UNIX.

.rtf: One of the documentation formats provided, this format can be generated in UNIX and Windows, but can only be visualized in Windows.

Tagged value: Tagged values are used to explicitly include or exclude a particular document item during generation, or to assign a specific nature to an element.

Chapter 2: First Steps

First steps - Overview

Generating documentation from a model

Objecteering/UML's general first steps describe how to create a model, add descriptions and generate documentation. We recommend you go through these first steps, in order to become familiar with the general services provided by Objecteering/UML, before continuing with the first steps specific to the *Objecteering/Documentation* module.

In order to obtain documentation from a model, you need simply understand the document work product notion.

The essential notions

In the document configuration window, the user can parameterize his favorite editors, as well as the generation directory. He must have a structured model, in which a package, called the documentation package, references all the elements which should be in the documentation. This package is used to group together classes, other packages or themes, as well as use cases. The order in which packages and other elements are presented will be used to organize the documentation into chapters.

For each element, different types of notes which describe different aspects of the element are proposed. Documentation generation can then be run, in order to generate the desired documentation.

Creating notes and tagged values

Creating notes and tagged values in the "Items" tab of the properties editor

Create a UML modeling project named "*UMLTraining*". Inside this UML modeling project, create the "*TrainingSystem*" package and then create the {*analysis*} tagged value on the "*TrainingSystem*" package, as shown in Figure 2-1.

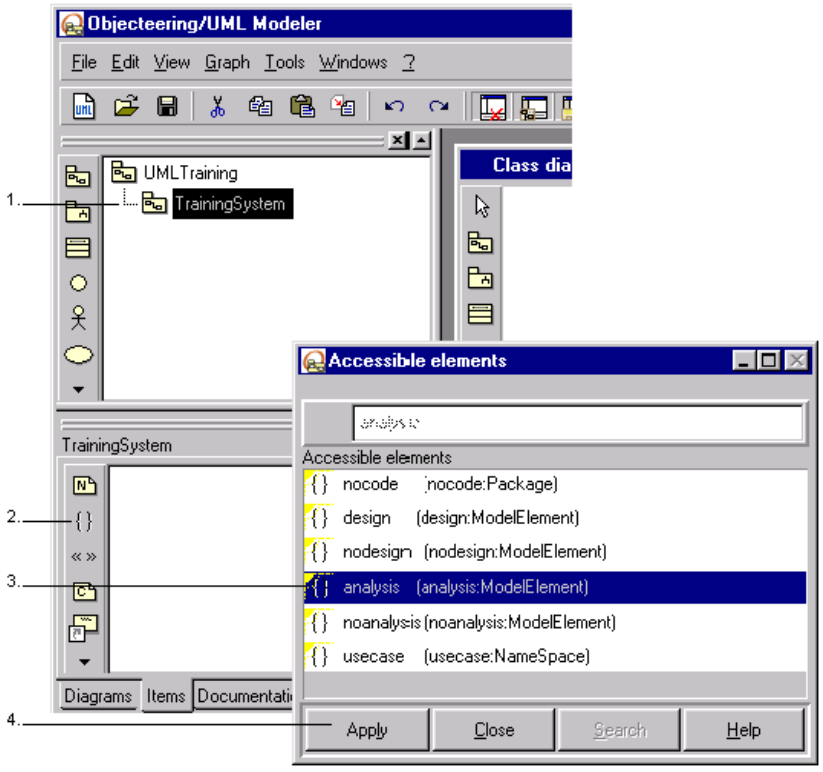


Figure 2-1. "*TrainingSystem*" is the document package

Steps:

1 - Select the "*TrainingSystem*" package in the explorer.

2 - Select the "*Items*" tab in the properties editor, and click on the  "*Associate a tagged value*" icon. The "*Accessible elements*" window then appears.

3 - Select the {*analysis*} tagged value.

4 - Click on "*Apply*" to confirm.

Continue by creating the "*overview*", "*purpose*", "*reference*" and "*dictionary*" notes in the same way (by clicking on the  "*Create a note*" icon in the "*Items*" tab).

Creating notes and tagged values in the "Documentation" tab of the properties editor

Certain notes and tagged values can be created through the "*Documentation*" tab of the properties editor. We are now going to continue by carrying out the steps shown in Figure 2-2.

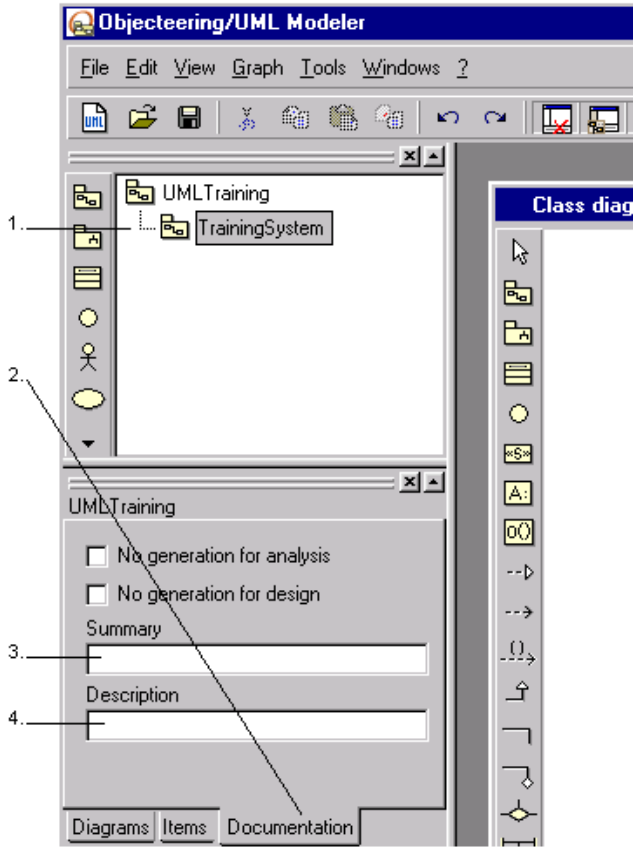


Figure 2-2. Creating notes in the "*Documentation*" tab of the properties editor

Steps:

1 - Select the "*TrainingSystem*" package in the explorer.

2 - Select the "*Documentation*" tab in the properties editor.

3 - Enter the contents of the summary note you wish to create in the "*Summary*" field.

4 - Enter the contents of the description note you wish to create in the "*Description*" field.

You can check the results of these entries by clicking on the "*Items*" tab of the properties editor. As you will see, a description note and a summary note are now present on the "*TrainingSystem*" package.

Note: The "*No generation for analysis*" and "*No generation for design*" fields are used to set the {*nodesign*} and {*noanalysis*} tagged values on the selected element.

Documentation notes

All model elements (class, attribute, association, diagram, etc.) have at least one note type to describe them, with the two most common types of note being "*summary*" and "*description*".

The ... note type	is used to ...
overview	give a general presentation of the document and the information system it describes.
purpose	explain the document's objective, what is represented and what is computerized.
reference	give the list of document references used by the present document.
dictionary	give the glossary of terms used in the model.
summary	give a short description (in one line) of the current package.
description	describe in detail the current package.

Creating document links

Creating a document link

It can be useful to create links within your document to other external documents, or to insert these documents into your document as images or text. To do this, we first have to create a document link, as shown in Figure 2-3.

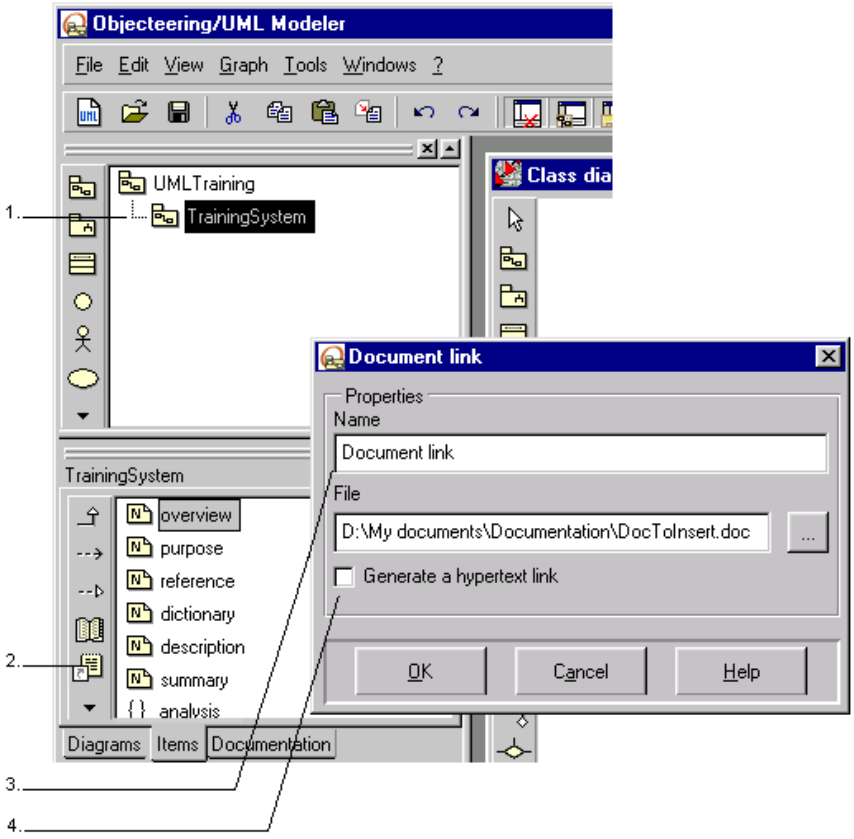


Figure 2-3. Creating a document link

Steps:

1 - Select the "*TrainingSystem*" package in the explorer.

2 - In the "*Items*" tab of the properties editor, click on the  "*Create a document link*" icon. The "*Document link*" window then appears.

3 - Enter a name for your document link and then enter the path of the document which you wish to link to your document. A file browser is opened by clicking on the  icon. Confirm by clicking on "*OK*".

4 - If you wish to generate a hypertext link to the specified document (instead of physically inserting the linked document into your document), check the "*Generate a hypertext link*" tickbox.

Note: The "*File*" field can contain environment variables (previously defined by the user), which must respect the following syntax: $(VariableName). For example, $(VariableName)\Documentation\DocToInsert.doc.

Running commands on your document link

After creating your document link, four commands are available on it via a context menu, as shown in Figure 2-4.

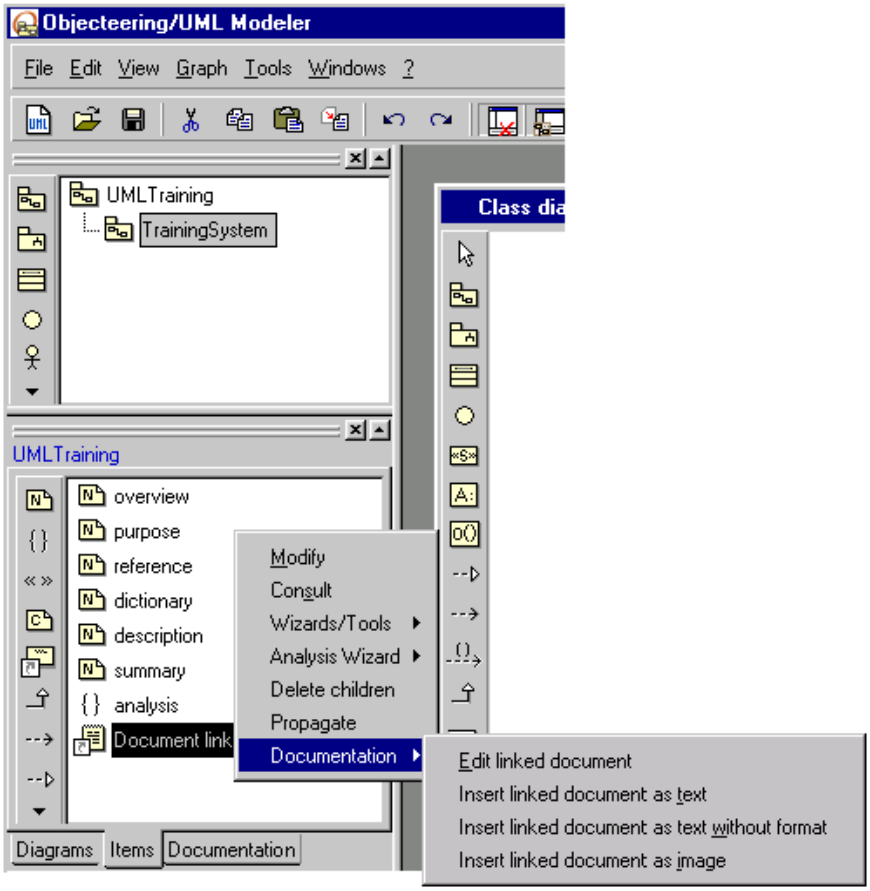


Figure 2-4. The context menu available on a document link

Key:

- "*Edit linked document*": This command edits the linked document using the appropriate tool.
- "*Insert linked document as text*": This command inserts the document you specified into your document in text format.
- "*Insert linked document as text without format*": This command inserts the document you specified into your document and retains its layout.
- "*Insert linked document as image*": This command inserts the document your specified into your document as an image.

Launching documentation generation

The document work product

The document work product is used to add specific information related to the documentation (title, author) and to generate the information in the desired format. We will proceed as in Figure 2-5, and try the four suggested format types.

Generation steps

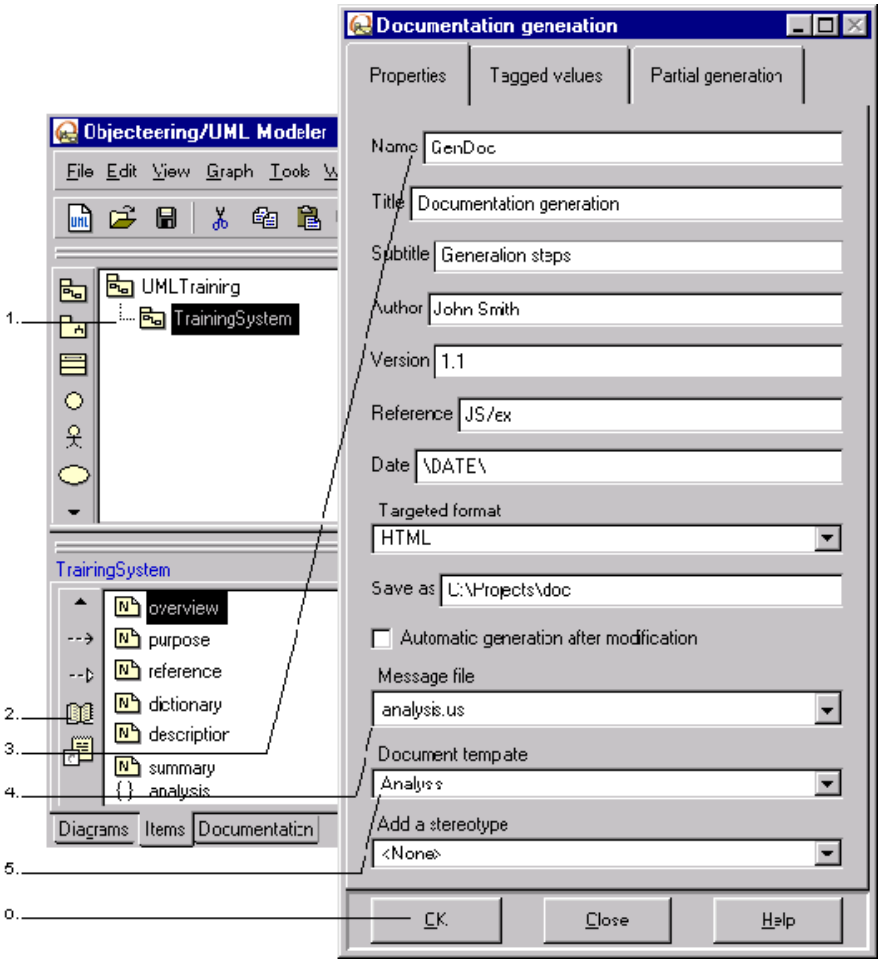


Figure 2-5. Creating a document work product

Steps:

1 - Select the "*TrainingSystem*" package.

2 - Click on the  "*Create a document*" button in the "*Items*" tab of the properties editor.

3 - Enter the necessary information in the fields (name, title, ...).

4 - Select the "*analysis.us*" message file.

5 - Select the "*Analysis*" template.

6 - Confirm.

7 - Right-click on the newly created document work product in the "*Items*" tab of the properties editor, and run the "*Documentation/Generate*" item from the context menu which then appears.

8 - Right-click on the newly created document work product in the "*Items*" tab of the properties editor, and run the "*Documentation/Visualize*" item from the context menu which then appears.

HTML format

This format can be generated and visualized in UNIX and Windows.

HTML documentation can be edited by all the web explorers. Hypertext links are used to browse, using the model's links to packages and classes.

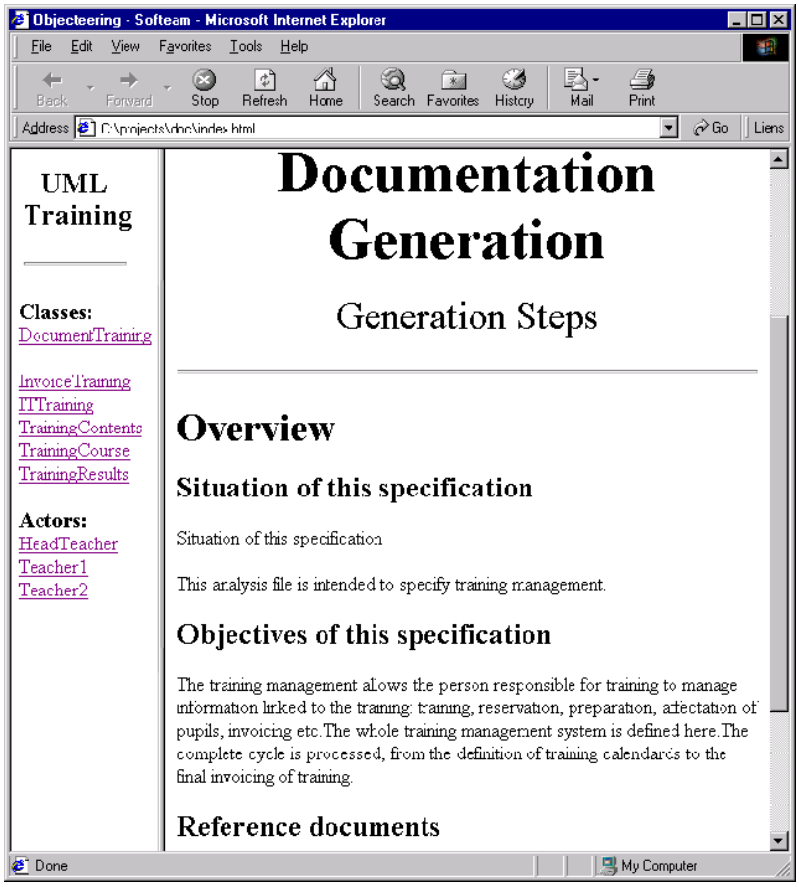


Figure 2-6. Documentation contents in Internet Explorer

"RTF" format

This format can be generated in UNIX and Windows, but can only be visualized in Windows.

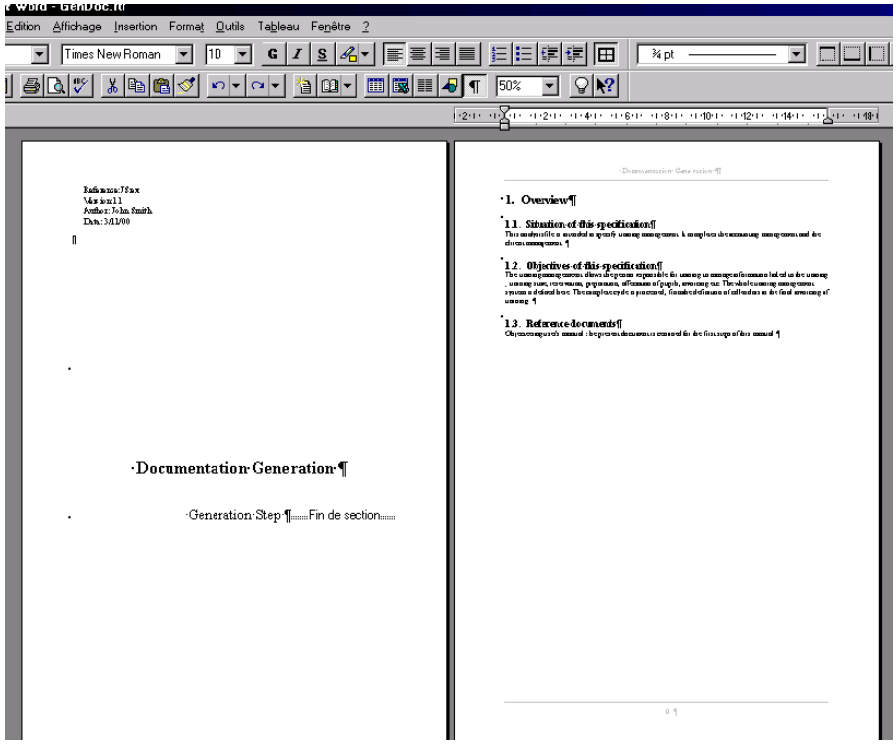


Figure 2-7. Documentation contents in Word

ASCII format

This format can be generated and visualized in UNIX and Windows.

In ASCII, the result, represented is Figure 2-8, is noticeably less attractive.

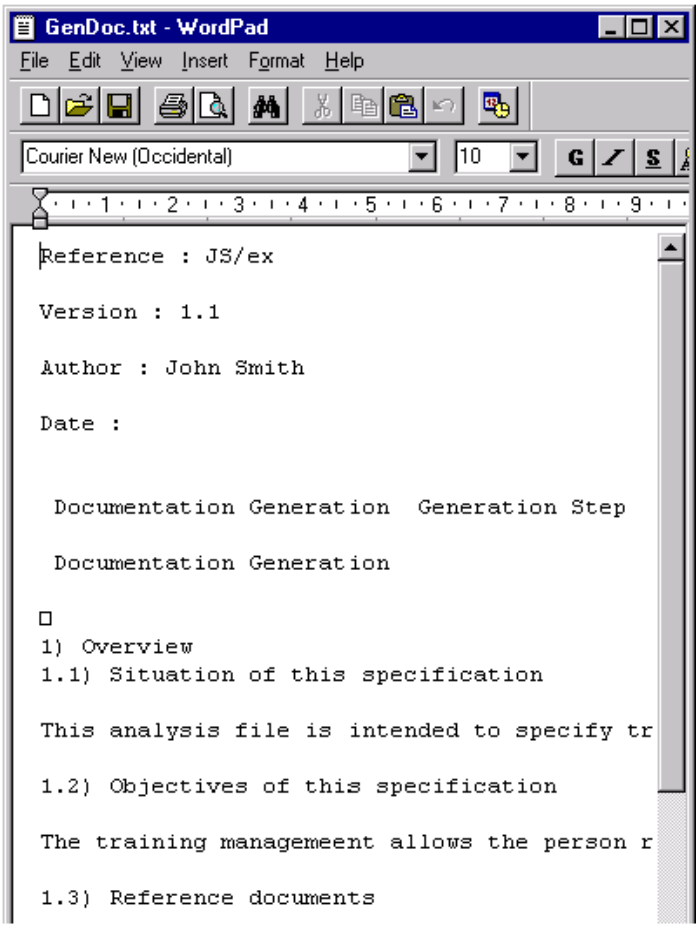


Figure 2-8. Documentation contents in ASCII

Postscript format

This format can only be generated and visualized in UNIX.

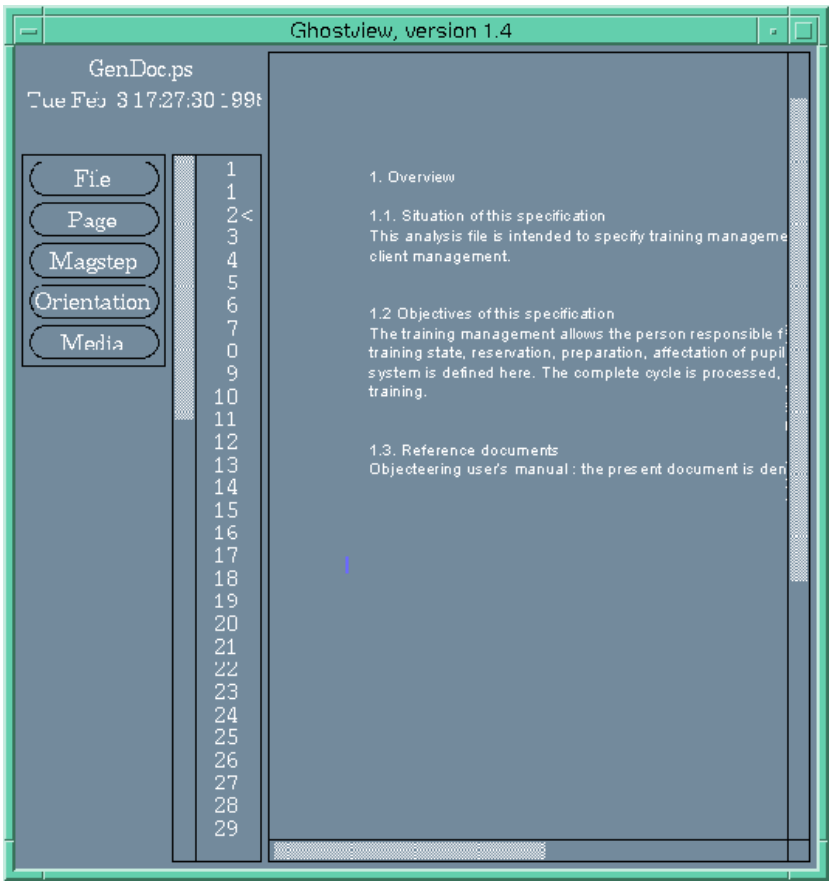


Figure 2-9. Documentation contents in Postscript

Partial documentation generation

Overview

Documentation can be generated partially, for example, at the level of a sub-package or a class. To carry out partial generation, proceed as in Figure 2-10, after having created the "*ResponsibleForTraining*" class.

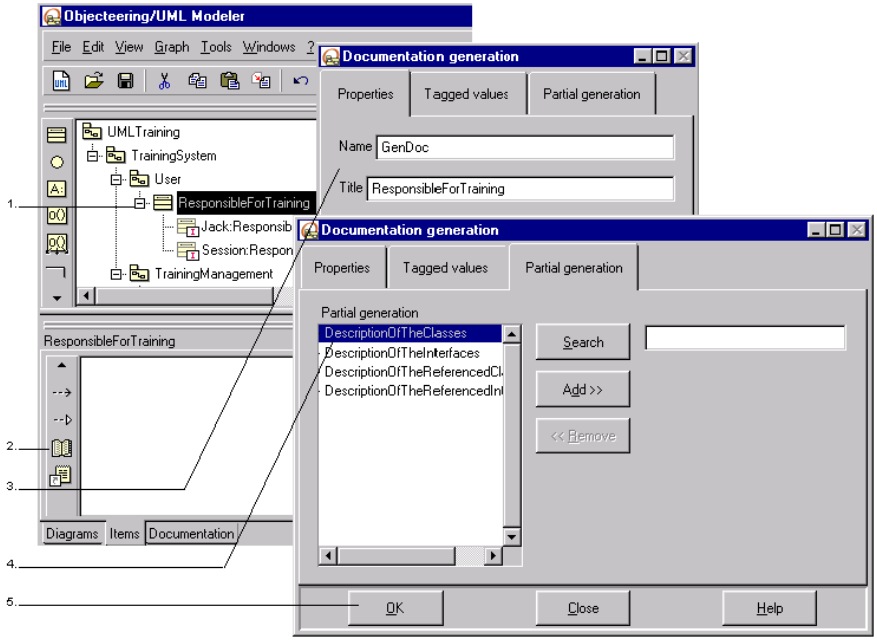


Figure 2-10. Partial generation process

Steps:

1 - Select the "*ResponsibleForTraining*" class.

2 - Click on the  "*Create a document*" button in the "*Items*" tab of the properties editor.

3 - Enter the relevant information in the fields in the dialog box.

4 - In the "*Partial generation*" tab, click on the "*Search*" button, then select one of the presented document template items and click on the "*Add*" button.

5 - Confirm and generate.

Chapter 3: General principles of documentation generation

Notes

Overview

Each model element can be documented by one or more notes or associated text zones. A note has:

- a name which is chosen from a list of possible names, according to the kind of element concerned. For example, a class can have note names like "*description*" or "*summary*". The name indicates the purpose of the entered note.
- a content, which is text freely entered by the user. Its nature must correspond to its purpose. For example, a text for "*summary*" contains a review of the element.

Note: New note types can be defined using the *UML Profile Builder* tool.

Example

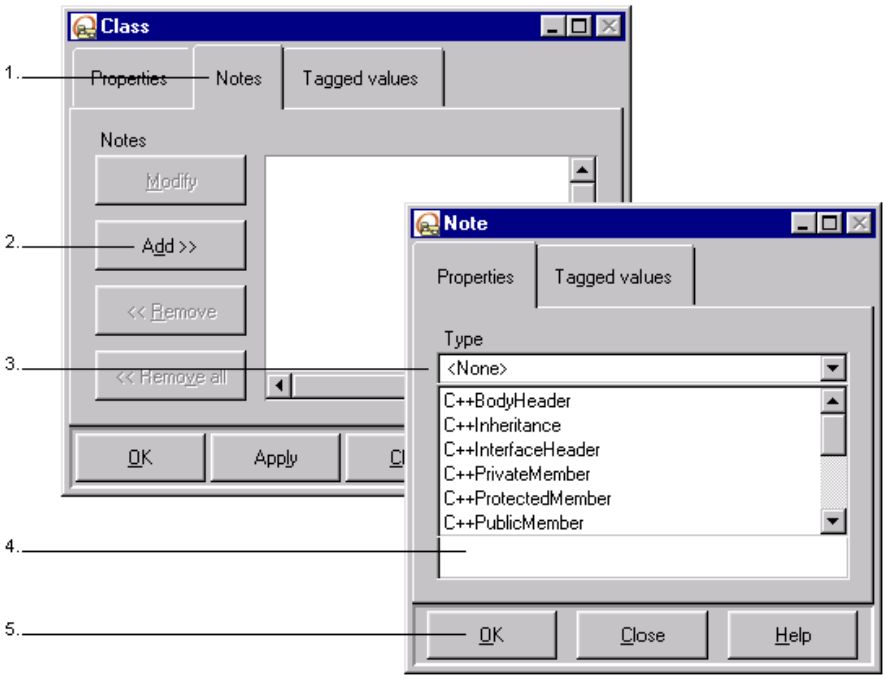


Figure 3-1. Entering a note for a class

Steps:

1 - Select the "*Notes*" tab in the element's dialog box.

2 - Click on "*Add*".

3 - Choose the type of note from the combobox.

4 - Enter the content.

5 - Confirm.

"Description" and "Summary" notes

Most document templates use these two note types:

- summary notes, which contain a short summary of the element, in less than a line, often used in numbered lists
- description notes, which contain the complete description of an element.

"*Summary*" notes are used when you refer briefly to the element. The detailed description of the element groups together the "*summary*" and "*description*" texts.

Note: Description and summary notes can be entered either via the "*Items*" tab or the "*Documentation*" tab of the properties editor. All other kinds of notes can be created via the "*Items*" tab. For further details, please refer to the "*Creating notes and tagged values*" section in chapter 2 of this user guide.

Example of description and summary notes

Here is an example of a typical document template which uses these two notes.

```
Chapter 2 : package S1
Chapter 2.1 : overview
   < summary S1 > < description S1 >
   classes of the package :
   C1 : < summary C1 >
   C2 : < summary C2 >
Chapter 2.2 : Class C1
   < summary C1 > < description C1 >
Chapter 2.3 : Class C2
   < summary C2 > < description C2 >
```

Characters generating bulleted lists

In order to generate bulleted lists in your documentation, certain characters are used within notes. These characters are specified at module parameter level (for further information, please refer to the "*Documentation configuration parameters*" section in chapter 4 of this user guide.

Let's imagine we had defined "*" as being a character used in the generation of bulleted lists. Figure 3-2 shows both the note containing these characters, and the result in the generated documentation.

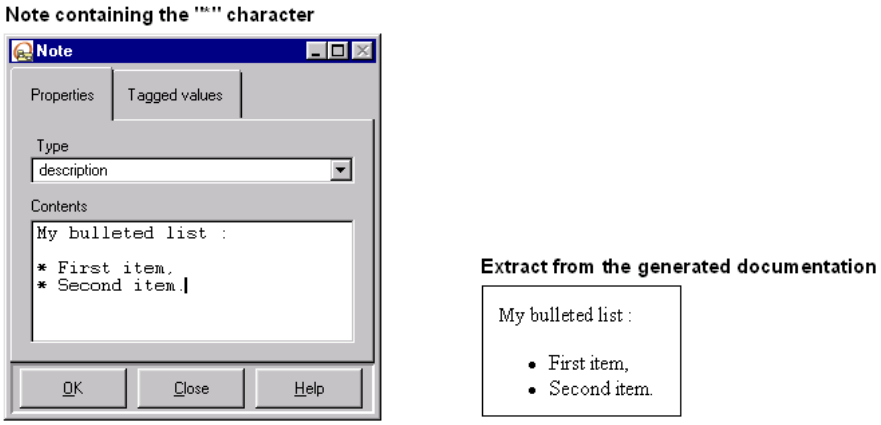


Figure 3-2. Generating a bulleted list

Tagged values

Overview

Almost all model elements can be annotated with tagged values. These can be used in documentation generation to:

- explicitly include a document item (for example, "*{analysis}*" tagged value in an analysis document).
- explicitly exclude a document item (for example, "*{noanalysis}*" tagged value in an analysis document).
- give an explicit nature to an element (for example, the "*{user}*" tagged value indicates that the class represents a user).

The meaning of tagged values depends on the document template used, which will specify those tagged values which allow the inclusion or exclusion of an element.

Note: The {*noanalysis*} and {*nodesign*} tagged values can be directly selected in the "*Documentation*" tab of the properties editor. For further details, please refer to the "*Creating notes and tagged values*" section in chapter 2 of this user guide.

Model structure

Overview

Documentation generation is entirely guided by the model's structure in terms of packages and classes. The numbering of the chapters, document layout and document composition are managed by the model's structure and the document template.

Documentation root

Documentation generation always starts from an element in the model, to generate all the elements structured by this element. Most of the time, the element concerned is a package, but it can also be a class, an actor, a use case, or another type of element.

Document package

A document package is a package which contains:

- the documentation root of a document
- the numbering of the chapters, the layout and the composition of this document
- general textual descriptions for the whole document ("*Introduction*", "*Reference documents*", etc)

Document template

Overview

A document template describes how to obtain a document from a model. It entirely conditions the nature of the produced document.

The document templates currently supplied are:

- the analysis document template, which is used to generate analysis documents
- the design document template, which is used to generate design documents

Document item

A document template also determines the documentation root. This is usually a package.

A document template is subdivided into "*document items*". These provide a special description for a given type of element. A document item describes each element or a set associated with a document item. For example, the analysis document template for a package (document package) is subdivided into "*document items*" which concern all the important types of elements, notably classes. If you wish to obtain the part of documentation which is relative to a class, you must designate in the analysis document template the document item relative to the class.

Message files

All messages generated by a document template (chapter title, list header, etc.) can be defined in an external file. The document work product must then define which file should be used.

Formatters

Overview

Formatters exist in order to adapt documentation to a particular processor or text editor. The user chooses a format (for example rtf), and Objecteering/UML produces a document in the selected format. It is, therefore, possible to:

- produce the same documentation in several different formats
- change the presentation rules for the same format

Text markers

The user can use a set of available markers. These can be used to:

- format a text (center, bold characters, etc.)
- give specific instructions (include a graph, introduce a new chapter, etc.)

These aspects are detailed in chapter 7 ("*Parameterization*") of this user guide.

Document templates are designed to save you the trouble of marking the text.

Consistency management

Review

Objecteering/UML checks the consistency of documentation generation work products with regard to the model. When generation is requested, the previously generated files are deleted.

In the event of generation being canceled, files are updated up to the next save.

Diagrams

Overview

In HTML format, all diagrams generated contain "clickable" zones. By clicking on a modeling element in an image, the file which describes this element can be opened. Only elements which have an external description in a file can be opened by clicking.

Example

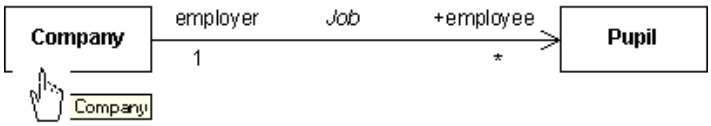


Figure 3-3. Class diagram

Steps:

1 - Click on the "*Company*" class. The file containing the description of this class is then edited.

Index generation

Overview

In HTML, an index containing all the elements browsed by the document template is generated. These elements are grouped by "*NameSpace*" and by alphabetical order within each group.

By clicking over the name which interests you, you can display its description in the right-hand window.

Example

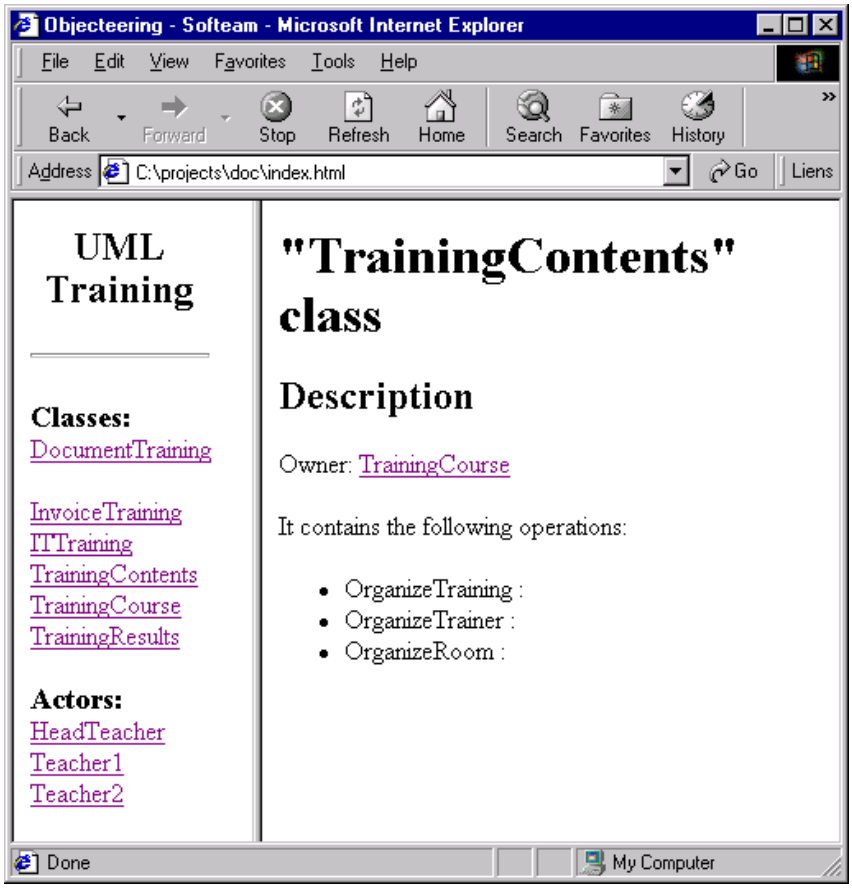


Figure 3-4. Generated index

Chapter 4: Generating documentation

Document description: Properties tab

Generation steps

Documentation is generated from a model which has already been built and documented.

"Properties" tab

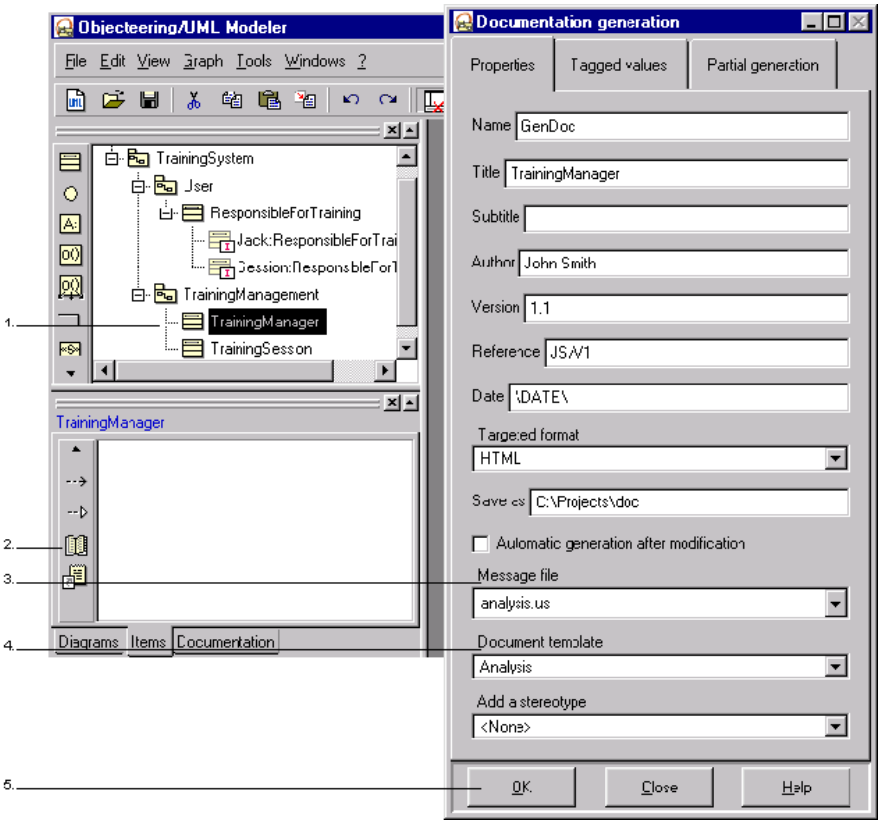


Figure 4-1. Generating a document

Steps:

1 - Select the initial element in the explorer.

2 - Click on the  "*Create a document*" button in the "*Items*" tab of the properties editor.

3 - Enter the relevant information in the dialog box which appears, and then select the message file.

4 - Select the document template.

5 - Confirm by clicking on "*OK*".

In the properties editor, double-click on the newly created document work product or right-click to open the context menu and then run the "*Generate*" command.

The ... field	**is used to**
Name	enter the name of the produced document. The name of the product corresponds to the name of the generated master file.
Title	enter the document title.
Subtitle	enter the document's subtitle.
Author	enter the name of the document's author.
Version	enter the document's version.
Reference	enter the reference number.
Date	enter the document's creation date. The document's date can be entered directly in this field. \ DATE\ inserts the current date.
Targeted format	choose the generated document's format (ASCII, Rtf, Postscript, HTML). Postcript is not available in Windows.
Save as	enter the path where the generated document will be stored. Note: the default path is defined in the "Configuration" window.
Automatic generation after modification	generate the document automatically after having closed the window (without selecting the "Generate" menu).
Message file	select a message file. The delivered document templates use external files. They allow the generation of documentation in English and French.
Document template	select a document template.
Add a stereotype	add a stereotype, previously defined at UML profiling project level, to the documentation generation.

Text formats

Files can be generated in different formats.

The ... format	platform ...
Postscript	UNIX
RTF	UNIX and Windows
Ascii	UNIX and Windows
HTML	UNIX and Windows

Image formats

It is also possible to include images in documents.

The ... format	platform ...	inclusion mode ...
Encapsulated Postscript (EPS)	UNIX	reference
EnhancedMetaFile (EMF)	Windows	reference
Graphics Interchange Format (GIF)	UNIX and Windows	reference

Please refer to chapter 4 ("*Generation principles*") of the *Objecteering/Document Template Editor* user guide for further information.

Partial generation

A document template is subdivided into several "*document items*". A document item corresponds to a special description of a model element (class, operation, etc.) within the document template.

For example, to generate a section of a document concerning a class from a document template related to a package, you must:

1 - Select the class.

2 - Click on the  "*Create a document*" icon in the "*Items*" tab of the properties editor.

3 - Enter the relevant information in the "*Properties*" tab, and select the document template.

4 - Select the "*Partial generation*" tab, click on the "*Search*" button and select a document item from those suggested (they all concern a class).

5 - Click on the "*OK*" button.

Documentation configuration parameters

Overview

Documentation directories (where the files are generated) and documentation editors can be parameterized in the configuration window (accessed by clicking on the  "*Modify module parameter configuration*" icon.

There are three sub-sets of parameters for the *Objecteering/Documentation* user guide (as shown in Figure 4-2, 4-3 and 4-4):

- General
- Editors
- RTF generation

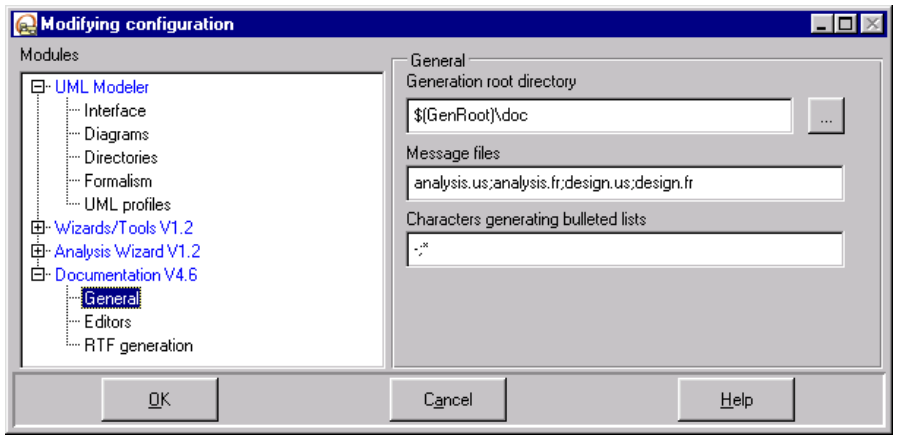


Figure 4-2. The "*General*" subset of "*Documentation*" parameters

The ... field	indicates ...
Generation root directory	the directory in which all the documentation files will be stored by default.
Message files	the files that contain the headers of the titles and messages of the documentation generation. This list will be proposed when the product is created. Each file must be separated by the ";" character.
Characters generating bulleted lists	the characters which are used in notes to identify text sections which are to be presented in the form of bulleted lists. These characters are separated by a ";".

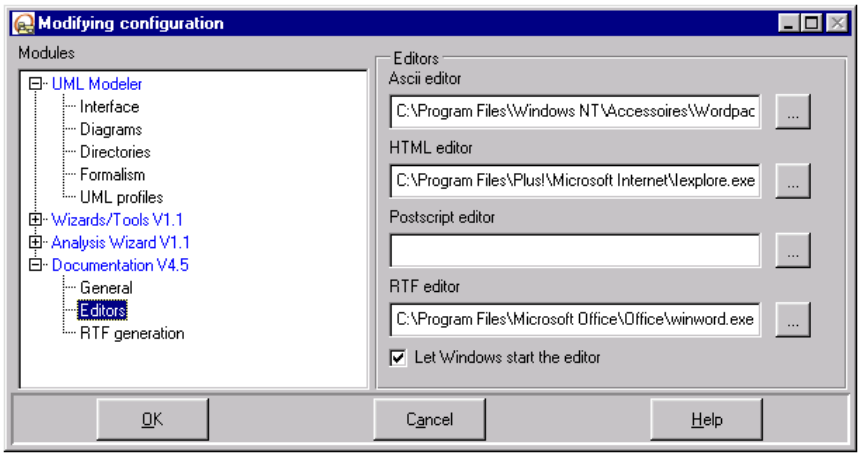


Figure 4-3. The "*Editors*" subset of "*Documentation*" parameters

The ... field	**indicates ...**
Ascii editor	the ASCII edition tool.
HTML editor	the tool which is used to visualize HTML documents (Netscape, Mosaïc, internet Explorer)
Postscript editor	the tool which is used to visualize a postscript document (ghostview).
RTF editor	the rtf edition tool.
Let Windows start the editor	whether Windows should determine which editor to run. If this tickbox is checked, the parameters which define editors are not used and Windows determines which editor is to be run, according to the type of file and on what has been defined in the registry.

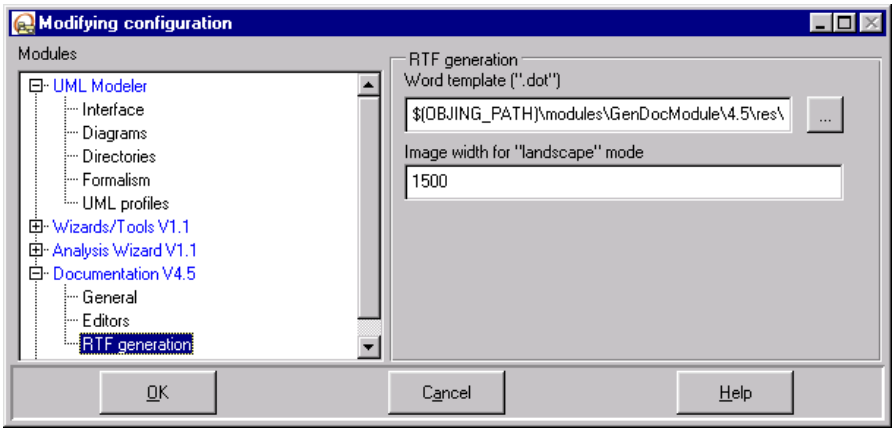


Figure 4-4. The "*RTF generation*" subset of "*Documentation*" parameters

The ... field	**indicates ...**
Word Template (".dot")	the Word model associated with the generated document.
Image width for "landscape" mode	the minimum width expressed in pixels, from which the page containing the diagram is generated in "landscape" mode. This parameter is only used if the width is greater than the height of the diagram.

Note: When a parameter does not contain an editor's absolute path, the user must define this editor's path in the "PATH" environment variable.

Document description: Partial generation tab

Document description: "Partial generation" tab

This tab is used to generate documentation for a type of element, using a document template which describes another type of element.

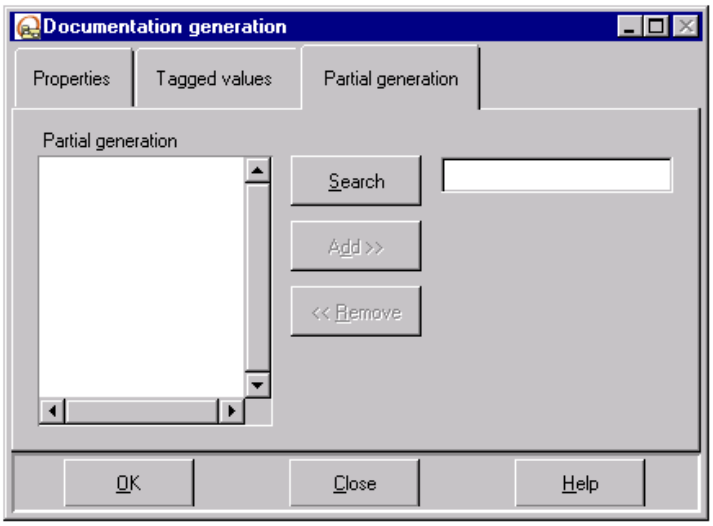


Figure 4-5. The "*Partial generation*" tab of the "*Documentation generation*" box

The ... button	is used to ...
Search	search all the document items belonging to the selected document template (in the "*document template*" field of the "*Properties*" tab) which describes the same kind of item as that associated with the documentation work product.
Add	specify the document item used for partial generation.
Remove	remove the document item used for partial generation.

Example of documentation generation

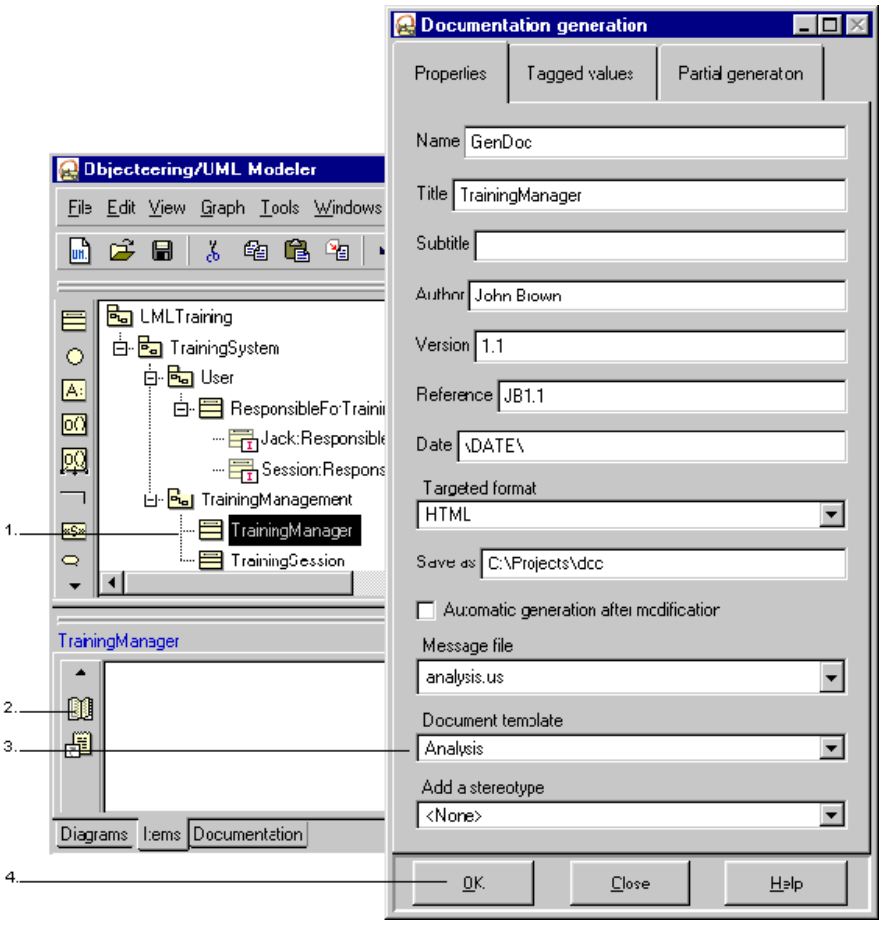


Figure 4-6. Creating a document in the "*TrainingManager*" class

Steps:

1 - Select the element.

2 - Click on the  "*Create a document*" icon in the "*Items*" tab of the properties editor.

3 - Enter the relevant information in the entry dialog box.

4 - Confirm.

Chapter 5: Analysis document template

Analysis document template - Overview

Presentation

The analysis document template delivered as standard with Objecteering/UML corresponds to a modeling procedure, which presents information in a particular form. This procedure is described below.

Class discovery techniques

Three main techniques are used to discover classes:

- *the global approach*: through flow diagrams between packages which represent the future system, we break this system down into sub-systems, and determine what represents, sends or receives the flows which designate classes in these sub-systems
- the *dictionary*: from terminology of the problem dealt with, provided in a dictionary, we define which model elements represent these terms
- *needs approach*: we identify *users* of the system represented by classes, and their needs represented by operations. We determine which model elements respond to these needs. "*Use Cases*" (group of sequences which implement actors or users) are used at this stage.

Note: According to the nature of the problem dealt with, only some of these techniques may apply.

Principles of the document template

1 - The analysis document starts with a context overview.

2 - Methodology entry points are presented: glossary or dictionary, analysis overview (packages graph, ...), definition of users and their needs (use case diagrams).

3 - Analysis notions themselves, that is to say non "*user*" classes of the model, are developed. They are structured through packages.

4 - The traceability between specified notions and retained entry points is provided: dictionary/model, users/needs, data flows/system notions.

Models used

"*Actors*", who represent users of the system, are distinguished from classes, which define the system.

This use case model describes needs.

The object model is used everywhere.

The operating model (pre/post-conditions, protocol state diagrams) is essentially used on interface classes.

The dynamic model is used on "*actors*", but very little on classes. We are more interested in "*how do the users do*" than in *"how do the system's classes do*". For classes, only the dynamic parts which are visible and important to users are represented. Sequence diagrams are used to describe use cases between objects.

Information

The analysis document template uses an external messages file, which must be indicated in the document work product, so that the generation may suitably translate chapter titles. Two files ("*analysis.us*" and "*analysis.fr*") are delivered as standard, to generate documentation in English or French. You can also define your own titles by creating a new message file.

Contents

The "Overview" chapter

Contents

The "*Overview*" chapter contains the following chapters:

- Situation of this specification
- Objectives of this specification
- Reference documents

"Situation of this specification" chapter

This chapter describes the situation of the analysis document (positioning with regard to other analysis documents, requirements specifications, indication of associated software parts).

This information is entered in the document package's overview notes.

"Objectives of this specification" chapter

This chapter describes the aim of this specification, fundamental needs met and the overall specification plan.

This information is entered in the document package's purpose notes.

"Reference documents" chapter

This chapter provides the list of documents on which the current document is based, and to what extent.

This information is entered in the document package's reference notes.

The "Preliminary specification" chapter

Contents

The "*Preliminary specification*" chapter contains the following chapters:

- Dictionary
- Overview of the application
- Summary

"Dictionary" chapter

This provides the list of terms used in the document, accompanied by their definition.

This information is entered in the dictionary notes of the document package.

"Overview of the application" chapter

This chapter contains:

- summary and description notes of the document package
- class diagrams (with their description notes)

"Summary" chapter

This chapter contains the list of:

- packages annotated *{usecase}* (with their summary notes)
- packages not annotated *{usecase}* (with their summary notes)
- classes (with their summary notes)
- interfaces (with their summary notes)
- referenced elements (with their summary notes)

The "Definition of the use cases" chapter

Contents

The "*Definition of the use cases*" chapter contains:

- packages annotated *{usecase}*

Note: All packages defined in the package's hierarchy, for which generation has been run, are generated. For example, if generation is run on the UML modeling project, then all the UML modeling project's packages annotated *{usecase}* will be generated.

The "Detailed specification" chapter

Contents

The "*Detailed specification*" chapter contains:

- packages
- classes
- interfaces
- referenced packages annotated *{usecase}* (only in HTML)
- referenced packages not annotated *{usecase}* (only in HTML)
- referenced classes (only in HTML)
- referenced interfaces (only in HTML)

Note: All packages defined in the package's hierarchy, for which generation has been run, are generated. For example, if generation is run on the project, then all the project's packages not annotated *{use case}* will be generated.

The "Use cases" chapter

Contents

The "*Use cases*" chapter contains:

- a "*Description*" chapter
- actors
- use cases

"Description" chapter

This chapter contains:

- use case diagrams (with their description notes)
- the list of actors (with their summary notes)
- the list of use cases (with their summary notes)

The "Examples" chapter

Contents

The "*Examples*" chapter contains:

- a "*Description*" chapter
- instances

"Description" chapter

This chapter contains:

- the list of instances (with their description notes)
- object diagrams (with their description notes)

Description of a package

Contents

The description of a package contains:

- a "*Description*" chapter
- a "*Use cases*" chapter
- an "*Examples*" chapter
- collaborations
- state machines
- activity graphs
- classes
- interfaces
- referenced packages (only in HTML)
- referenced classes (only in HTML)
- referenced interfaces (only in HTML)

"Description" chapter

This chapter contains:

- the name of the owner package
- the contents of the summary and description notes
- the list of parent packages (with their summary notes)
- the list of packages used (with their summary notes)
- class diagrams (with their description notes)
- invariants
- the list of enumerates (with their description notes)
- the list of types (with their description notes)
- the list of received dataflows (with their description notes)
- the list of sent dataflows (with their description notes)
- the list of packages (with their summary notes)
- the list of classes (with their summary notes)
- the list of interfaces (with their summary notes)
- the list of referenced elements (with their summary notes)

"Use cases" chapter

This chapter contains:

- a "*Description*" chapter which allows you to display the use case diagrams (with their description notes), the list of actors (with their summary notes), the list of use cases (with their summary notes)
- actors
- use cases

"Examples" chapter

This chapter contains:

- a " *Description* " chapter, which allows you to display the list of instances (with their description notes), the object diagrams (with their description notes)
- instances

Description of a class or an interface

Contents

The description of a class contains:

- a " *Description*" chapter
- an "*Examples*" chapter
- collaborations
- state machines
- operations
- classes
- interfaces

Note: The generation of an interface is absolutely identical to the generation of a class. However, the main interface generation title is "*Interface <className>*" instead of "*Class <className>*".

"Description" chapter

This chapter contains:

- the component name
- summary and description notes
- the list of parent classes
- invariants
- class diagrams (with their description notes)
- the list of enumerates (with their description notes)
- the list of types (with their description notes)
- the list of received dataflows (with their description notes)
- the list of sent dataflows (with their description notes)
- the list of operations (with their summary notes)
- the list of roles (with their description notes)
- the list of attributes (with their description notes)
- the list of classes (with their summary notes)
- the list of interfaces (with their summary notes)

"Examples" chapter

This chapter contains:

- a " *Description* " chapter, which allows you to display the list of instances (with their description notes) and the object diagrams (with their description notes)
- instances

Description of a collaboration

Contents

The description of a collaboration contains:

- the list of instances (with their description notes)
- sequence diagrams (with their description notes)

Generating instances

All the instances of the collaboration are displayed in the form of a bulleted list.

The title contains the name of the instance, the ":" character, and then the name of its class (in the case of HTML generation, a hypertext link leads to the file describing the added class).

The instance's description notes are inserted.

Description of a state machine

Contents

The description of a state machine contains:

- the list of states (with their description notes)
- state diagrams (with their description notes)

Generating states

All state machine states are displayed in the form of a bulleted list.

The title contains the name of the parent state, the "::" character, and then the name of the state.

The state's description notes are inserted.

Description of an activity graph

Contents

The description of an activity graph contains:

- the list of action states (with their description notes)
- the list of sub-activity states (with their description notes)
- the list of object flow states (with their description notes)
- the list of partitions (with their description notes)
- the list of transitions (with their description notes)
- activity diagrams (with their description notes)

Description of a use case

Contents

The description of a use case contains:

- a “*Description*” chapter
- operations
- collaborations
- activity graphs

"Description" chapter

This chapter contains:

- the name of the component
- summary and description notes
- the list of parent use cases (with their summary notes)
- the list of use cases used (with their summary notes)
- the list of "extend" use cases (with their summary notes)
- the list of intervening actors (with their summary notes)
- the list of collaborations (with their summary notes)
- the list of attributes (with their summary notes)
- the list of operations (with their summary notes)

Description of an actor

Contents

The description of an actor contains:

- a " *Description* " chapter
- operations

"Description" chapter

This chapter contains:

- the component name
- summary and description notes
- the list of parent actors (with their summary notes)
- the list of cooperating use cases (with their summary notes)
- the list of cooperating actors (with their summary notes)
- the list of attributes (with their summary notes)
- the list of operations (with their summary notes)

Description of an operation

Contents

The description of an operation contains:

- summary and description notes
- the list of parameters (with their description notes)
- return parameters (with their summary notes)
- pre-conditions
- post-conditions

Description of an instance

Contents

The description of an instance contains:

- description notes
- the list of attributes and their values (with their description notes)
- the list of links (with their description notes)

Partial generation

Overview

The analysis document template allows the partial generation of the analysis document. To carry out this partial generation, go into the "*Partial generation*" tab and select a particular document element of the document template (for further information, please refer to chapter 4 of this user guide).

These document elements are used to generate:

- the "*Overview*" chapter
- the "*Preliminary specification*" chapter
- the "*Definition of the use cases*" chapter
- the "*Detailed specification*" chapter
- user packages
- non-user packages
- the "Use cases" chapter
- classes
- interfaces
- the class or interface "*Use cases*" chapter
- actors
- use cases

"Overview" document item

This is used to generate the "*Overview*" chapter.

"PreliminarySpecification" document item

This is used to generate the "*PreliminarySpecification*" chapter.

"DefinitionOfTheUseCases" document item

This is used to generate the "*Definition of the use cases*" chapter.

"DetailedSpecification" document item

This is used to generate the "*Detailed specification*" chapter.

"DescriptionOfTheUserPackages" document item

This is used to generate the description of all the sub-packages annotated *{usecase}*.

"DescriptionOfTheNonUserPackages" document item

This is used to generate the description of all the sub-packages not annotated *{usecase}*.

"PackageUseCase" document item

This is used to generate the "*Use cases*" chapter for packages.

"DescriptionOfThePackageActors" document item

This is used to generate the description of an actor of a package.

"DescriptionOfTheClasses" document item

This is used to generate the description of a class, and does not function with interfaces.

"DescriptionOfTheInterfaces" document item

This is used to generate the description of an interface, and does not function with classes which are not interfaces.

"DescriptionOfThePackageUseCases" document item

This is used to generate the description of a package use case.

Chapter 6: Design document template

Design document template - Overview

Information

The design document template uses an external messages file. This file has to be defined in the document work product, so that chapter titles can be properly translated during generation. Two files ("*design.us*" and "*design.fr*") are delivered as standard, to generate documentation in English or French. You can also define your own titles by creating a new message file.

Contents

4 Traceability

5 Integration

The "Overview" chapter

Content

The "*Overview*" chapter contains the following chapters:

- Situation of this design file
- Aim of the technical design
- Reference documents
- Dictionary

"Situation of this design file" chapter

This chapter describes the situation of the design file.

This information is entered in the overview notes of the document package.

"Aims of the technical design" chapter

This chapter describes the purpose of the design file.

This information is entered in the purpose notes of the document package.

"Reference documents" chapter

This chapter provides the list of documents to which the current document is linked and to what extent.

This information is entered in the reference notes of the document package.

"Dictionary" chapter

This chapter supplies the list of terms used in the document, and their definition. It is used when the design addresses a module that is separate from the specification (library, generic element, etc.).

This information is entered in the dictionary notes of the document package.

Architecture

Contents

The "*Architecture*" chapter contains the following chapters:

- Implementation constraints
- Technical architecture
- Implementation principles

"Implementation constraints" chapter

This chapter describes all the application's implementation constraints which condition the architecture.

This information is entered in the design_constraint notes of the document package.

"Technical architecture" chapter

This chapter describes the application's general architecture, as well as its integration in a larger-scale architecture.

It contains:

- the design_architecture notes
- class diagrams annotated *{design_architecture}* (with their description notes)
- the list of nodes (with their summary notes)
- the list of components (with their summary notes)
- deployment diagrams (with their description notes)

"Implementation principles" chapter

This chapter defines the implementation principles that have been chosen and corresponds to the implementation of the *UML Profile Builder* tool.

It contains:

- the design_principles notes
- class diagrams annotated *{design_principles}* (with their description notes)
- node instances (with their summary notes)
- deployment instance diagrams (with their description notes)

The "General design" chapter

Contents

The "*General design*" chapter contains:

- a "*Description*" chapter
- an " *Essential elements of this design*" chapter
- a "*Use cases*" chapter
- an "*Examples*" chapter
- collaborations
- collaborations
- activity graphs
- sub-systems
- packages
- classes
- interfaces
- components
- nodes
- referenced packages (only in HTML) and sub-systems
- referenced classes (only in HTML)
- referenced interfaces (only in HTML)
- referenced components (only in HTML)
- referenced nodes (only in HTML)

Note: All the packages defined in the package's hierarchy, for which generation has been run, are generated. For example, if generation is run on the UML modeling project, then all the UML modeling project's packages will be generated.

"Description" chapter

This chapter contains the following elements defined on the document package:

- the summary and description notes
- class diagrams not annotated *{design_architecture}*, *{design_integration}*, *{design_principles}* tagged values

"Essential elements of this design" chapter

This chapter contains the list of:

- sub-systems
- packages
- classes
- interfaces
- collaborations
- referenced elements

"Use Cases" chapter

This chapter contains:

- a "*Description*" chapter, which allows you to display the use case diagrams (with their description notes), the list of actors (with their summary notes) and the list of use cases (with their summary notes)
- actors
- use cases

"Examples″ chapter

This chapter contains:

- a "*Description*" chapter, which allows you to display the list of instances (with their summary notes), and object diagrams (with their description notes)
- instances

Traceability

Objective

This chapter allows the definition of links between the design elements and the specification elements. This allows you to justify the design and to make sure it is exhaustive.

Content

This information is entered in the design_traceability notes of the document package.

Integration

Objective

This chapter defines the application's integration plan. It determines how and in which order the components, necessary for the obtaining of the application and identified during the preliminary design phase, must be assembled.

The integration is based on the packages' class diagrams, which allow the representation of each integration step, by modeling groups of components of the system, and components specifically dedicated to the integration tests.

Content

This information is entered in the design_integration notes of the document package or in the class diagrams annotated by the *{design_integration}* tagged value.

Description of a package

Contents

The description of a package contains:

- a "*Description*"" chapter
- a "*Use cases*" chapter
- an "*Examples*" chapter
- an "*Implementation*" chapter
- collaborations
- state machines
- activity graphs
- classes
- interfaces
- components
- nodes
- referenced packages (only in HTML)
- referenced classes (only in HTML)
- referenced interfaces (only in HTML)
- referenced components (only in HTML)
- referenced nodes (only in HTML)

"Description" chapter

This chapter contains:

- the name of the owner package
- summary and description notes
- the list of tagged values
- stereotypes
- the list of constraints
- the list of parent packages (with their summary notes)
- the list of used packages (with their summary notes)
- class diagrams (with their description notes)
- invariants
- the list of enumerates (with their description notes)
- the list of types (with their description notes)
- the list of received dataflows (with their description notes)
- the list of sent dataflows (with their description notes)
- the list of packages (with their summary notes)
- the list of classes (with their summary notes)
- the list of interfaces (with their summary notes)
- the list of collaborations (with their description notes)
- the list of state machines (with their description notes)
- the list of referenced elements (with their summary notes)

"Use cases" chapter

This chapter contains:

- a "*Description*" chapter, which allows you to display use case diagrams (with their description notes), the list of actors (with their summary notes) and the list of use cases (with their summary notes)
- actors
- use cases

"Examples" chapter

This chapter contains:

- a "*Description*" chapter which allows you to display the list of instances (with their summary notes) and the object diagrams (with their description notes)
- instances

"Implementation" chapter

This chapter contains:

- an "*Architecture*" chapter which allows you to display the list of nodes (with their summary notes), the list of components (with their summary notes) and the deployment diagrams (with their description notes).
- an "*Examples*" chapter which allows you to display the list of node instances (with their description notes), the list of component instances (with their description notes) and the deployment instance diagrams (with their description notes).

Example

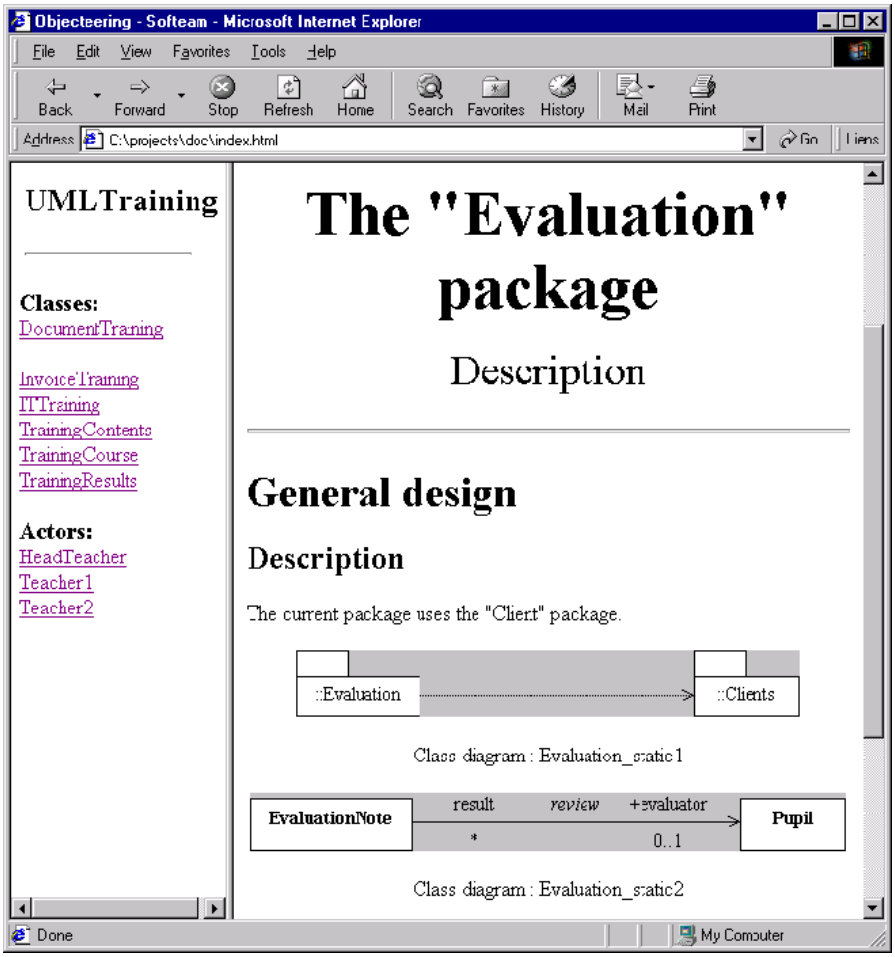


Figure 6-1. Example of generating a package

Description of a class or an interface

Contents

The description of a class contains:

- a "*Description*" chapter
- an "*Examples*" chapter
- collaborations
- state machines
- activity graphs
- operations
- classes (only in HTML)
- interfaces (only in HTML)

Note: Generating an interface is done in exactly the same way as generating a class. However, the main title of an interface generation is "*Interface <className>*" instead of "*Class <className>*".

"Description" chapter

This chapter contains the list of:

- the name of the owner
- the summary and description notes
- the list of tagged values
- stereotypes
- the list of constraints
- the list of parent classes (with their summary notes)
- the list of classes used (with their summary notes)
- class diagrams (with their description notes)
- invariants
- the list of implemented interfaces (with their summary notes)
- the list of enumerates (with their description notes)
- the list of types (with their description notes)
- the list of received dataflows (with their description notes)
- the list of sent dataflows (with their description notes)
- the list of operations in syntactical form (with their summary notes)
- the list of roles in syntactical form (with their description notes)
- the list of attributes in syntactical form (with their description notes)
- the list of classes (with their summary notes)
- the list of interfaces (with their summary notes)
- the list of collaborations (with their description notes)
- the list of state machines (with their description notes)

"Examples" chapter

This chapter contains:

- a "*Description*" chapter, allowing you to display the list of instances (with their summary notes) and object diagrams (with their description notes)
- instances

Example

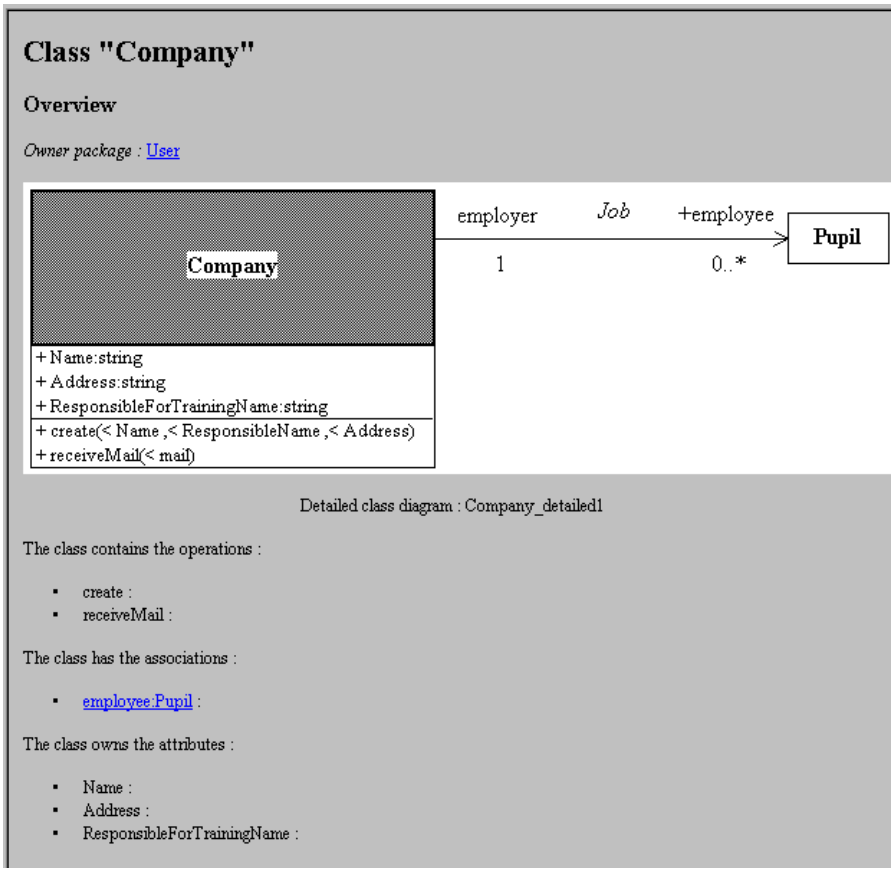


Figure 6-2. Example of class generation

Generating an association

The description of an association role contains:

- syntax
- tagged values
- description notes

The syntax of an association role contains:

- the name of the association
- the minimum and maximum multiplicity of the two links
- the names of the roles of the two links
- the name of the destination class (if the class is a class modeled in Objecteering/UML, a hypertext link towards the file describing this class is inserted.)

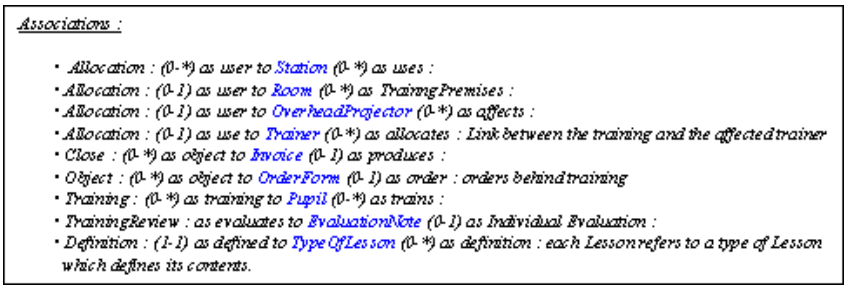


Figure 6-3. Example of the generation of an association

Description of a collaboration

Content

The description of a sequence contains:

- description notes
- the list of tagged values
- the stereotype
- the list of constraints
- the list of instances (with their description notes)
- the list of sequence messages exchanged between instances (with their description notes)
- sequence diagrams (with their description notes)
- collaboration diagrams (with their description notes*)*

Generation of instances

All the sequence instances are displayed in the form of a bulleted list.

The title contains the instance name, the ":" character, followed by the name of its class (in the case of the HTML generation, a hypertext link towards the file which describes the class or the actor is added).

The instance's description notes are inserted.

Generating messages

All the messages exchanged by the sequence instances are displayed in the form of a bulleted list.

The title contains the origin instance (see "*Generation*" of Instances), the destination instance and the ":" character, followed by the message name.

The message's description notes are inserted.

Generating sequence diagrams

All diagrams defined on the sequence are displayed followed by their description notes.

Generating sequence diagrams: all collaboration diagrams defined on the sequence are displayed followed by their description notes.

Example

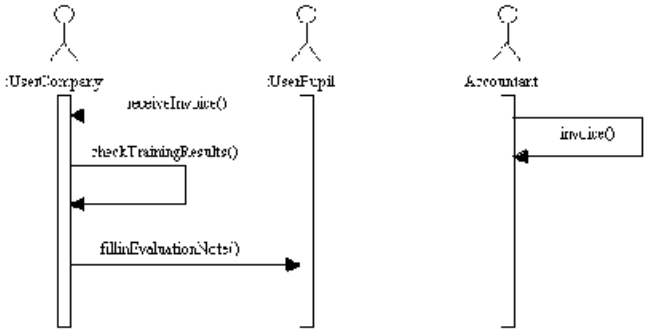
"ProcessingInvoice" Sequence

Instances :

- :UserCompany
- :UserPupil
- :Accountant

Exchanged messages :

- Accountant to_: Accountant : invoice :
- Accountant to : UserCompany : receiveInvoice :
- UserCompany to : UserCompany : checkTrainingResults :
- UserCompany to : UserPupil : fillinEvaluationNote :



ProcessingInvoice : Sequence Diagram

When the invoice is received, it must be ascertained that the pupil has had satisfactory training before paying.

Figure 6-4. Example of sequence generation

Description of a state machine

Contents

The description of a state machine contains:

- the redefinition of the state machine (a sentence which contains the name of the parent state machine and its class is generated when the state machine is a redefined state machine)
- description notes
- the list of tagged values
- stereotypes
- the list of constraints
- the list of states (with their description notes)
- the list of transitions (with their description notes)
- state diagrams (with their description notes)

Generating states

All the state machine's states are displayed in the form of a bulleted list.

The title contains the name of the parent state, the "::" character, followed by the state name.

The name of the parent state is built in the same way, i.e., the name of the parent state, the "::" character and the name of the state.

The state's "description" notes are inserted.

Generating transitions

All the state machine's transitions are displayed in the form of bulleted lists.

The title contains the name of the source state and the name of the arrival state.

The transition's description notes are inserted.

Generating state diagrams

All diagrams defined on the state machine are displayed following their description notes (Figure 6-5).

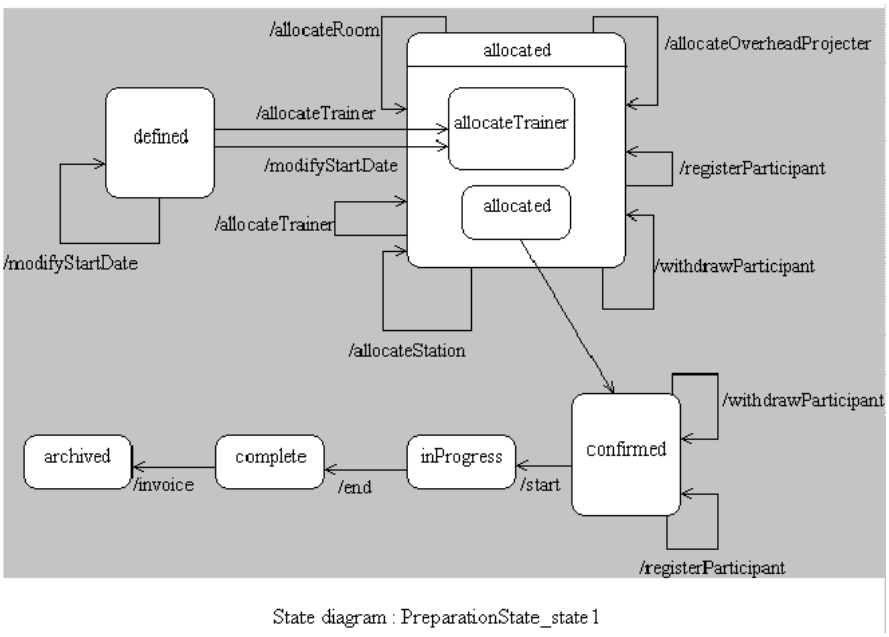


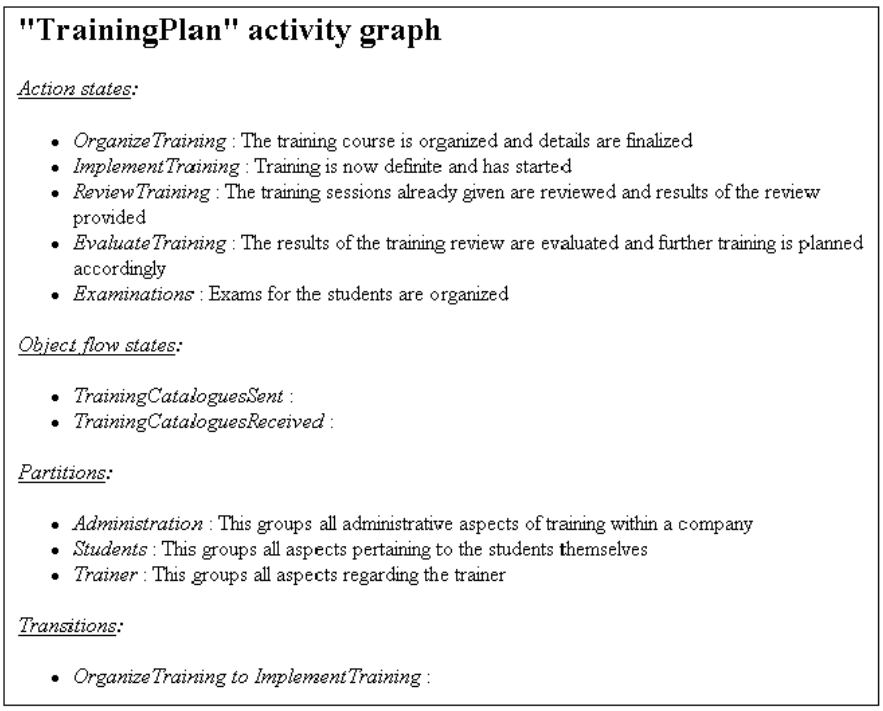
Figure 6-5. Example of generating a state machine

Description of an activity graph

Contents

The description of an activity graph contains:

- the list of action states (with their description notes)
- the list of sub-activity states (with their description notes)
- the list of object flow states (with their description notes)
- the list of partitions (with their description notes)
- the list of transitions (with their description notes)
- activity diagrams (with their description notes)



"TrainingPlan" activity graph

Action states:

- *OrganizeTraining* : The training course is organized and details are finalized
- *ImplementTraining* : Training is now definite and has started
- *ReviewTraining* : The training sessions already given are reviewed and results of the review provided
- *EvaluateTraining* : The results of the training review are evaluated and further training is planned accordingly
- *Examinations* : Exams for the students are organized

Object flow states:

- *TrainingCataloguesSent* :
- *TrainingCataloguesReceived* :

Partitions:

- *Administration* : This groups all administrative aspects of training within a company
- *Students* : This groups all aspects pertaining to the students themselves
- *Trainer* : This groups all aspects regarding the trainer

Transitions:

- *OrganizeTraining to ImplementTraining* :

Figure 6-6. Example of generating an activity graph

Description of a use case

Content

The description of a use case contains:

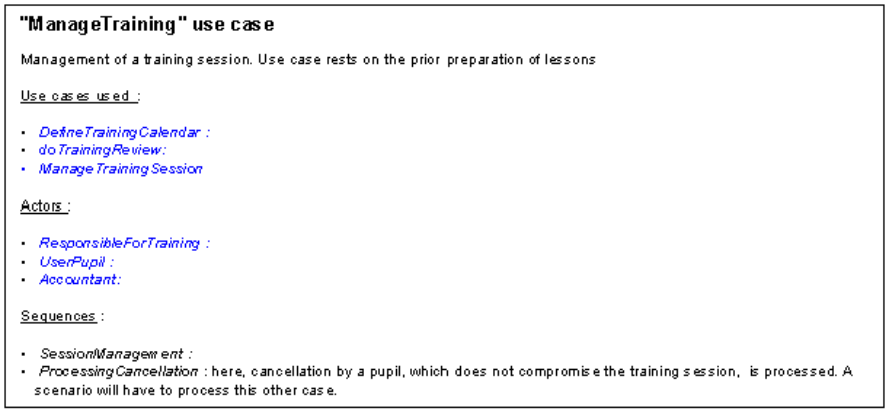
- a "*Description*" chapter
- operations
- collaborations
- state machines
- activity graphs

"Description" chapter

This chapter contains:

- the name of the owner
- summary and description notes
- the list of tagged values
- stereotypes
- the list of constraints
- the list of parent use cases (with their summary notes)
- the list of included use cases (with their summary notes)
- the list of intervening actors (with their summary notes)
- the list of operations in syntactic form (with their summary notes)
- the list of attributes in syntactic form (with their summary notes)
- the list of collaborations (with their summary notes)
- the list of state machines (with their description notes)
- sequence diagrams (with their description notes)

Example



"ManageTraining" use case

Management of a training session. Use case rests on the prior preparation of lessons

Use cases used :

- *DefineTrainingCalendar :*
- *doTrainingReview:*
- *ManageTrainingSession*

Actors :

- *ResponsibleForTraining :*
- *UserPupil :*
- *Accountant:*

Sequences :

- *SessionManagement :*
- *ProcessingCancellation* : here, cancellation by a pupil, which does not compromise the training session, is processed. A scenario will have to process this other case.

Figure 6-7. Example of generating use cases

Description of an actor

Contents

The description of an actor contains:

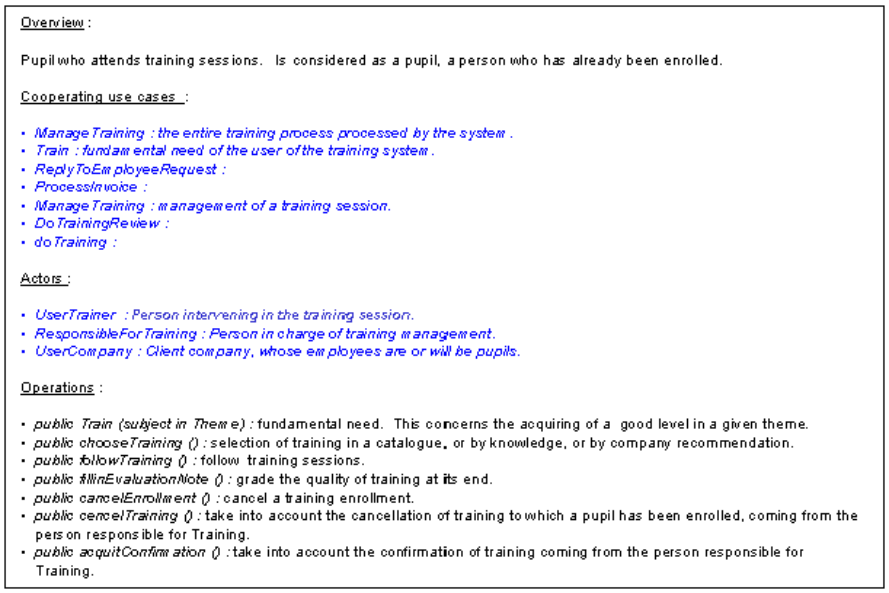
- a "*Description*" chapter
- operations
- activity graphs

The " Description " chapter

This chapter contains:

- the name of the owner
- summary and description notes
- the list of tagged values
- stereotypes
- the list of constraints
- the list of parent actors (with their summary notes)
- the list of cooperating use cases (with their summary notes)
- the list of cooperating actors (with their summary notes)
- the list of attributes in syntactic form (with their summary notes)
- the list of operations in syntactic form (with their summary notes)

Example: The "UserPupil" actor



Overview :

Pupil who attends training sessions. Is considered as a pupil, a person who has already been enrolled.

Cooperating use cases :

- *ManageTraining : the entire training process processed by the system .*
- *Train : fundamental need of the user of the training system .*
- *ReplyToEmployeeRequest :*
- *ProcessInvoice :*
- *ManageTraining : management of a training session.*
- *DoTrainingReview :*
- *doTraining :*

Actors :

- *UserTrainer : Person intervening in the training session.*
- *ResponsibleForTraining : Person in charge of training management.*
- *UserCompany : Client company, whose employees are or will be pupils.*

Operations :

- *public Train (subject in Theme) :* fundamental need. This concerns the acquiring of a good level in a given theme.
- *public chooseTraining () :* selection of training in a catalogue, or by knowledge, or by company recommendation.
- *public followTraining () :* follow training sessions.
- *public fillInEvaluationNote () :* grade the quality of training at its end.
- *public cancelEnrollment () :* cancel a training enrollment.
- *public cancelTraining () :* take into account the cancellation of training to which a pupil has been enrolled, coming from the person responsible for Training.
- *public acquitConfirmation () :* take into account the confirmation of training coming from the person responsible for Training.

Figure 6-8. Example of generating an actor

Description of a component

Contents

The description of a component contains:

- a "*Description*" chapter
- operations
- components
- activity graphs

"Description" chapter

This chapter contains:

- the name of the owner
- summary and description notes
- the list of tagged values
- stereotypes
- the list of constraints
- the list of used elements (with their summary notes)
- the list of implemented elements (with their summary notes)
- the list of received dataflows (with their description notes)
- the list of sent dataflows (with their description notes)
- the list of operations in syntactic form (with their summary notes)
- the list of attributes in syntactic form (with their description notes)
- the list of components (with their summary notes)

Description of a node

Contents

The description of a node contains:

- a "*Description*" chapter
- operations
- activity graphs

"Description" chapter

This chapter contains:

- the name of the owner
- summary and description notes
- the list of tagged values
- the stereotype
- the list of constraints
- the list of parent nodes (with their summary notes)
- the list of operations in syntactic form (with their summary notes)
- the list of roles in syntactic form (with their description notes)
- the list of attributes in syntactic form (with their description notes)

Description of an operation

Contents

The description of an operation contains:

- a "*Description*" chapter
- collaborations
- state machines
- activity graphs

"Description" chapter

This chapter contains:

- the syntax of the operation
- the redefinition of the operation
- summary and description notes
- the list of tagged values
- stereotypes
- the list of constraints
- the list of classes used (with their summary notes)
- the list of parameters in syntactic form and their tagged values (with their description notes)
- the return parameters and tagged values (with their summary notes)
- pre-conditions
- post-conditions

Generating the syntax

The syntax of an operation contains:

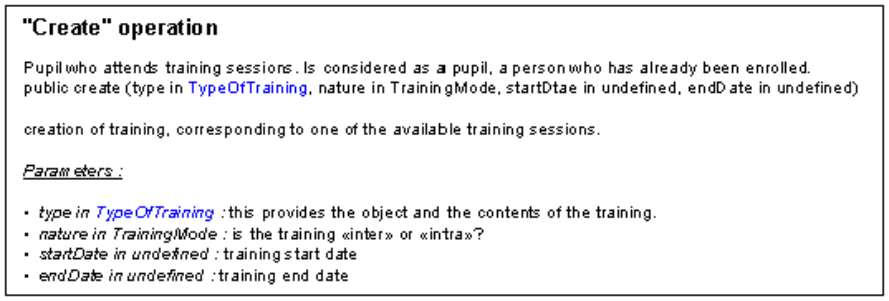
- the visibility (public, protected, private)
- the "*abstract*" keyword if the operation is abstract
- the "*final*" keyword if the operation is final
- the "*class*" keyword if the operation is "*class*"
- the name of the operation
- the syntax of the parameters
- the syntax of the return parameter

Generating a parameter's syntax

The syntax of a parameter contains:

- the parameter name
- the default value
- the passing mode (*in, out, inout*)
- the size if the parameter is a set
- the name of the class which types the parameter (if the class is a class modeled in Objecteering/UML, a hypertext link towards the file which describes this class is inserted)

Example



"Create" operation

Pupil who attends training sessions. Is considered as a pupil, a person who has already been enrolled.
public create (type in TypeOfTraining, nature in TrainingMode, startDtae in undefined, endDate in undefined)

creation of training, corresponding to one of the available training sessions.

Parameters :

- *type in TypeOfTraining :* this provides the object and the contents of the training.
- *nature in TrainingMode :* is the training «inter» or «intra»?
- *startDate in undefined :* training start date
- *endDate in undefined :* training end date

Figure 6-9. Example of generating an operation

Description of an instance

Contents

The description of an instance contains:

- description notes
- the list of tagged values
- stereotypes
- the list of constraints
- the list of attributes (with their description notes)
- the list of links (with their description notes)

Other described elements

Generation of an attribute

The description of an attribute is used during the generation of a class, an interface, a use case or of an actor. It contains:

- syntax
- tagged values
- description notes

The syntax of an attribute allows the generation of:

- the visibility (public, protected, private)
- the "*class*" keyword if the operation is "class"
- the "*const*", "*unsigned*", "*short*" keywords according to the tagged values that annotate the attribute
- the size if the attribute is a set
- the name of the class that types the attribute (if the class is a class modeled in Objecteering/UML, a hypertext link towards the file which describes this class is inserted)
- the attribute name
- the default value

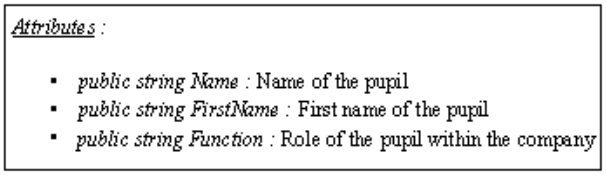


Figure 6-10. Example of generating an attribute

Partial generation

Overview

The design document template allows the partial generation of the design file. In order to do this, you simply have to select a specific document template document item in the "*Partial Generation*" tab (for further information, see chapter 4, "*Carrying out documentation generation*", of this user guide.).

These document items allow you to generate:

- the "*Overview*" chapter
- the "*Architecture*" chapter
- the "*General design*" chapter
- the "*Traceability*" chapter
- the "*Integration*" chapter
- packages and sub-systems
- nodes
- components
- classes
- interfaces
- use cases
- actors

"Architecture" document item

This is used to generate the "Architecture" chapter.

"DescriptionOfTheClasses" document item

This allows you to generate the description of a class. It does not function with interfaces.

"DescriptionOfTheComponents" document item

This allows you to generate the description of a component.

"DescriptionOfTheInterfaces" document item

This is used to generate the description of an interface. It does not function with classes which are not interfaces.

"DescriptionOfTheNodes" document item

This is used to generate the description of a node.

"DescriptionOfThePackageCompleteHierarchy" document item

This is used to generate description of all the sub-packages.

"DescriptionOfTheSubsystemCompleteHierarchy" document item

This is used to generate a description of all sub-systems.

"DescriptionOfThePackageActors" document item

This is used to generate the description of an actor.

"PackageUseCase" document item

This is used to generate the package "*Use cases*" chapter.

"GeneralDesign" document item

This is used to generate the "*General design*" chapter.

"Integration" document item

This is used to generate the "Integration" chapter.

"Overview" document item

This is used to generate the "*Overview*" chapter.

"DescriptionOfThePackageUseCases" document item

This is used to generate the description of a package use case.

"Traceability" document item

This is used to generate the "*Traceability*" chapter.

Chapter 7: Parameterization

Overview

Comment

Since documentation generation document templates ensure very precise default text markers, the text marker mechanism is automatically used. This chapter is addressed to advanced users who wish to carry out specific operations. In general, users choose a format, and the "tpf" tool carries out the processing according to the formats.

Tpf

The Tpf (Text Pre-Formatter) tool is used to process text generated by Objecteering/UML (documentation) to make the format compatible with the text processors.

Examples:

Postscript, Rtf, HTML

Objecteering/UML can therefore adapt itself to any DTP tool or text processor provided it accepts a "*marked*" format.

Processing: Illustration

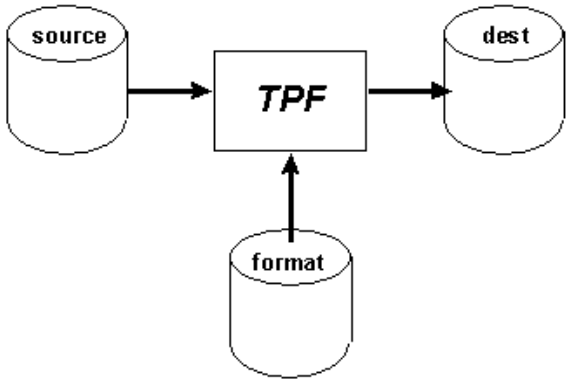


Figure 7-1. Producing end documents with the help of tpf

Processing: Description

Phase ...	Processing ...
1	Document text can be entered through dialog boxes.
2	Introducing tags in this text allows you to: - format this text - apply styles
3	A " format description " file allows you to determine the target text processor.
4	You can generate the same documentation for several types of different formats.

Command

The following example allows you to convert the document located in the "*input_file*" file to the format called "*formator_name*":

```
tpf -f output_file -tf formator_name input_file
```

Note: The result can be on screen, in a file, or used by a processor, according to the " *formator_name* " format.

Presentation macros

Role

When writing documentation, you may use or define several presentation macros (tags).

Marker macros

The marker macros provided with Objecteering/UML are presented in the following sections.

Markers defined by the user

All markers, except chapter markers, can be specifically defined.

Chapters

Dealing with chapters

Tpf provides a chapter management function (levels, automatic numbering, etc.), using the "*CHAP**" marker inserted into the text.

Example:

For a chapter numbered "6.*2.4*", the level is 2.

Note: We recommend against using this marker with a document template, since generation using document templates automatically generates markers. The use of this marker can, therefore, introduce inconsistencies in the chapter numbering of the document.

Markers

The following marker can be used:

Marker...	Description ...
\CHAP n\	Defines a chapter at level n, according to the last chapter with the corresponding level.

Example

```
\CHAP 4\ Current title 1
\CHAP 5\ Current title 2
\CHAP 3\ Current title 3
\CHAP 1\ Current title 4
```

If the previous chapter is numbered 5.2.3.1, you will obtain the result below:

```
5.2.3.2 Current title 1
5.2.3.2.1 Current title 2
5.2.4 Current title 3
6 Current title 4
```

Escape markers

Purpose

Tpf:

- interprets the described text
- replaces the tagged presentation macros and translates them

Use the escape tags if you wish the source text to remain unchanged in the final text.

Format

These surround the text which must remain unchanged with the "<*ESC*>...<*ESC*>" marker.

As a result, any text surrounded by an escape tag is not interpreted by tpf.

Example

If you wish to write a text which contains "*CHAP*\", tpf will normally:

- interpret this macro
- replace it with the chapter's style

If you want the "\CHAP\" text to remain unchanged, surround it with an escape tag:

```
<ESC> free text \CHAP\free text <ESC>
```

Result: `free text \CHAP\free text`

Default markers

Aim

Objecteering/UML supplies several predefined format descriptions, containing the same markers for the different text processors.

Style definition markers

Marker...	Description ...
\B...B\	Puts the text in bold. Example: \B example B\ -> **example**
\I...I\	Puts the text in italic. Example: \I example I\ -> *example*
\U...U\	Underlines the text. Example: \UexampleU\ -> example
\CR\	Inserts a carriage return.
\TAB\	Inserts a tabulation.
\JUST\	Justifies the text located after this marker.
\NO_ JUST\	Does not justify the text located after this marker.
\CENTER...CENTER\	Centers the surrounded text..
\HEADER...HEADER\	Defines the page headers.
\FOOTER...FOOTER\	Defines the footnotes.
\TITLEPAGE...TITLEPAGE\	Surrounds the markers that define a title's page. In this marker, sub- markers can be used. \TITLE...TITLE\: Defines a document title for the surrounded text. \SUBTITLE... \SUBTITLE... Defines a subtitle. \ TITLEHEADER... \ TITLEHEADER... Defines the title page header.
\PAGE\	Inserts a page skip.
\PAGENUM\	Inserts the number of the current page.
\DATE\	Inserts the current date.

Graph insertion markers

Two insertion markers are available:

Marker ...	Description ...
\INCG	Graphic insertion by reference, which only includes the reference of the designated graphic file (UNIX or MS-DOS access path) and not the graphic file itself. Advantage: This option generates less bulky document files. Disadvantage: The documentation is spread over several files. To insert a graphic file by reference, enter the following: \INCG file _name END_PARAM\ According to the generation format used, the suffix will automatically be added (“gif” for HTML, “eps” for Postscript,”emf” for RTF in Windows and “eps” for RTF in UNIX).
\INCP	Insertion of all kinds of graphics by reference, which only includes the reference of the designated graphic file (UNIX or MS-DOS access path) and not the graphic file itself Advantage: This option generates less bulky document files. Disadvantage: The documentation is spread over several files. To insert a graphic file by reference, enter the following: \INCP fileName.jpg END_PARAM\ The graphic file included must be a format which can be read by the final tool. For example, GIF and JPEG are two formats which in principal can be used for HTML. Using another format risks the image not appearing, unless the explorer is able to interpret another format.

Text insertion marker

The \INCT “fileName” END_PARAM\ marker is used to insert a text file into the generated document.

The format of this file must be compatible with the chosen formatter.

For an HTML document, this marker is translated in the form of a hypertext link. In this case, two parameters are necessary: the name of the file and the text which is to be used for the link.

Example

```
\Bobjecteering/UMLB\ is both a \UmodelingU\ \IeditorI\ and
\IcompilerI\.
```

Result:

Objecteering/UML is both a modeling *editor* and *compiler*.

Specific markers

Purpose

New markers can be created by adapting the "*format description*" file.

Format description file

You can define the necessary markers in the format description file. The file is located in the *$OBJING_PATH/tpf* directory.

- html.fmt for HTML
- winword.fmt for RTF
- ascii.fmt for ASCII
- groff.fmt for Postscript

All markers must have the following syntax:

MARKER_NAME: comment:position:translation

MARKER_NAME: name of the marker

comment: description of the marker

position: BEGIN_LINE or END LINE

translation: value of the marker for the target

Example

In the following example, we are going to create a marker used to generate text in italics and in bold.

Definition of markers in the <OBJING_PATH>\tpf\html.fmt file:

```
\IB:begin line in bold and italics:BEGIN_LINE:<I><B>
IB\:end line in bold and italics:END_LINE:</I></B>
```

In an Objecteering/UML note, it is now possible to use this marker.

"\IB text in italics and in bold IB\" will generate the following text: "***text in italics and in bold***"

Word document model

Overview

RTF documents generated are associated to a Word model named “*styles.dot*”. You can use your own Word model to modify all document styles generated (font, size, etc.)

Styles

Several styles can be redefined in the Word model:

- the "*Normal*" mode
- chapter levels
- chapter levels for the table of contents
- the header for all the pages (except the first page)
- the footer (except the first page)
- the title
- the sub-title
- the page number
- the bulleted list
- the numbered list
- diagram legends
- diagrams: Diagram
- method syntax: Syntax
- table header
- table text

Use

A module parameter is used to indicate the Word model which is to be associated with the generated document. This model is copied into the generation directory and renamed "*styles.dot*". During generation, the document generated is associated with a model named "*styles.dot*" and stored in the same directory as the RTF document. It is for this reason that your model is duplicated and renamed.

In this way, all elements relative to the document generated (model, images) are stored in the same directory and your document can, therefore, be used by all users, wherever it be stored.

From Word, you can also modify the model associated with the generated document. However, this link will be lost after the next generation.

Messages

Overview

All messages generated in the document (except the contents of Objecteering/UML notes) can be parameterized (chapter titles, bulleted list headers, etc.).

These messages are defined in the message file indicated in the document work product and are called by the document template. The contents of these files depend on the type of document template used.

To parameterize messages, you should simply copy the file relevant to the document template (analysis.* or design.*), modify the contents of the file, copy it into the $OBJING_PATH\GenDocModule\res directory, and indicate this file in the document work product. (<GenDocModule> is the name of the *Objecteering/Documentation* module).

In this way, you can easily generate a document in another language (Spanish, German, etc.). For example, “analysis.fr” is used to generate an analysis document in French, whilst “analysis.us” is used to generate the same analysis document in English.

For further information on messages, please refer to the "*Internationalizing messages*" in chapter 4 of the *Objecteering/Document Template Editor* user guide.

Message syntax

In the message file, messages must have the following syntax:

```
Identifier:
This is the translation of the message.
end Identifier
```

Example

You wish to rename “*Package*” “*Categories*”. To do this, you must modify the contents of the “*Package*” message:

```
Package:
"%1" categories
end Package
```

Chapter 8: Advanced document template parameterization

Advanced document template parameterization - Overview

Purpose of this chapter

Objecteering/UML documentation generation can be customized using the tools for building document templates. For more sophisticated parameterization services, the *Objecteering/UML Profile Builder* tool is used to realize J parameterization rules.

This chapter, therefore, presents the necessary additional information to parameterize a document template (generation rules, hypertext links, etc.)

For further information on how to create and use a document template, please refer to the *Objecteering/Document Template Editor* user guide.

Customizable rules

Document templates call several rules which are in fact J methods. These methods have the following nature:

- filter rules: produce a filtered set of model elements to be processed in the documentation (for example, for the public methods of a class, the filter is the method's visibility)
- generation rules: produce a description to be incorporated in the documentation
- scrolling rules: offer useful browsing through model elements, through an interface which does not require knowledge of the metamodel

UML profiles

The generation of documentation using document templates is defined on the "*default#external#Documentation*" UML profile. It is, therefore, necessary to define your document template on a UML profile which specializes this UML profile.

Furthermore, in order to use the commands defined for the *Objecteering/Documentation* module, it is necessary to make the module which contains your document template specialize this module. For further information, please refer to the "*Objecteering/UML Profile Builder*" user guide.

Document work product

Overview

In order to parameterize the documentation generator, it is useful to retrieve the values entered in the document work product that runs the generation.

This document work product can be retrieved using the getStartUpObject() J method.

```
Object : getStartUpObject() return Object
```

Attributes

The ... attribute	type ...	corresponds in the document work product to...
Author	String	the author
Date	String	the date
Formator	enumerated: Ascii, Groff, HTML, Winword	the targeted format
messageFile	String	the message file
Name	String	the name of the work product
Path	String	the generation directory
Title	String	the title
Reference	String	the reference
SubTitle	String	the subtitle
suffix	String	the generation suffix ("txt" for Ascii, "ps" for Postscript, "html" for HTML, "rtf" for Winword)
Version	String	the version

Example

```
file = getStartUpObject().messageFile;
```

Creating rules

Overview

It may be necessary to define new generation or filter rules in order to have rules applying to a specific requirement.

The generation and filter rules are in fact simple J methods referenced by document items. However, these J methods have a definition and behavior which are specific to the document template.

In order to avoid overloading the dialog boxes that reference the rules, only the J methods designed for the document template should be displayed. This explains why certain criteria allow you to filter the J methods of interest to the user.

Filter rule

To make a J method available as a filter rule on a document item, the method must have:

- a Boolean type return parameter
- the {*template*} tagged value

The methods provided are those corresponding to the criteria previously defined on the document template's UML profile and the parent UML profiles.

Creating new filter rules allows you to filter model elements according to the value of their attributes.

For example, it is possible to create document items that only generate public methods, or string type attributes, etc.

Generation rules

To make a J method available as a pre-generation, generation, or post-generation rule on a document item, the method must have:

- a string type return parameter
- the {*template}* tagged value

The methods provided are those corresponding to the criteria previously defined on the document template's UML Profile and parent UML profiles.

Creating new generation rules allows you to generate new information on the model elements according to the value of their attributes. For example, it is possible to create rules that generate the visibility of a method, the type of an attribute, etc.

Running a template

While generating the document template, the J methods used for the filter rules, and the three types of generation rules are assessed. The method's syntax does not undergo any checks.

Modifying the signature of such a method after it has been referenced by a document item would definitely risk bringing the generation to a sudden stop.

Parameterizing HTML generation

Automatic hypertext links

When generating HTML, hypertext links can be generated automatically during the generation of a model element.

The generation of hypertext links can be parameterized by redefining a J method defined on the model element.

The hypertext links defined on ...	fetch a file with the identifier ...
packages	of the package
classes	of the class
parameters	of the class which gives the parameter type
roles	of the class which is the destination of the association

The following method is used to define a hypertext link for a model element:

```
default#external#Documentation#getHyperLink return String
```

It should return the name of the file to which the link is referring to.

Note: During the generation of the hypertext link, the existence of the file is not checked. Indeed, the file can be generated after the hypertext link is generated by a document item that will be defined later in the document template.

Example of automatic hypertext links

This method returns the name of a file to which the hypertext link should point to. This name is made up of the name of the class and the "*html*" extension.

```
Class:default#external#Documentation#MyDocPov#getHyperLink
{
   return.strcat (Name, ".html");
}
```

Generated file names

In HTML generation, a document item may need to generate the information for each model element scrolled through in a specific file. The file name is customizable by redefining a J method.

The following method allows you to define the name of the file that will be generated for a model element:

```
ItemDocumentation:default#external#Documentation#getFileName
                     (element : in DescribedCRElement,
                      path : out String,
                      fileName : out String,
                      suffix : out String)
```

It is called for each model element during the generation of a document item. This document item is called as parameter.

The ... parameter	corresponding to ...	is by default ...
path	directory	the directory defined in the document work product
fileName	file name	the identifier of the model element scrolled through
suffix	suffix	out" (a file is formatted with TPF which only accepts files with the "out" extension). It renames the file with the correct suffix after the formatting

Generated file names

This method returns the name of the file that must be created by the document item for a specific model element. This name is made up of the name of the model element and the "out" extension (TPF will translate this extension adequately when the files are formatted in the desired targeted format.)

```
ItemDocumentation:default#external#Documentation#MyDocPov#ge
tFileName (element : in DescribedCRElement, path : out
string, fileName : out string, suffix : out string)
{
   path     = "";
   fileName = element.Name;
   suffix   = "out";
}
```

Generating hypertext links

The following method is a service used to generate hypertext links:

```
Object:generateHyperLink (link : in String, text : in
String) return String
```

If the content of the "link" parameter is empty or if the generation is not HTML, then the method returns the content of the "text" parameter.

In other cases, a character string with the following type is returned:

```
<A HREF="link">text</A>
```

Generating hypertext links: Example

The following method is used to generate the name of the parent of a *NameSpace* with a hypertext link towards it.

```
String NameSpace::generateOwner ()
{
String fileName;
    String result;

    if (isHtmlGeneration()) {
        result.strcat (getMulMessage (MSG_FILE, "Owner"),
                       generateHyperLink
(OwnerNameSpace.<getHyperLink(), OwnerNameSpace.<Name),
                       generateCarriageReturn());
    }
    return result;
}   // method generateOwner
```

HTML generation

The following method is used to find out whether the current generation generates in HTML format. The target format information is retrieved in the document work product that ran the generation.

```
Object:isHtmlGeneration() return boolean
```

Generating tables

Overview

J services are provided to create tables for the RTF and HTML formats.

Note: For the ASCII and Postscript formats, each column of the table is separated by a blank.

The ... J service	is used to ...
generateTableStart	define the table.
generateTableLine	define a line in the table.
generateTableEnd	end generation of the table.

Detailed description

```
String generateTableStart (in String width,
                           in String align,
                           in boolean border,
                           in String caption,
                           in String captionAlign)
```

The ... parameter	designates ...
width	the width of the table on the page. This width is expressed as a percentage (from 0 to 100).
align	the alignment of the table on the page. Possible values are "*left*", "*right*" and "*center*".
border	the possibility of adding a border to the table.
caption	the table's caption.
captionAlign	the position of the caption with regard to the table. Possible values are "*top*" (the caption is before the table) and "*bottom*" (the caption is after the table).

```
String generateTableLine (in boolean header,
                          in String [] cells,
                          in String [] columnWidth,
                          in String columnAlign)
```

The ... parameter	designates ...
header	the line used as the header of the table. In RTF, all this line's cells have a style different to that of the other lines. This style can be parameterized in the "*styles.dot*" Word model. In RTF, this line is also repeated at the top of every page, where the table is generated over several pages.
cells	cell text in the form of a set of Strings.
columnWidth	the width of the columns in the form of a set of Strings. This width is expressed as a percentage with regard to the size of the table (from 0 to 100).
columnAlign	the alignment of the column's text in the form of a set of Strings. Possible values are "*left*", "*right*", "*center*" and "*justify*".

```
String generateTableEnd()
```

This service is essential to stopping the definition of the table. If this service is not called, Microsoft Word runs the risk of not being able to open the RTF document generated. Similarly, internet browsers are not able to display the generated document correctly.

Example

The following example allows the generation of a table containing the operations of a class. This table will contain two columns:

- the name of the method
- the description of the method

To run this code, you should simply:

- create a generation method which can be referenced by a document template item (for further information, please refer to the "*Creating rules*" section in the current chapter of this user guide)
- define a document item which browses the classes of a package, and reference the previous method as a pre-generation method
- enter the following code for the pre-generation method

```
String   result;
String[] cells;
String[] columnWidth;
String[] columnAlign;

// Definition of column width
columnWidth.addElement ("25");
columnWidth.addElement ("75");

// Definition of column alignment
columnAlign.addElement ("left");
columnAlign.addElement ("left");
```

```
if (PartOperation.size() != 0) {
    result.strcat (generateTableStart
                   ("80", "center", true, "", ""));
    // Definition of table header
    cells.addElement ("Name");
    cells.addElement ("Description");

    result.strcat (generateTableLine (true, cells,
                   columnWidth, columnAlign));

    PartOperation {
        // Generation of a line per operation of the
           class
        String description = «»;
        cells.clear();
        cells.addElement (Name);

        // Retrieval of "description" note contents
        DescriptorNote.<select (ModelNoteType.Name ==
                                "description") {
            description = description + Content;
        }

        cells.addElement (description);
        result.strcat (generateTableLine (false, cells,
columnWidth, columnAlign));
    }
    result.strcat (generateTableEnd());
}

return result;
```

The result is shown in Figure 8-1 below.

Dispenser operation

Name	Description
store	This method will add to the list : "contains" a "Food" type element.This operation can only be run by a person that has been identified by the "identification ()" method..
takeFromStock	Recovers the element with the Food type corresponding to the type of food asked for.If the type of food asked for is not present in the stocks , it returns NULL.
Choose	Offers products available in its stocks and waits for the user's choice..
accept	This method frees a product kept in the dispenser.
create	This constructor creates a new dispenser and gives it a key that will be necessarily for filling it at a later date.
cancellation	
identification	Asks the supplier for a key and checks that the entered key is correct (corresponding to the dispenser's key). Returns TRUE if the key is correct andFALSE otherwise.

Figure 8-1. The newly-created table

Scrolling through the metamodel

Defining the scrolling methods

To make a J method available as a scrolling method on a document item, the method must have:

- A set type return parameter, whatever the type of the set (*Class[]*, *Attribute[]*, etc.)
- the {*template*} tagged value

The methods provided are those corresponding to the criteria previously defined on the document template's UML profile and parent UML profiles.

The creation of new scrolling methods allows a document item to only scroll through a set of model elements. For example, it is possible to create document items that only scroll through public methods, or String type attributes, etc.

It is necessary to indicate that it is possible to scroll through a set of model elements, either by creating a new scrolling method, or by using a filter method.

Translating scrolling methods

The metamodel scrolling methods available in the "*Described association*" scrolling list of a document item can be translated. The complete name of the J method (metaclass name, UML profile name, method name) then no longer appears. This translation allows users who may not be familiar with the metamodel to easily get to know the model elements processed by the document item.

Messages must be defined in the info.us file stored in the *$OBJING_PATH/res directory*.

The identifier of this message is:

```
<Class>_<PointOfView>_<Name>
```

Note: The '#' character defined in the UML profile name must be replaced by the '_' character.

Example:

```
Class_default_external_Documentation_MyPov_publicAttrs:
Public attributes
end Class_default_external_Documentation_MyPov_publicAttrs
```

Methods for scrolling through the metamodel

On the ... metaclass	the journey named ...	scrolls through the ... association
ActivityGraph	Action states	TopStatesActionState
ActivityGraph	Object flow states	TopStatesObjectFlowState
ActivityGraph	Partitions	SwimlanePartition
ActivityGraph	Sub-activity states	TopStatesSubActivityState
Actor	Use case which communicates	CommunicationLinkCommunication .<TransmitterUseCase and CooperationCommunication .<CooperationUseCase
Actor	Actors which communicate	CooperationCommunication .<CooperationActor and CommunicationLinkCommunication .<TransmitterActor
Actor	Parent actors	ParentGeneralization.<SuperTypeActor
Actor	Attributes	PartAttribute
Actor	Operations	PartOperation
Association	Associated class	LinkToClassClassAssociation .<ClassPartClass
Association	Roles	ConnectionAssociationEnd
AssociationEnd	Association	RelatedAssociation
AssociationEnd	Component class of the role	OwnerClass
AssociationEnd	Qualifiers	QualifierAttribute
Attribute	Class which types an attribute	TypeGeneralClass

On the ... metaclass	the journey named ...	scrolls through the ... association
Class	Attributes	PartAttribute
Class	State machines	BehaviorStateMachine.<select(ClassOf.Name == "StateMachine")
Class	Use cases	OwnedElementUseCase
Class	Component classes	OwnedElementClass
Class	Parent classes	ParentGeneralization.<SuperTypeClass
Class	Used classes	DestinationUse.<UsedClass
Class	Collaborations	ExampleCollaboration
Class	Diagrams	productDiagram
Class	Enumerations	OwnedElementEnumeration
Class	Sent data flows	SentDataFlow
Class	Received data flows	ReceivedDataFlow
Class	Instances	DeclaredInstance .<select (ClassOf.Name == "Instance")
Class	Implemented interfaces	RealizedRealization.<ImplementedClass
Class	Operations	PartOperation
Class	Navigable roles	PartAssociationEnd.<select (IsNavigable)
Class	Non-navigable roles	PartAssociationEnd .<select (IsNavigable == false)
Class	Types	OwnedElementDataType
ClassifierRole	Represented instance	RepresentedInstance
ClassifierRole	Messages sent	SentSequenceMessage
ClassifierRole	Messages received	ReceivedSequenceMessage
ClassifierRole	Type of instance	BaseGeneralClass

On the ... metaclass	the journey named ...	scrolls through the ... association
Collaboration	Diagrams	productDiagram
Collaboration	Roles	ComponentClassifierRole
Collaboration	Sequence messages	na
Collaboration	Collaboration messages	na
Component	Attributes	PartAttribute
Component	Components	OwnedElementComponent
Component	Implemented elements	OriginComponentModelElement and RealizedRealization.<ImplementedClass
Component	Elements used	DestinationUse.<UsedNameSpace
Component	Data flows sent	ReceivedDataFlow
Component	Data flows received	SentDataFlow
Component	Operations	PartOperation
DataType	Roles	PartAssociationEnd .<select (IsNavigable)
DataType	Attributes	PartAttribute
DataType	Operations	PartOperation
DataType	Parent types	ParentGeneralization .<SuperTypeDataType
DataType	Types used	DestinationUse.<UsedDataType
Enumeration	Literal values	ValueEnumerationLiteral
Feature	Support class of the member	OwnerClassifier
Instance	Attributes	SlotAttributeLink
Instance	Class which types the instance	BaseClassifier
Instance	Links	ConnectionLinkEnd

On the ... metaclass	the journey named ...	scrolls through the ... association
Message	Condition	GuardCondition
Message	Operation	InvokedOperation
ModelElement	Notes	DescriptorNote
ModelElement	Products	productMpGenProduct
ModelElement	Tagged values	TagTaggedValue
NameSpace	activity graphs	BehaviorActivityGraph
Node	Roles	PartAssociationEnd.<select (IsNavigable)
Node	Attributes	PartAttribute
Node	Parent nodes	ParentGeneralization.<SuperTypeNode
Node	Operations	PartOperation
Note	Tagged values	AnnotationTaggedValue
Note	Type	ModelNoteType
Operation	State machine	BehaviorStateMachine.<select(ClassOf.Name == "StateMachine")
Operation	Classes used	UsedClass
Operation	Collaborations	ExampleCollaboration
Operation	Redefined operation	RedefinesOperation
Operation	Parameters	IOParameter
Operation	Return parameter	ReturnParameter
Operation	Signals	RaisedExceptionSignal
Operation	activity graphs	BehaviorActivityGraph
Package	Actors	OwnedElementActor
Package	Referenced actors	ReferencedActor
Package	State machines	BehaviorStateMachine.<select(ClassOf.Name == "StateMachine")
Package	Use cases	OwnedElementUseCase
Package	Referenced use cases	ReferencedUseCase

On the ... metaclass	the journey named ...	scrolls through the ... association
Package	Classes	OwnedElementClass
Package	Referenced classes	ReferencedClass
Package	Collaborations	ExampleCollaboration
Package	Components	OwnedElementComponent
Package	Diagrams	productDiagram
Package	Elements used	DestinationUse.<UsedNameSpace
Package	Elements referenced	ReferencedNameSpace
Package	Enumerations	OwnedElementEnumeration
Package	Sent data flows	SentDataFlow
Package	Received data flows	ReceivedDataFlow
Package	Instances	DeclaredInstance .<select (ClassOf.Name == "Instance")
Package	Node instances	DeclaredNodeInstance
Package	Component instances	DeclaredComponentInstance
Package	Nodes	OwnedElementNode
Package	Packages	OwnedElementPackage
Package	Package complete hierarchy	OwnedElementPackage recursively
Package	Packages parents	ParentGeneralization.<SuperTypePackage
Package	Referenced packages	ReferencedPackage
Package	Referenced components	ReferencedComponent
Package	Referenced nodes	ReferencedNode
Package	Packages used	DestinationUse.<UsedPackage

On the ... metaclass	the journey named ...	scrolls through the ... association
Package	Types	OwnedElementDataType
Parameter	Class which types the parameter	TypeGeneralClass
Partition	activity graphs	ContentsStateVertex
SequenceMessage	Instance which received the message	ReceiverClassifierRole
SequenceMessage	Instance which sends the message	SenderClassifierOccurence
SequenceMessage	Activated messages	ActivatedSequenceMessage
SequenceMessage	Previous messages	PredecessorSequenceMessage
State	Parent state	ParentState
State	Events	DeferredEvent
State	Sub-states	
State	Transitions which reach the state	IncomingTransition
State	Transitions which start from the state	OutGoingTransition
StateMachine	Diagrams	productDiagram
StateMachine	Root state	TopState
StateMachine	States	na
StateMachine	Transitions	na
TaggedValue	Parameters	ActualTagParameter
TaggedValue	Type	DefinitionTagType
Transition	Trigger condition	GuardCondition
Transition	Condition obtained after triggering	na

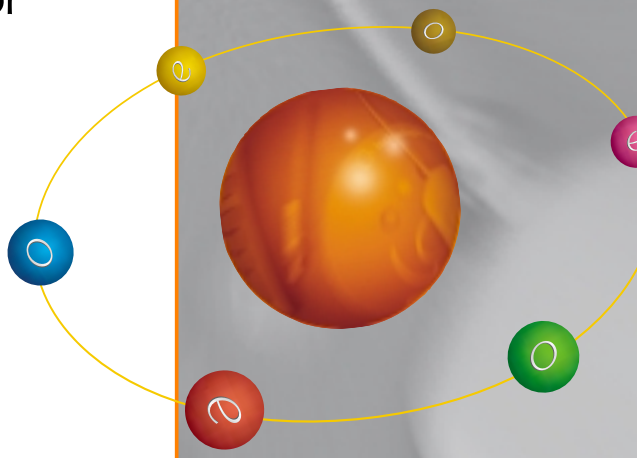
On the ... metaclass	the journey named ...	scrolls through the ... association
Transition	Initial state	SourceState
Transition	Final state	TargetState
Transition	Trigger event	TriggerEvent
Transition	Activated operation	ProcessedOperation
Transition	Signal sent	EffectsSignal
UseCase	Actors which communicate	CommunicationLinkCommunication .<TransmitterActor and CooperationCommunication .<CooperationActor
UseCase	Attributes	PartAttribute
UseCase	State machines	BehaviorStateMachine.<select(ClassOf.Name == "StateMachine")
UseCase	Parent use cases	ParentGeneralization .<SuperTypeUseCase
UseCase	Included use cases	na
UseCase	Extended use cases	na
UseCase	Collaborations	ExampleCollaboration
UseCase	Diagrams	productDiagram
UseCase	Operations	PartOperation

Index

Objecteering/UML

Objecteering/Document Template Editor
User Guide

Version 5.2.2



Objecteering
Software

www.objecteering.com

Taking object development one step further

Contents

Chapter 1: Overview

Overview of the Objecteering/Document Template Editor module

Introduction

Welcome to the *Objecteering/Document Template Editor* user guide!

The *Objecteering/Document Template Editor* edits document templates, which are used to obtain homogeneous documentation, automatically generated from a UML model built in Objecteering/UML. If you wish to generate a document (analysis, technical design, test, quality control, etc.) with a definite structure, this tool allows you to produce the document template which will generate the required document. The *Objecteering/Document Template Editor* is designed for process and quality engineers, who wish to define and drive the structure of typical documents.

A document template is a sort of "*generic*" table of contents that is highly detailed. Like a table of contents, it has a tree hierarchy.

The difficult task of designing a document template

The *Objecteering/Document Template Editor* tool is essentially based on an explorer (Figure 1.1).

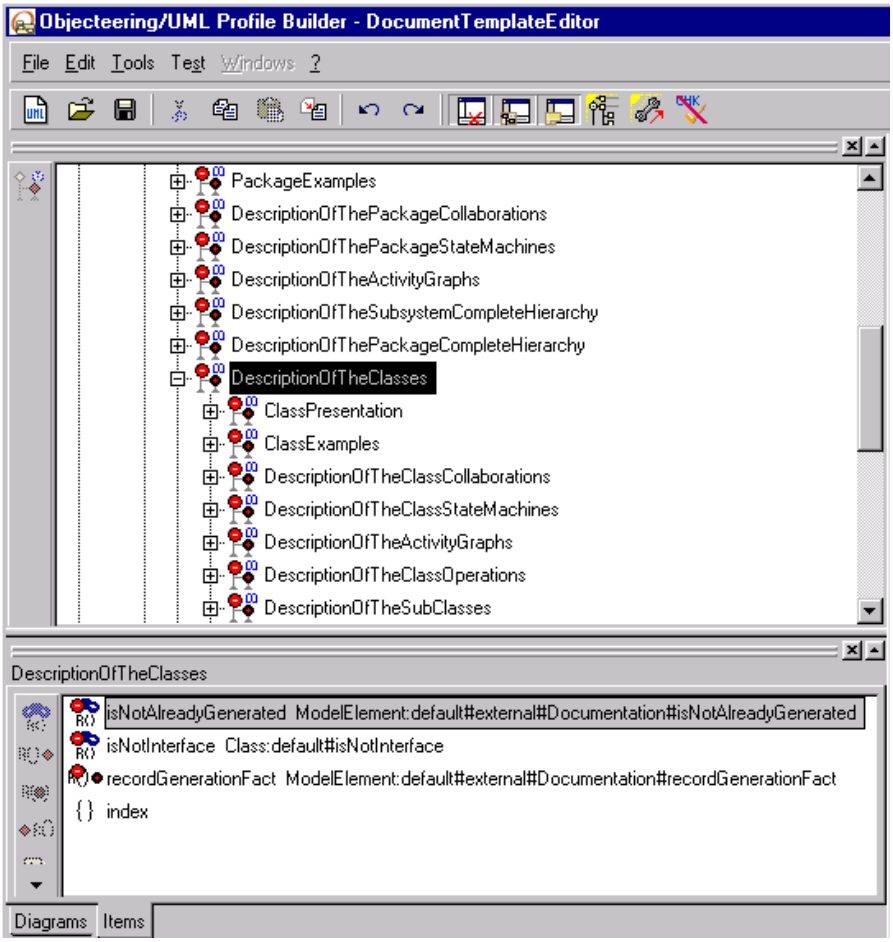


Figure 1-1. The *Document Template Editor* explorer

The difficulty consists of defining a typical tree hierarchy, adapted to any model variation: not too wordy, but providing, in a clearly laid out manner, information that is of interest to those reading the documentation.

To obtain the required document, you must already have an idea of the structure. You can then proceed by improving its form.

Two customization levels

The standard user of the tool is a "*quality*" man, interested in the content of the documentation. He is not necessarily a "*technical*" person, but is familiar with the models and the Objecteering/UML tool. The *Objecteering/Document Template Editor* can satisfy a large number of requirements.

More sophisticated customization of the tool can be achieved using the *Objecteering/UML Profile Builder* module, which allows new document rules to be created in the J language, or new note types to be defined, and so on. More technical knowledge (*Objecteering/Metamodel*, *Objecteering/J Language*) is required if you wish to use these services.

Glossary

Document template: Structure defining a general documentation form. Each piece of documentation generated from a specific document template respects its structure.

Model element: Element of the user's model that is defined in a UML modeling project (packages, operation, etc.)

Document item: Description unit within a document template. It can represent a chapter, a graph, the description of a model element, etc. A document is a group of document items, which are structured hierarchically.

Rule: Processing linked to a document item. A rule can act as a filter, or can generate the content of a zone of the document. The *Objecteering/UML Profile Builder* module allows new generation and filter rules to be created.

Chapter 2: First Steps

First Steps - Overview

Purpose of this example

The example provided in these first steps illustrates a document template that will generate essential information on a class.

First Steps procedure

During the First Steps, a document template will be created, as shown in Figure 2-1.

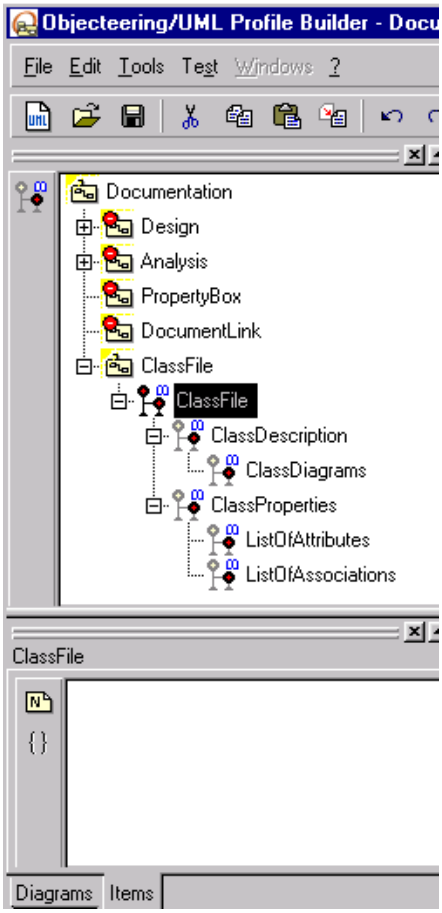


Figure 2-1. "*ClassFile*" document template

Structure of the generated document

The document template, thus defined, will allow us to generate the following document for a class, in HTMLformat (as shown in Figure 2-2).

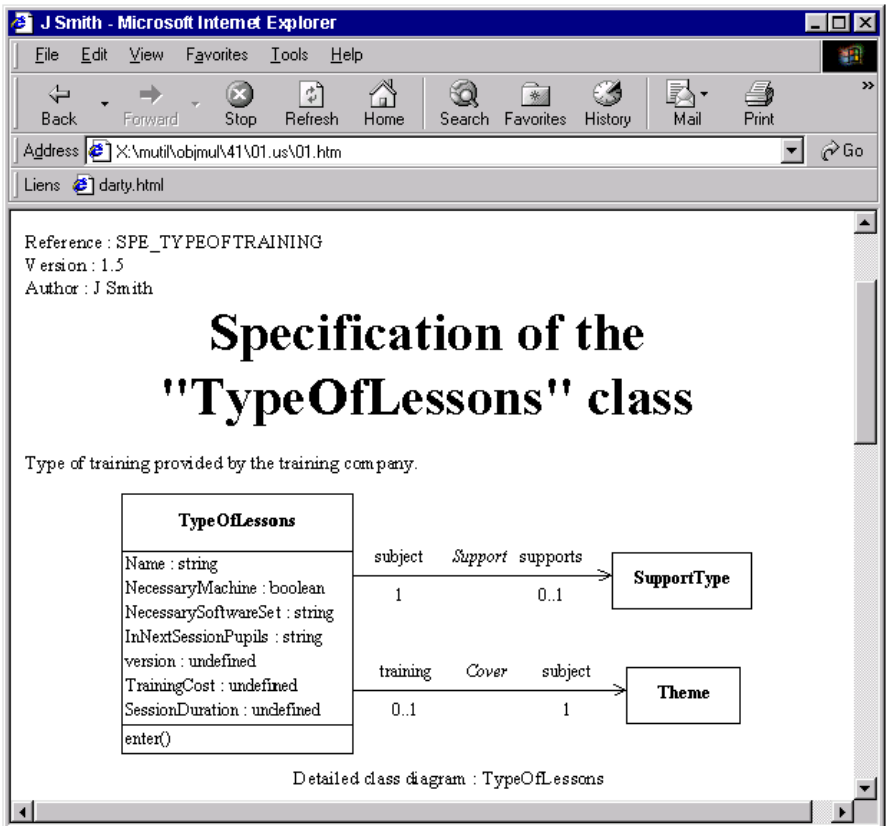


Figure 2-2. Generated document

Creating the "ClassFile" document template

Introduction

Before starting this example, we recommend that you first carry out the general Objecteering *First Steps*.

The *First Steps* of the *Objecteering/Documentation* module must be installed on your machine.

Launch Objecteering and create a document template named "*ClassFile*" (as shown in Figure 2-3).

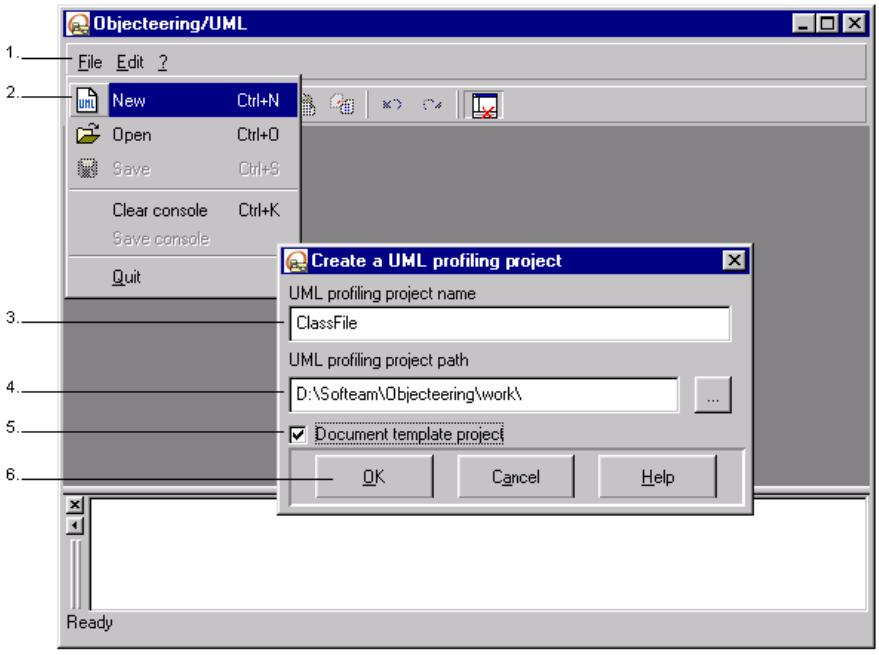


Figure 2-3. Creating the "*ClassFile*" document template

Steps:

1. Launch *Objecteering/UML Profile Builder*.
2. Click on the "*File/New*" menu. The "*Create a UML profiling project*" window then appears.
3. In the "*UML profiling project name*" field, enter "*ClassFile*".
4. In the "*UML profiling project path*" field, enter the path of the directory where the new document template is to be created. You may also use the  icon to open a file browser through which you can select your document template path.
5. Check the "*Document template project*" tickbox.
6. Confirm.

Modifying the document template

Edit in modification mode the "*ClassFile*" document template (Figure 2-4). The document template must describe the "*Class*" metaclass and must not generate a table of contents.

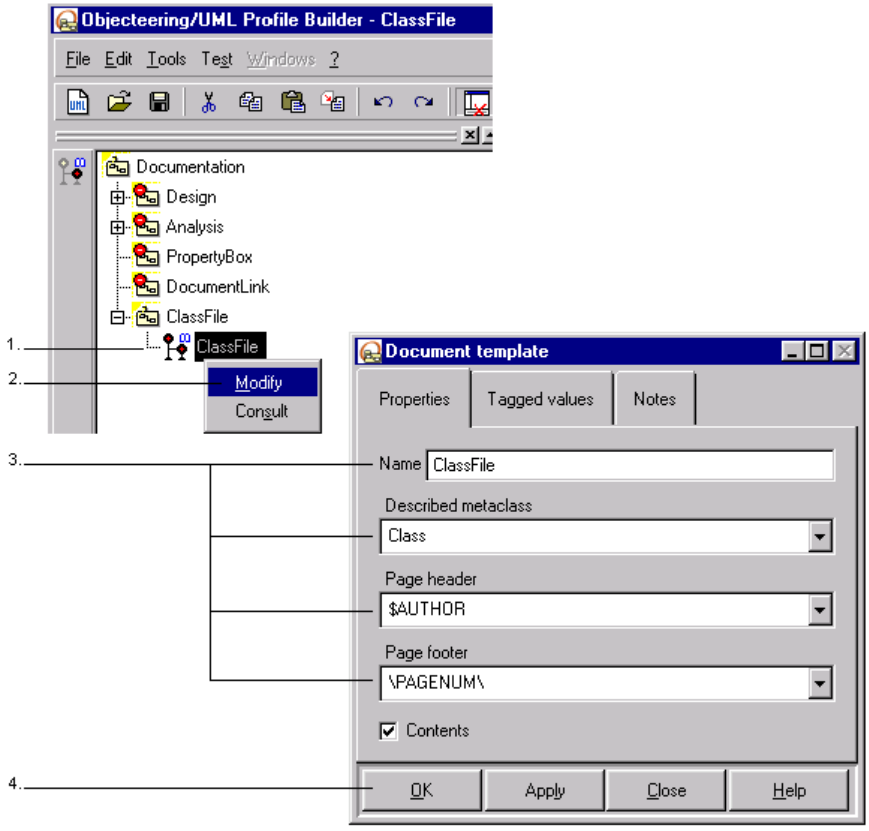


Figure 2-4. Modifying the "*ClassFile*" document template

Steps:

1 - Expand the "*ClassFile*" UML profile and select the *"ClassFile*" document template.

2 - Right-click on the "*ClassFile*" document template and run the "*Modify*" command, or double-click on "*ClassFile*".

3 - Fill in the the text fields.

4 - Confirm.

Creating the "Class description" document item

Overview

This document item allows you to group together all the document items that will generate the final document.

It allows you to generate notes defined on the class, and to launch the generation of the "*ClassDiagrams*" and "*ClassProperties*" document item.

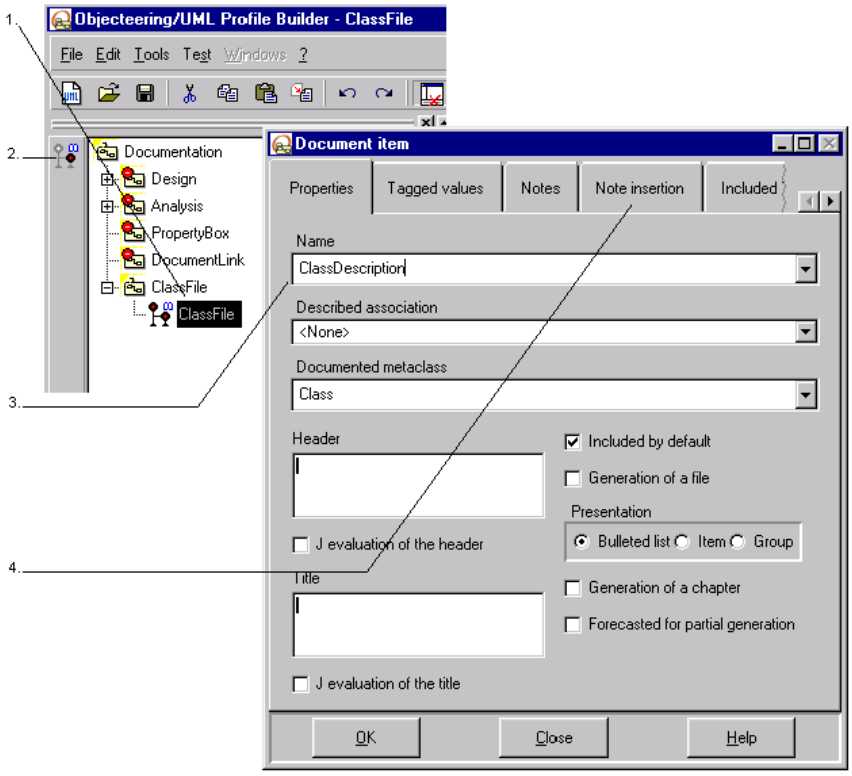


Figure 2-5. Creating the "*ClassDescription* " document item

Steps:

1 - Select the "*ClassFile*" document template.

2 - Click on the  "*Create a document item*" icon.

3 - Enter the name of the document item.

4 - Click on the "*Note insertion*" tab and proceed with the next step (inserting notes).

Inserting notes

To automatically generate "*summary*" and "*description*" notes defined on a class, you simply have to select these notes in the "*Note insertion*" tab

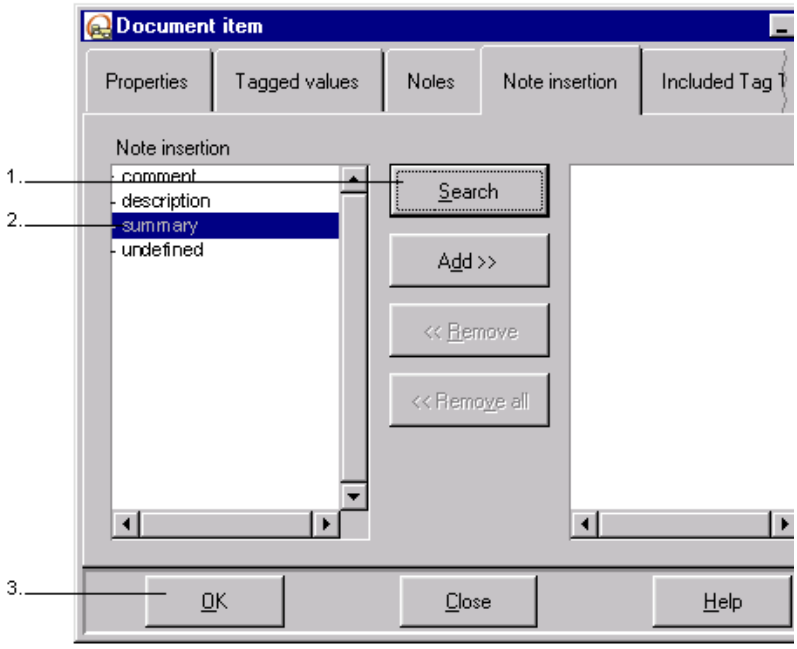


Figure 2-6. Inserting text for the "*Class description* " document item

Steps:

1 - Click on the "*Search*" button.

2 - Select the "summary" and "description" types and click on "*Add*".

3 - Confirm.

Creating the "Class Diagrams" document item

Overview

This document item allows the generation of all the class diagrams. It has to go through all the diagrams that have been defined for this class.

A key is automatically generated in the diagram which gives the diagram type and its name.

"*Description*" type notes are generated after each diagram.

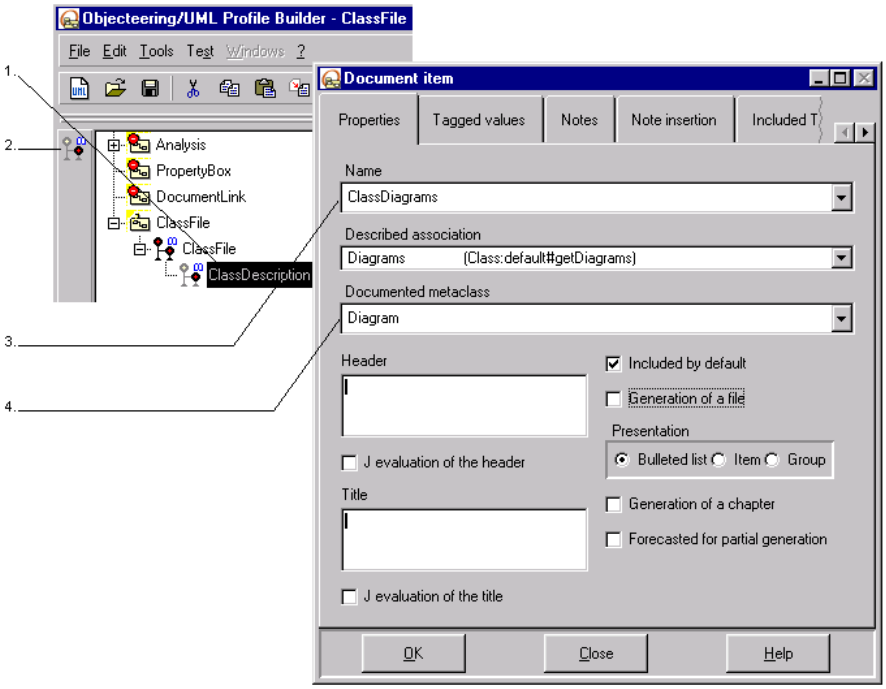


Figure 2-7. Creating the "*ClassDiagrams*" document item

Steps:

1 - Select the "*ClassDescription*" document item.

2 - Click on the  "*Create a document item*" icon.

3 - Enter the "*ClassDiagrams*" name.

4 - Enter the relation described as "*Diagrams*" in the combobox.

Note: The "*Documented metaclass*" field is automatically updated (during validation). For further information on this field, please refer to the "*Document item*" section of chapter 3 of this user guide.

Inserting notes

To automatically generate "*description*" notes defined on the class diagram, simply select "*description*" in the "Note insertion" tab.

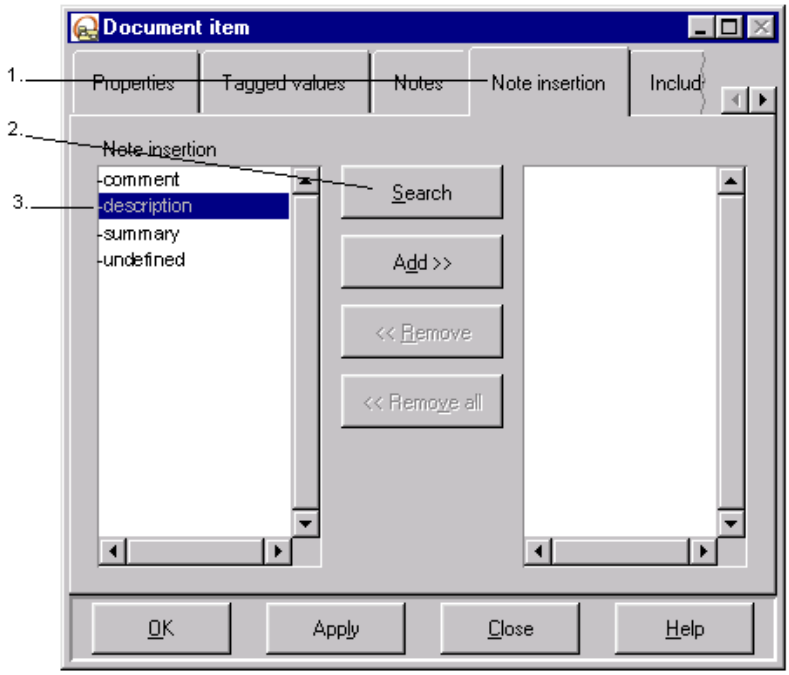


Figure 2-8. "Note Insertion" tab for the "*ClassDiagrams*" document item

Steps:

1 - Select the "*Note insertion*" tab for the document item.

2 - Click on the "*Search*" button.

3 - Select the "*description*" type and click on "*Add*".

4 - Proceed with the next step ("Excluded Tag Type").

Exclusion

To prevent a diagram annotated with the {*noanalysis*} tagged value from being generated, you must select it in the "*Excluded Tag Type*" tab.

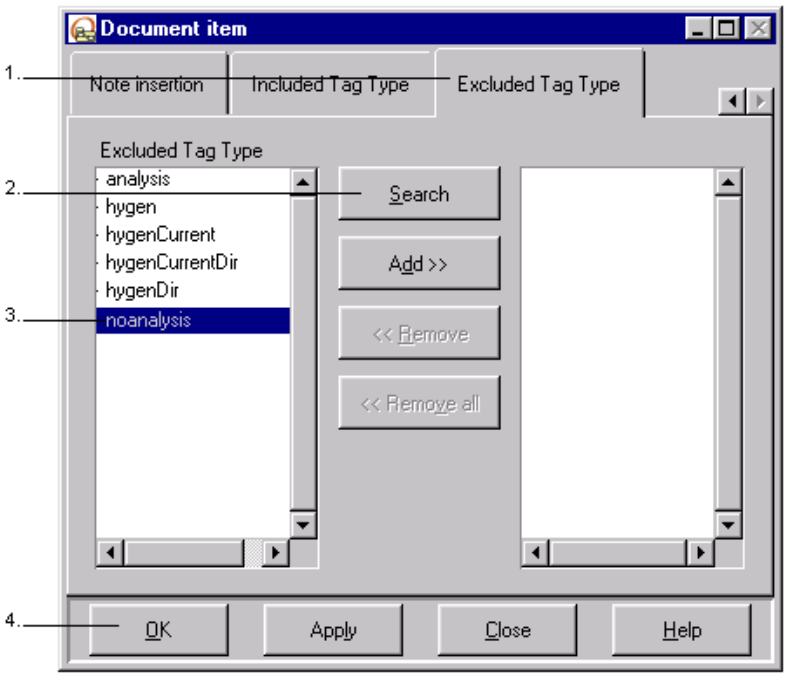


Figure 2-9. "Excluded Tag Type" tab for the "*ClassDiagrams*" document item

Steps:

1 - Select the "*Excluded Tag Type*" tab.

2 - Click on the "*Search*" button.

3 - Select the {*noanalysis*} tagged value and click on "*Add*".

4 - Confirm.

Creating the filter rule

Several types of diagram can exist on a class. Given that only class diagrams are required, the "*isClassDiagram*" filter rule, which allows a document item only to go through the class diagrams, must be used.

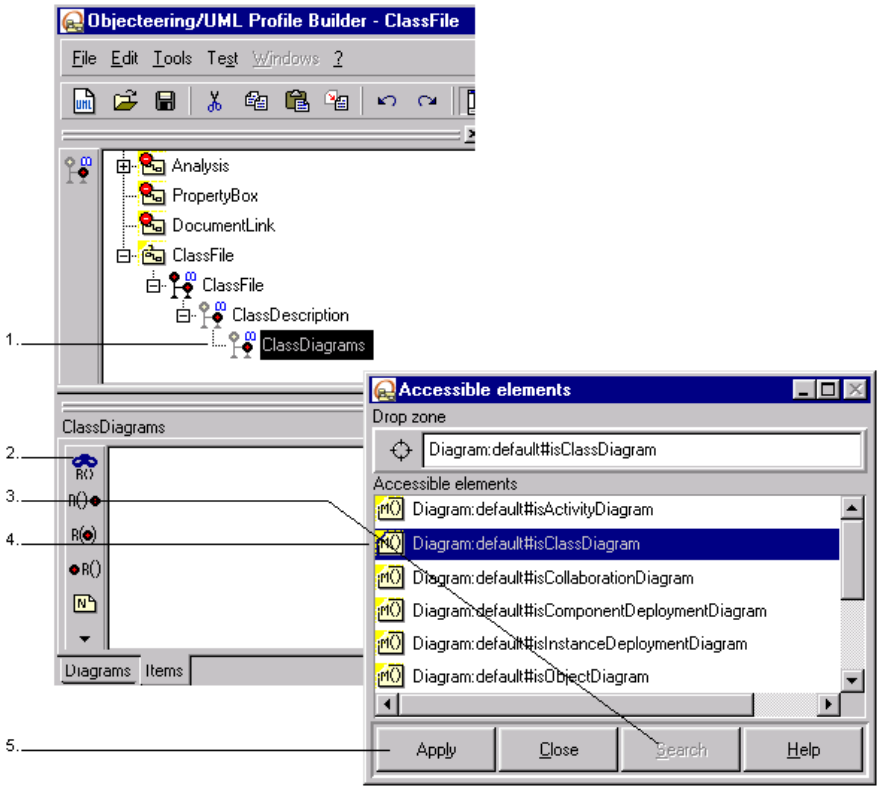


Figure 2-10. Creating the filter rule for class diagrams.

Steps:

1 - Select the "*ClassDiagrams*" document item in the explorer.

2 - Click on the  "*Reference a J method for the filtering*" icon in the "*Items*" tab of the properties editor.

3 - Click on the "*Search*" button.

4 - Select the "*isClassDiagram*" rule.

5 - Confirm.

Creating a diagram generation rule

The "*includeDiagram*" rule is used to generate the current diagram. Since we wish to generate the diagram's description after it, the inclusion rule of the diagram has to be a pre-generation rule. This rule allows the generation of the file containing the image, and the insertion of the file reference and the key in the document.

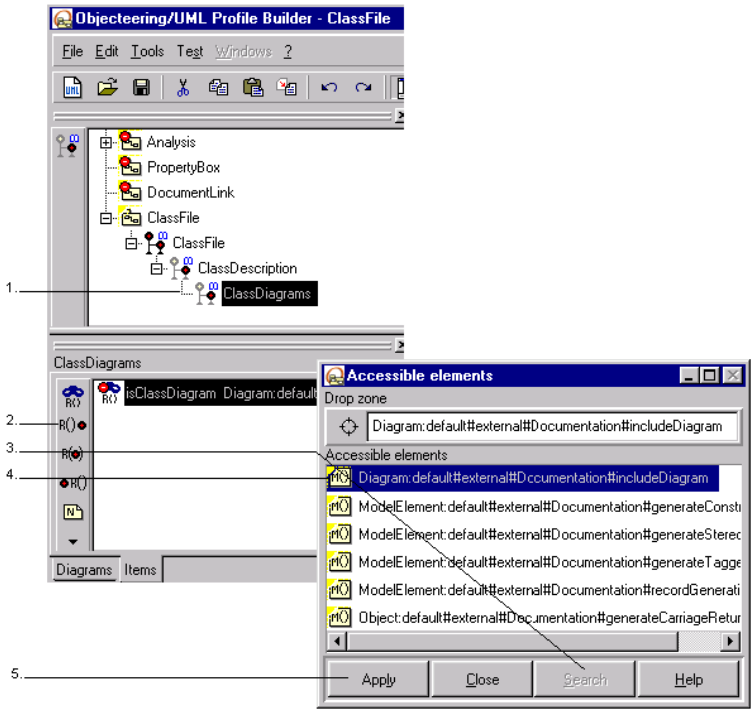


Figure 2-11. Creating the diagram generation rule.

Steps:

1 - Select the "*ClassDiagram*" document item.

2 - Click on the  "*Reference a J method for pre-generation*" icon in the "*Items*" tab of the properties editor.

3 - Click on the "*Search*" button.

4 - Select the "*includeDiagram*" rule.

5 - Confirm.

Creating the "Class properties" document item

Overview

This document item is used to group together the document items that will generate the class' attributes and associations by launching the "*ListOfAttributes*" and "*ListOfAssociations*" document items.

It contains two document sub-items, the first for generating attributes and the second for generating associations.

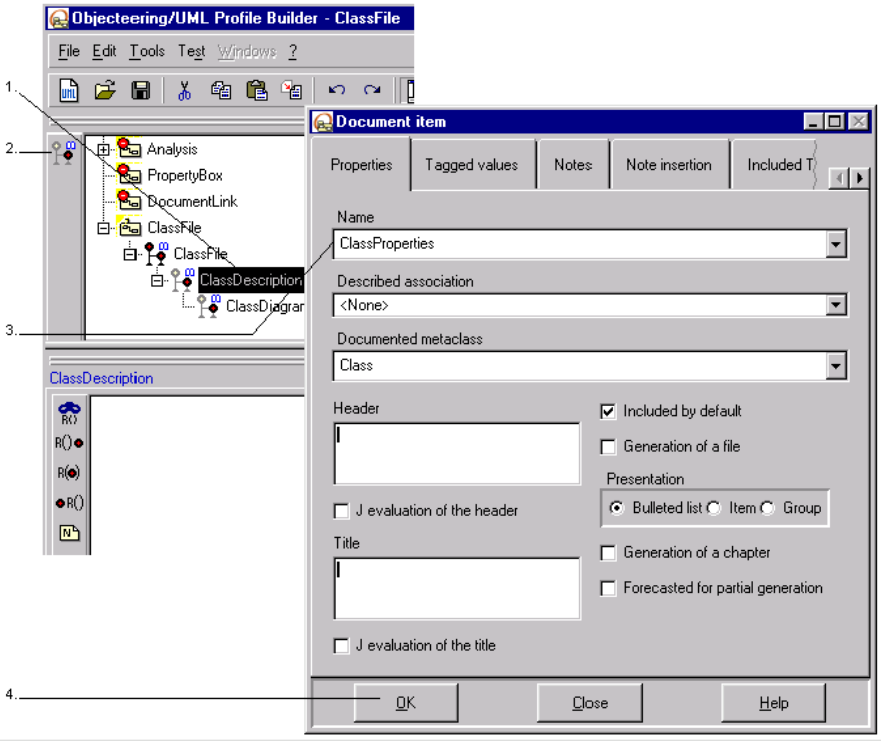


Figure 2-12. Creating the "*ClassProperties* " document item

Steps:

1 - Select the "*ClassDescription*" document item.

2 - Click on the  "*Create a document item*" icon.

3 - Enter the "*ClassProperties*" name.

4 - Confirm.

Creating the "List of attributes" document item

Overview

This document item generates class attributes in the form of a bulleted list. Before they are generated, a header message is inserted. "*Description*" notes are then inserted for each attribute.

The title of each attribute will be its name.

Note: The "*J Evaluation of the title*" box must be checked, since the title contains the "Name" value. "Name" is an attribute for the current described element which is to be evaluated during the generation.

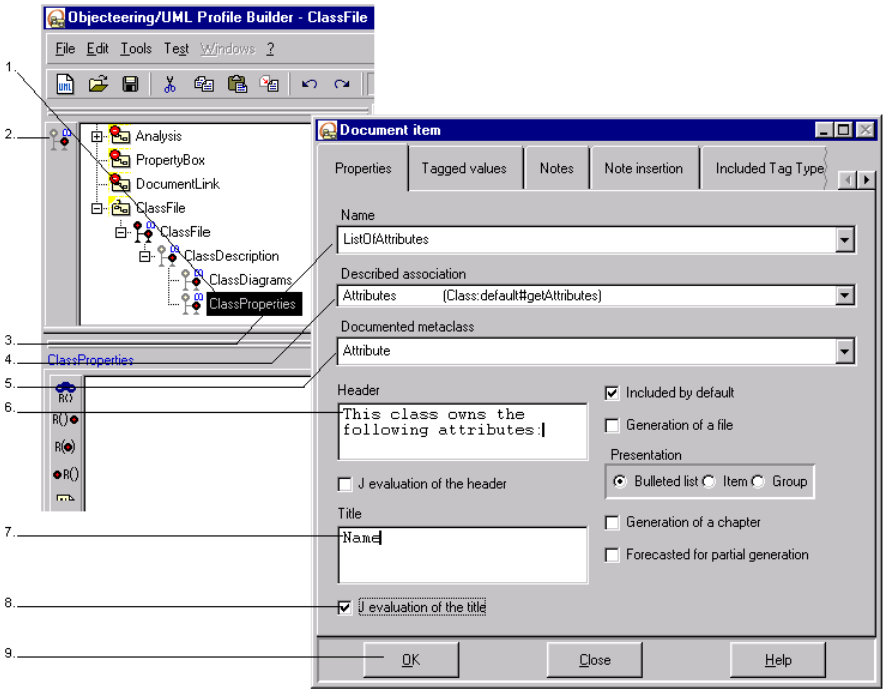


Figure 2-13. Creating the "*ListOfAttributes*" document item

Steps:

1 - Select the "*ClassProperties*" document item.

2 - Click on the  "*Create a document item*" icon.

3 - Enter the "*ListOfAttributes*" name.

4 - Enter the *"Attributes*" described association.

5 - Select "*Attribute*".

6 - Enter the "*The class owns the attributes:*" header.

7 - Enter the "*Name*" title.

8 - Check the "*J evaluation of the title*" box.

9 - Confirm.

Inserting texts

To automatically generate "*description*" notes defined on the attributes, simply select the "*description*" note in the "*Note insertion*" tab.

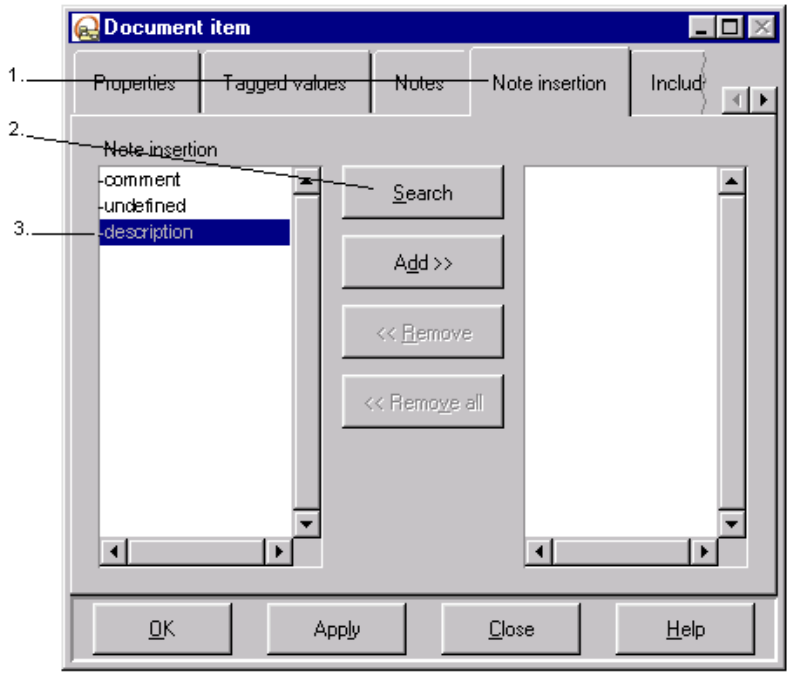


Figure 2-14. "*Note Insertion*" tab for the "*ListOfAttributes*" document item

Steps:

1 - Select the "*Note Insertion*" tab.

2 - Click on the "*Search*" button.

3 - Select the "*description*" type and click on "*Add*".

4 - Proceed with the next step ("*Excluded Tag Type*").

Exclusion

To prevent an attribute annotated with the {*noanalysis*} tagged value from being generated, you must select it in the "*Excluded Tag Type*" tab.

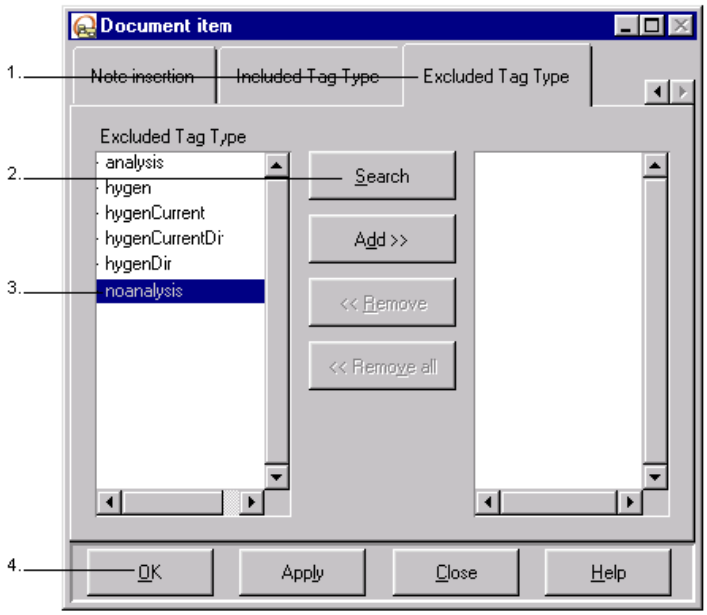


Figure 2-15. "*Excluded Tag Type*" tab of the "*ListOfAttributes*" document item

1 - Select the "*Excluded Tag Type*" tab.

2 - Click on the "*Search"* button.

3 - Select the {*noanalysis*} tagged value and click on "*Add".*

4 - Confirm.

Creating the "List of Associations" document item

Overview

This document item generates class associations in the form of a bulleted list. Before generating them, a header message is inserted, and the "*description*" notes are then inserted for each association.

The title of every association will be the role name.

Note: The "*J Evaluation of the title*" box must be checked, since the title contains an attribute belonging to the "*Link*" metaclass.

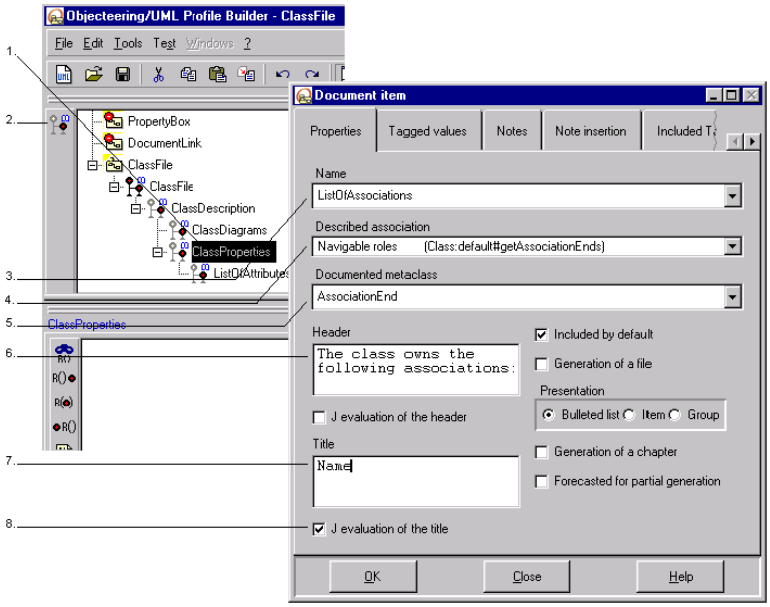


Figure 2-16. Creating the "*ListOfAssociations*" document item

Steps:

1 - Select the "*ClassProperties*" document item.

2 - Click on the  "*Create a document item*" icon.

3 - Enter the "*ListOfAssociations*" name.

4 - Enter the *"Navigable roles* " described association.

5 - Select "*AssociationEnd*".

6 - Enter the "*The class owns the associations:*" header.

7 - Enter the "*Name*" title.

8 - Check the "*J evaluation of the title*" box.

9 - Proceed with the next step ("*Inserting notes*").

Inserting notes

To generate "*description*" notes automatically defined on the roles, you simply have to select the "*description*" note in the "*Note insertion*" tab.

Note: It is also possible to create "*description*" type texts on associations, but they will not be generated by this document item.

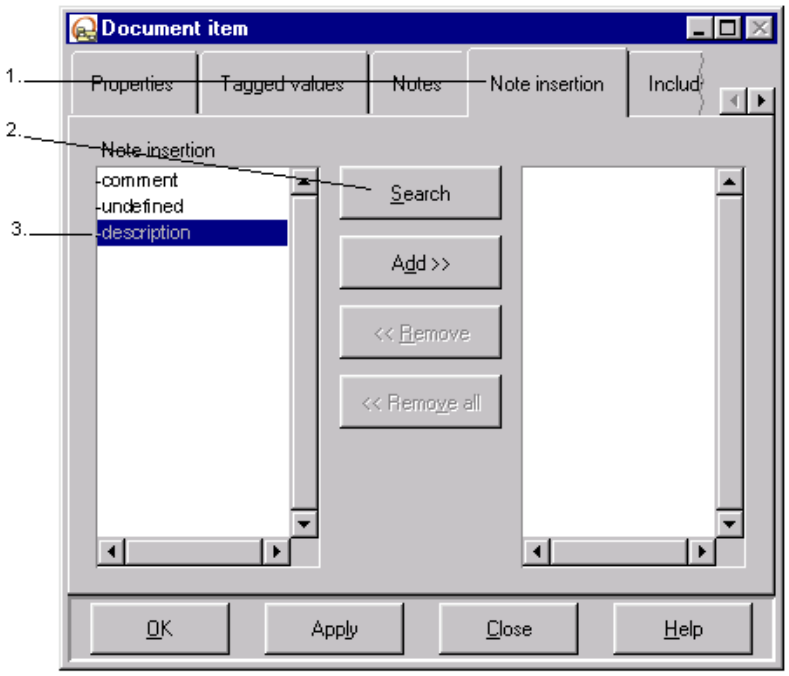


Figure 2-17. "*Note Insertion*" tab for the "*ListOfAssociations*" document item

Steps:

1 - Select the "*Note insertion*" tab.

2 - Click on the "*Search*" button.

3 - Select the "*description*" type and click on "*Add*".

4 - Proceed with the next step ("*Exclusion*").

Exclusion

To prevent a diagram annotated with the {*noanalysis*} tagged value from being generated, it must be selected in the "*Excluded Tag Type*" tab.

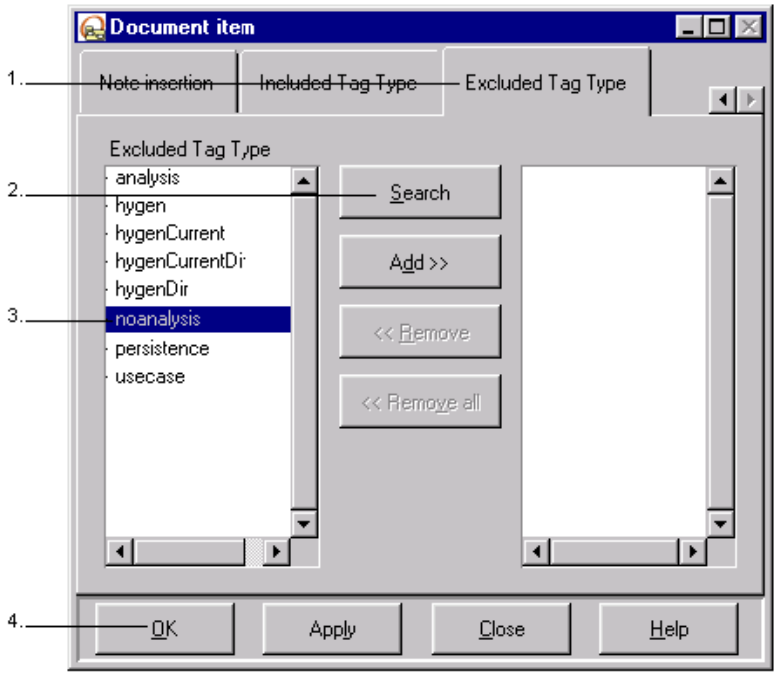


Figure 2-18. "*Excluded Tag Type*" tab of the "*ListOfAssociations*" document item

Steps:

1 - Select the "*Excluded Tag Type*" tab.

2 - Click on the "*Search"* button.

3 - Select the {*noanalysis*} tagged value and click on "*Add"*.

4 - Confirm.

Testing the document template

Introduction

The "*ClassFile*" document template has now been created. It is possible to test it in real time with a test project.

When your document template has been specified and is ready to be distributed, you can install it in any other UML modeling project.

Selecting a test project

In your base, a predefined UML modeling project must exist. If you have followed our recommendations, the "*TestProject*" project created during the general first steps is available.

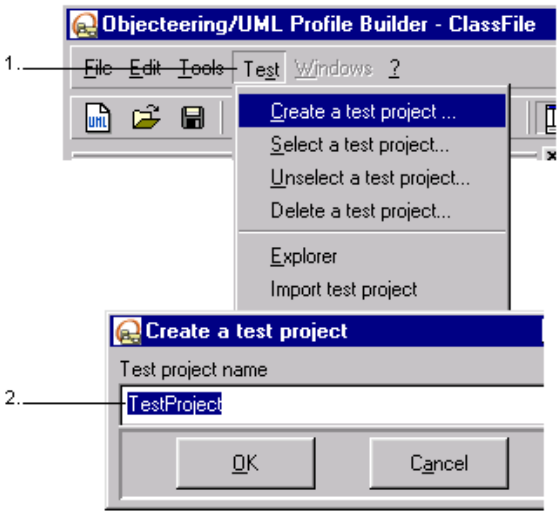


Figure 2-19. Selecting the "*TestProject*" test project

Steps:

1 - Select the "*Test/Create a test project*" menu options.

2 - Create "*TestProject*" test project and confirm.

Result: An explorer is activated on the "*TestProject*" test project.

Creating resource files

In order to translate all the messages and commands belonging to your module, resource files must be duplicated, by carrying out the following steps.

1 - Copy the <OBJING_PATH>\modules\GenDocModule directory and name the newly created directory "*ClassFile*".

2 - Rename the directory, specifying the version number (1.0 instead of 4.4) (as shown in Figure 2-20):

Figure 2-20. The "*ClassFile*" module directory

3 - In the <OBJING_PATH>\modules\ClassFile\0.0\res directory, rename the "*GenDocModuleIhmlabel.us*" file "*ClassFileIhmlabel.us*".

4 - Restart Objecteering/UML in order to take into account these modifications.

Configuring a module

Define the parameters of your new module.

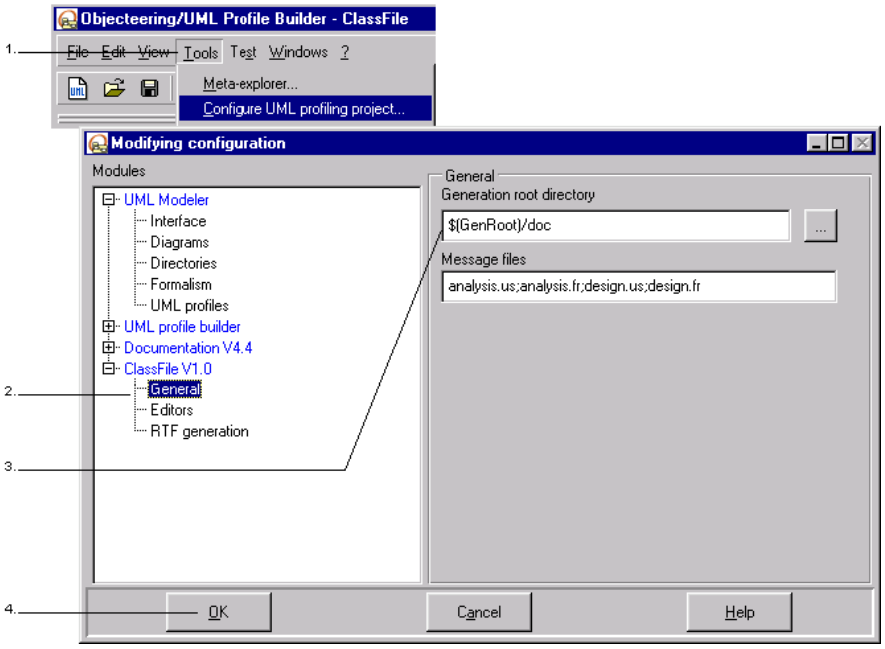


Figure 2-21. Configuring the "*ClassFile*" module

Steps:

1 - Select the "*Configure the UML profiling project*" option from the "*Tools*" menu.

2 - Choose the "*ClassFile*" tab.

3 - Enter module parameter values in the three sub-categories.

4 - Confirm.

Creating a document work product

Create the document work product by using the "*ClassFile*" module, then the "*Class File*" document template.

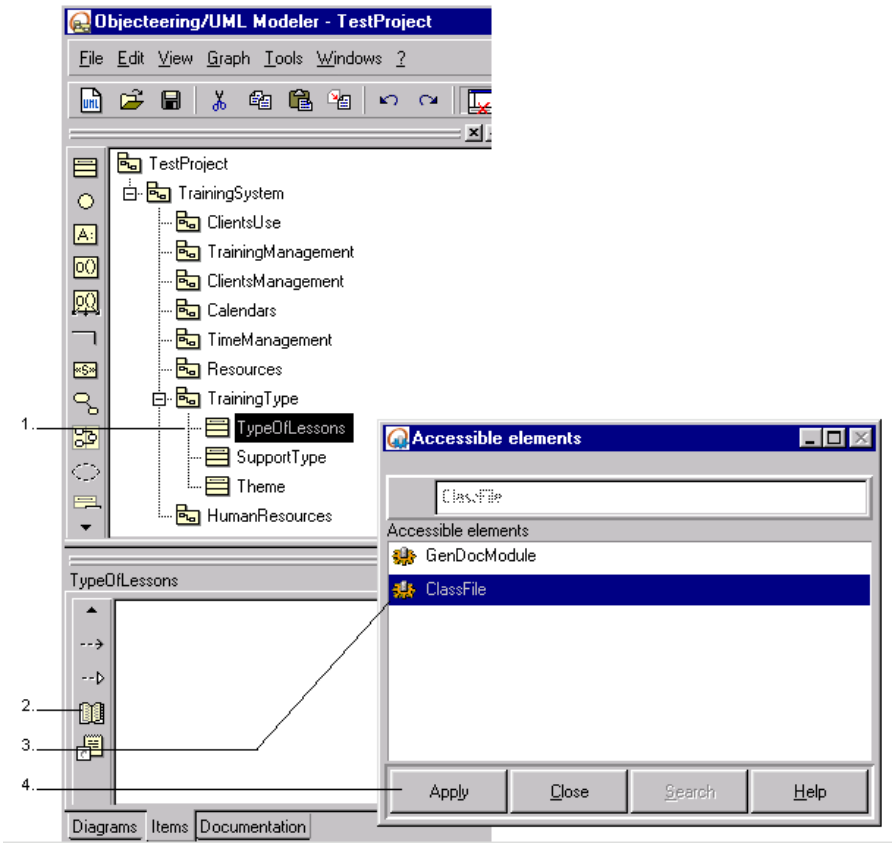


Figure 2-22. Creating a document work product

Steps:

1 - Select a class.

2 - Click on the  "*Create a document*" icon in the "*Items*" tab of the properties editor.

3 - Select the "*ClassFile*" module and click on "*Apply*". The window for generating documentation is launched automatically.

4 - Continue with the operations in figure 2-23.

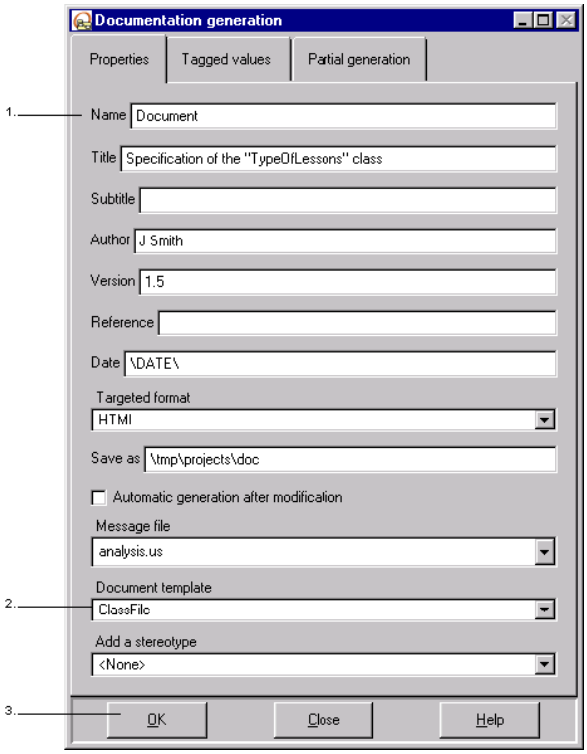


Figure 2-23. Entering values for the document work product

Steps:

1 - Enter values for the document work product.

2 - Select the "*ClassFile*" document template.

3 - Confirm.

Generating the document

Generate the document by executing the "*Generate*" command of the document work product..

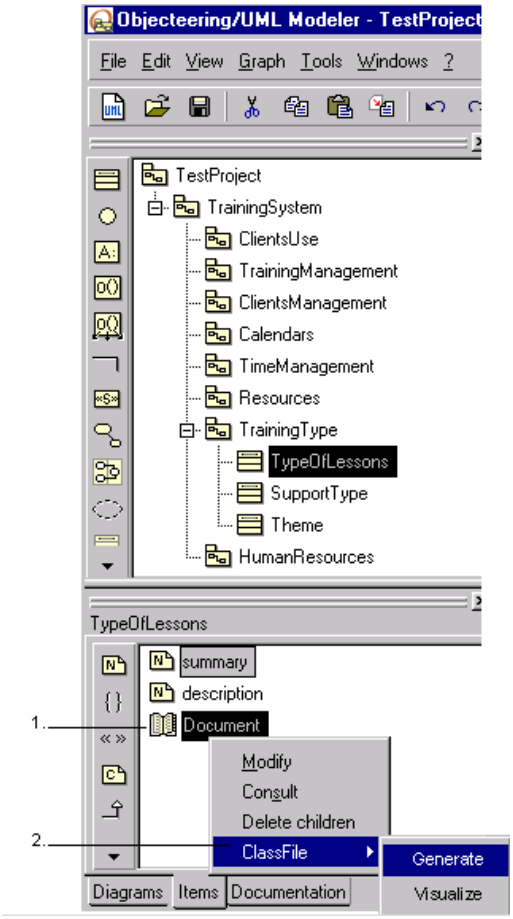


Figure 2-24. Generating the document on the "*TrainingType*" class

Steps:

1 - Select the document work product with the right mouse-button in the "*Items*" tab of the properties editor.

2 - Launch the "*ClassFile/Generate*" command.

Visualizing the generated document

Visualize the documentation by executing the "*Visualize documentation*" command of the document work product.

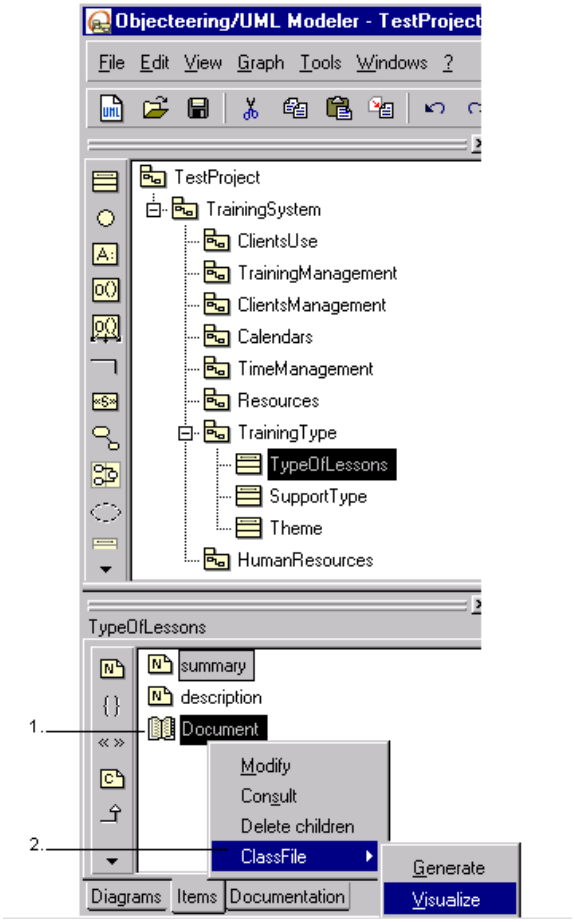


Figure 2-25. Visualizing the document generated on the "*TypeOfLessons*" class

Steps:

1 - Select the document work product in the "*Items*" tab of the properties editor.

2 - Launch the "*ClassFile/Visualize*" command.

Chapter 3: Tool for building document templates

Creating a document template project

Launching Objecteering/UML

The document template editor tool is launched in the same way as *Objecteering/UML Modeler*. For information on how to launch the tool, see the "*Launching the Objecteering tool*" section in chapter 1 of the *Objecteering/UML Modeler* user guide.

The first window you will see

The window shown in Figure 3-1 is the first to appear when Objecteering/UML is launched.

From this window you can:

- create a new document template
- open an existing document template

Figure 3-1 shows the creation of a new document template project.

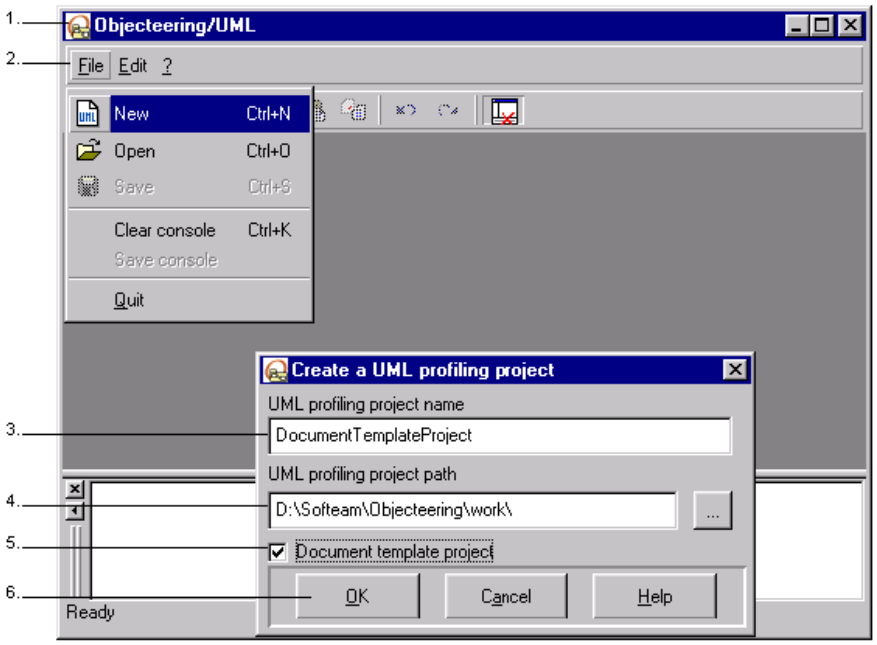


Figure 3-1. Creating a document template project

Steps:

1 - Launch the *Objecteering/UML Profile Builder* tool.

2 - Click on the "*File/New*" menu. The "*Create a UML profiling project*" window will then appear.

3 - In the "*UML profiling project name*" field, enter the name of the new document template project.

4 - In the "*UML profiling project path*" field, enter the path of the directory where the new document template project is to be created. You may also use the  icon to open a file browser, in which you can select your document template project path.

5 - Check the "*Document template project*" tickbox.

6 - Confirm.

Result

The creation of a document template project triggers:

- the creation of a UML profile with the same name as the "*Documentation*" UML profile
- the creation of a module with the same name (this module is only available and can only be modified using the *Objecteering/UML Profile Builder* module)

Document template explorer

Overview

The document template explorer is used to:

- create UML profiles
- create, copy or move document templates (it is not possible to move document templates delivered with Objecteering/UML)
- create, reference, copy or move document items
- reference J methods to define filter and generation rules

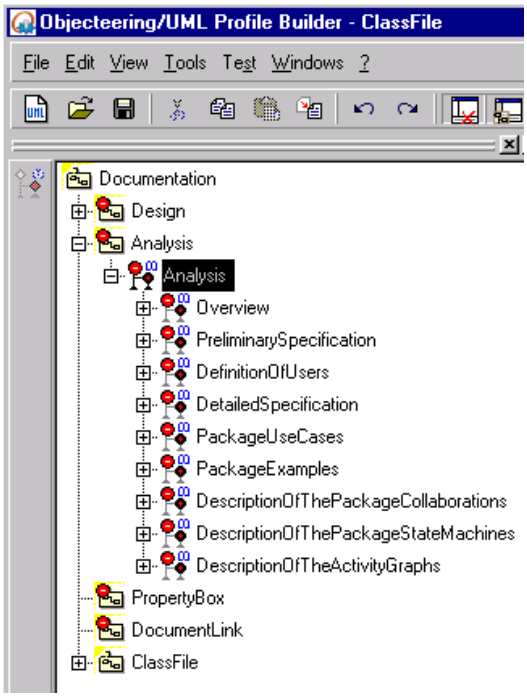


Figure 3-2. Document template explorer

Creation buttons

These buttons are located in the explorer.

The button ...	is used to create...	in ...
	a UML profile	a UML profile
	a document template	a UML profile
	a document item	a document template or a document item

Referencing buttons

These buttons are located in the "*Items*" tab of the properties editor.

The button ...	is used to reference...	on ...	to define ...
	a J method	a document item	a filter rule
	a J method	a document item	a pre-generation rule
	a J method	a document item	a generation rule
	a J method	a document item	a post-generation rule

Document template

Overview

 A document template is a consistent group of document items. It represents the final document and is used to define the header and footnote, as well as to automatically insert a table of contents.

It describes a model metaclass (package, class, operation, etc.). This metaclass allows you to dedicate the document template to a type of model item.

This document template is used for a document work product on a model item with an appropriate metaclass.

For example, a document template that describes a package can only be used by document work products associated to packages. However, when the generation is partial, the document template can describe another metaclass, but in this case, the document items provided in the "*Partial Generation*" tab are linked to the associated model item.

Dialog box

A document template is created in a UML profile, by clicking on the "*Create a document template*" button. There are three tabs in the document template dialog box - "*Properties*", "*Tagged values*" and "*Notes*". For further information on these standard dialog box tabs, please refer to the "*Standard dialog box tabs*" section of the *Objecteering/Model Dialog Boxes* user guide.

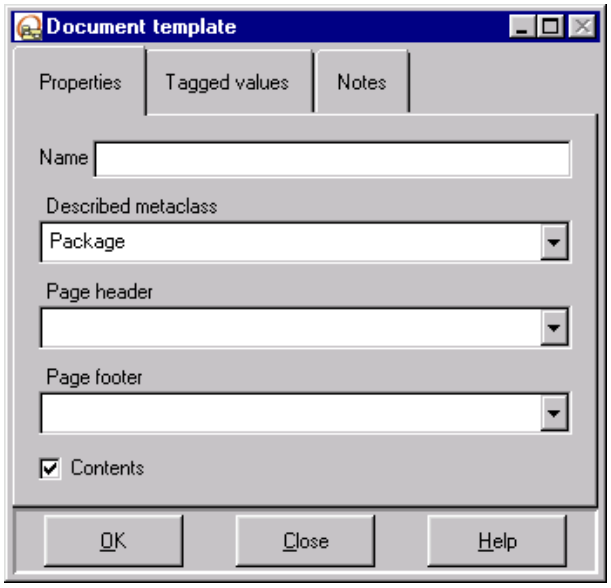


Figure 3-3. The "*Document template*" dialog box

Definition of fields in the dialog box

The ... text zone	defines ...
Name	the name of the document template. This name appears in the scrolling list of the document work product, allowing you to select the document template.
Described metaclass	the metaclass that the document template describes. This metaclass allows you to define from which model item the model's journey is carried out.
Page header	the content of the page header ($TITLE, $AUTHOR, $REFERENCE, $VERSION, $DATE, \PAGENUM\).
Page footer	the content of the footnote ($TITLE, $AUTHOR, $REFERENCE, $VERSION, $DATE, \PAGENUM\).
Contents	the automatic insertion of a table of contents.

The $TITLE, $AUTHOR, $REFERENCE, $VERSION and $DATE variables represent respectively the values of the title, author, reference, version and date text zones of the document work product. The "*Page header*" and "*Page footer*" scrolling lists are used to select one of the variables, but the zone can be entered manually in order to combine the variables and enter free text.

Example:

```
$AUTHOR  -  Page \PAGENUM\
```

Consistency rules

A document template must have a name. All characters can be used (spaces, accented characters, etc.). It is not possible to have two document templates with the same name defined on the same UML profile and on parent UML profiles.

A document template must describe a metaclass and the value of this metaclass must be part of the suggested list. Once the document template has been defined, it can only be modified if the document template contains no document item. This is because the document items that constitute it are defined using this metaclass. It is therefore an error to modify the metaclass if the document template is already made up of document items.

Redefining existing document templates

At present, it is not possible with Objecteering/UML to redefine a document template.

However, to customize an existing document template (the "*Analysis*" document template, for example), you can copy this document template. After this operation, there will no longer a be a link between the original document template and its copy. The two document templates will, from now on, on be completely separate.

Document item

Overview

A document item is in charge of the generation of a whole part of a document. It corresponds to a certain zone of the generated document.

Like the document template, a document item describes a metaclass. When it owns a described relation, the metaclass is the one defined by the association (for example, the document item which has "*Classes*" as its described association describes the "*Class*" metaclass). Otherwise, the document item describes the same metaclass as the document item which references it.

Dialog box

A document item is created in a document template or in a document item. In both cases, this document item is referenced.

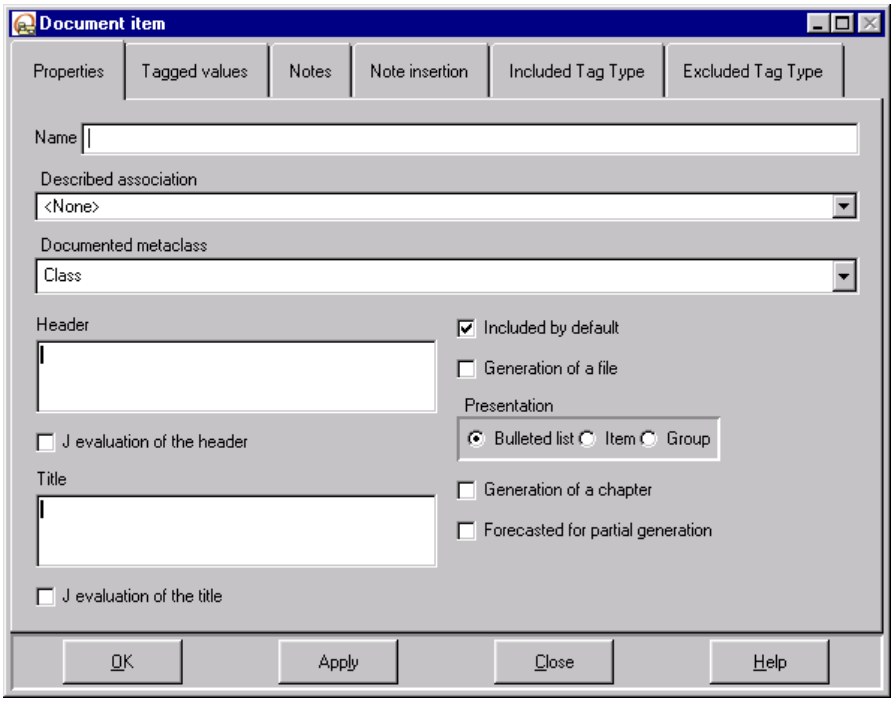


Figure 3-4. The "*Document item*" dialog box

Tabs

The tabs of this dialog box have the following purpose:

- Properties: used to enter the main values of the document item
- Tagged values: used to enter tagged values attached to the document item
- Notes: used to enter notes attached to the document item
- Note insertion: used to define the text types (Notes) to insert during the generation of the document item
- Included Tag Type: used to define the tagged values allowing you to select the model items to be generated
- Excluded Tag Type: used to define the tagged values allowing you to select the model items that must not be generated

"Properties" tab

The ... field	defines ...
Name	the name of the document item. This name appears in the scrolling list of a document item to reference it and also in the "Partial generation " tab of the document work product when the "Forecasted for partial generation" box has been ticked.
Described association	the metamodel's association to the document item (package classes, class attributes, etc.) browsed.
Documented metaclass	the metaclass impacted by the document item.
Header	the text inserted before the generation of all the model items the document item browses.
J evaluation of the header	the J evaluation of the text defined in the "Header" when the box has been checked. This allows J variables or expressions to be used, as opposed to a fixed text.
Title	the text inserted before the generation of all the model items the document item runs through.
J evaluation of the title	the J evaluation of the text defined in the "Title" when the box has been checked. This allows J variables or expressions to be used, as opposed to a fix text.
Included by default	the inclusion mechanism of model items run through according to their tagged values. If the box is checked, the model items which are run through are generated, except if they have been defined with a tagged value in the "*Excluded Tag Type*" tab.
Generation of a file	the generation in a file of the model item the document item runs through when the box is checked. This check box is used only for HTML format generation. For the other formats, it is ignored.
Presentation	the type of presentation for the generated items (list, heading, group).
Generation of a chapter	the generation of a chapter when the box has been checked.
Forecasted for partial generation	the permission for the document item to carry out partial generation, when the box is checked. If the box has been checked, the document item will appear in the "Partial generation" tab of the document work product.

"Tagged values" tab

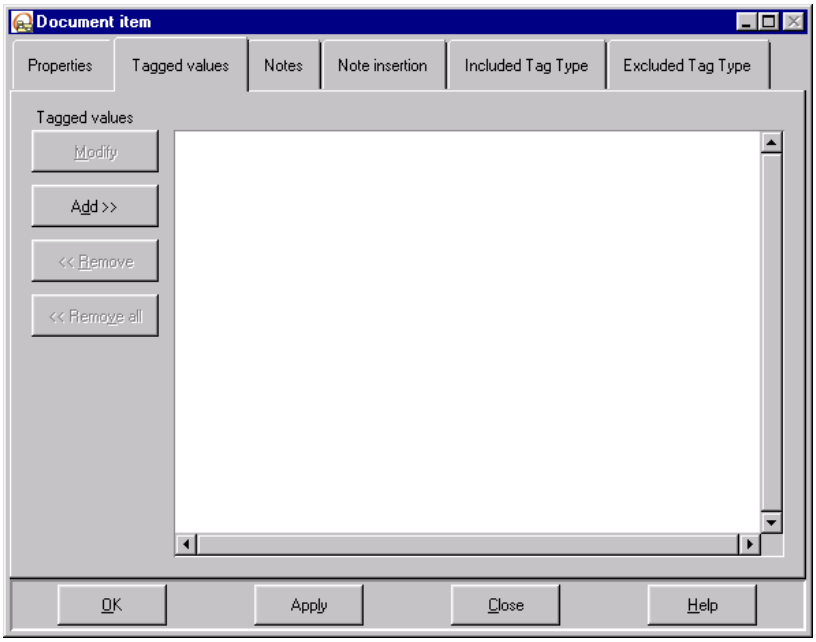


Figure 3-5. The "*Tagged values*" tab of the "*Document item*" dialog box

The ... button	is used to ...
Modify	modify the tagged value(s) in question.
Add	choose the tagged values the document item will insert.
Remove	remove the chosen tagged values the document item should insert.
Remove all	simultaneously remove in one go all the chosen tagged values the document item should insert.

"Notes" tab

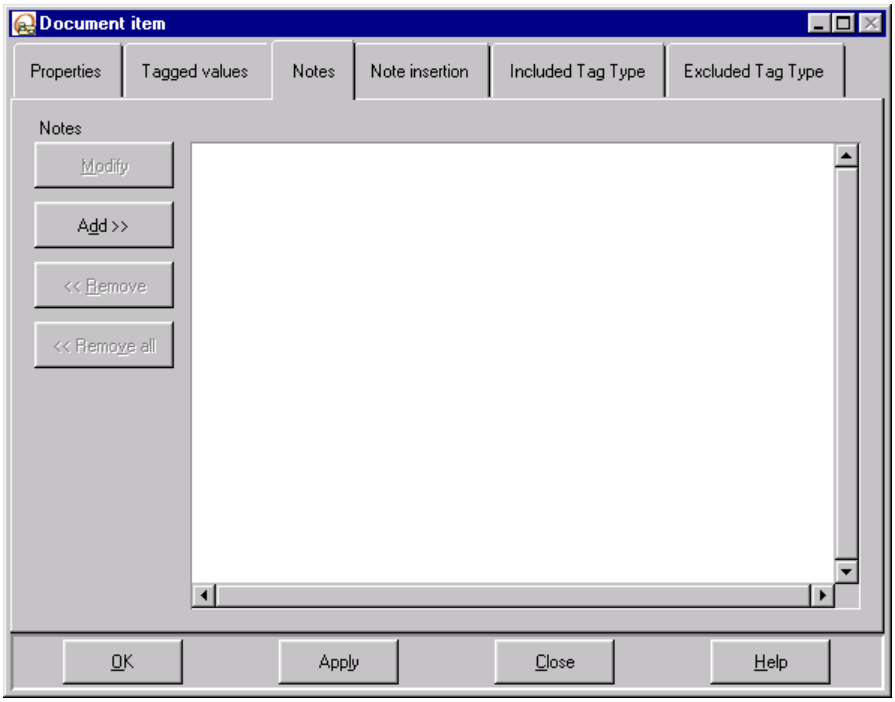


Figure 3-6. The "*Notes*" tab of the "*Document item*" dialog box

The ... button	is used to ...
Modify	modify the note(s) in question.
Add	choose the notes the document item will insert.
Remove	remove the chosen notes the document item should insert.
Remove all	simultaneously remove all the chosen notes the document item should insert.

"Note insertion" tab

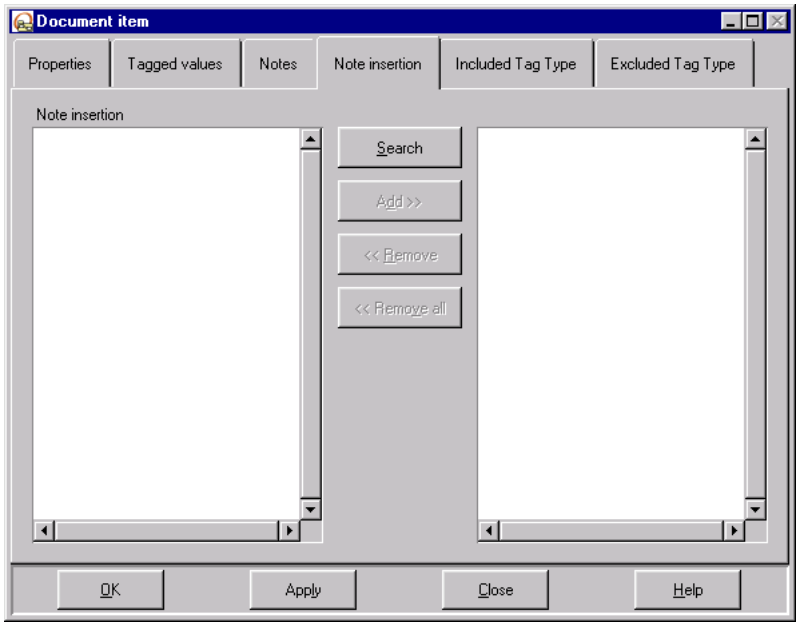


Figure 3-7. The "*Note Insertion*" tab of the "*Document item*" dialog box

The ... button	is used to ...
Search	search all the text types defined on the "Documentation" UML profile and on its parent UML profiles.
Add	choose the text types the document item will insert.
Remove	remove the chosen text types the document item should insert.
Remove all	simultaneously remove all the chosen text types the document item should insert.

"Included tag type" Tab

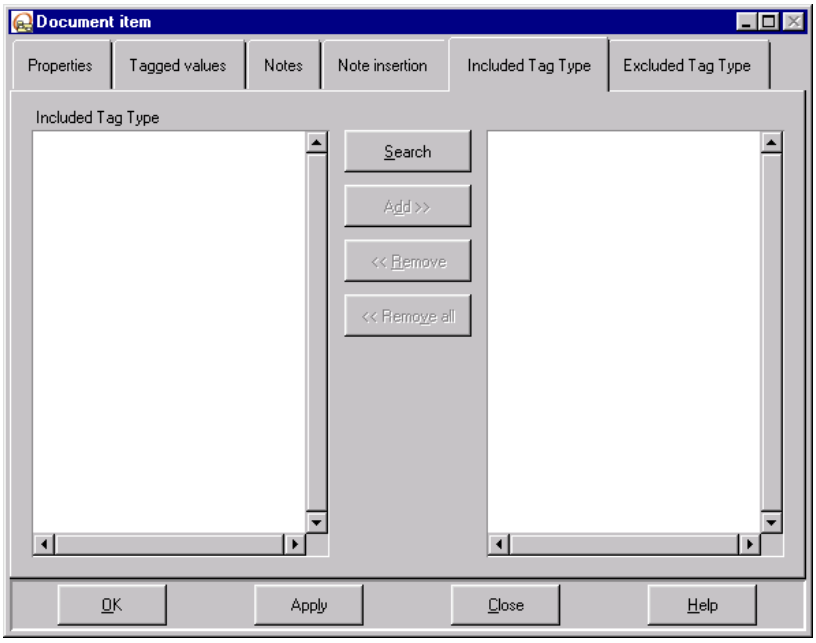


Figure 3-8. The "*Included tag type*" tab of the "*Document item*" dialog box

The ... button	is used to ...
Search	search all the tagged values defined on the "*Documentation*" UML profile and on its parent UML profiles.
Add	choose the tagged values allowing you to include the model items the document item runs through.
Remove	remove the tagged values allowing you to include the model items the document item runs through.
Remove all	simultaneously remove all the tagged values allowing you to include the model items the document item runs through.

"Excluded tag type" Tab

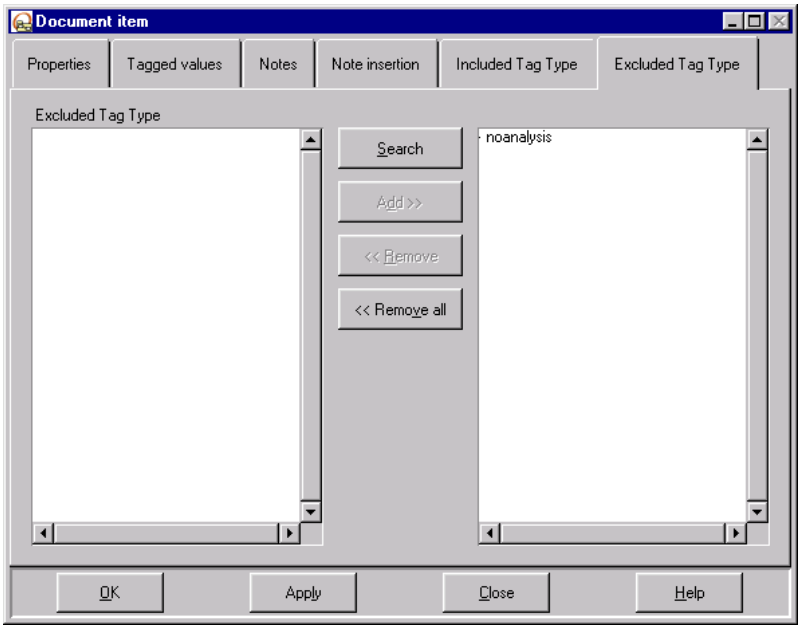


Figure 3-9. The "*Excluded tag type*" tab of the "*Document item*" dialog box

The ... button	is used to ...
Search	search all the tagged values defined on the "Documentation" UML profile and on its parent UML profiles.
Add	choose the tagged values allowing you to exclude the model item the document item runs through.
Remove	remove the tagged values allowing you to exclude the model item the document item runs through.
Remove all	simultaneously remove all the tagged values allowing you to exclude the model item the document item runs through.

Consistency rules

A document item must have a name. All characters can be used (spaces, accented characters, etc.) It is not possible to have two document items with the same name in a document template.

A document item only generates a chapter if the document items that reference it generate chapters.
The text types to insert and the tagged values, defined in the "*Text Insertion*", "*Included Tag Type*" and "*Excluded Tag Type* " tabs, must be compatible with the metamodel relation described by the document item. Indeed, it is possible to modify the relation the document item describes, after having defined the text types and the tagged values.

The document items referenced by the same document item must either all generate chapters, or none of them must generate chapters. According to the formats, template inconsistencies may exist in what is generated, in relation to the document template that is defined.

During copy/paste or copy/move, the metamodel association described by the document item must be compatible with the document item into which it is pasted or moved. It is not possible to copy or move a document item to any location whatsoever.

During a copy/move of a document template towards another, all the document items to be moved must only be referenced by document items to be moved. If a document item that is not meant to be moved, references a document item while it is being moved, then the move is canceled. A document item cannot be referenced by document items with different document templates.

Copy/Paste

Copying a document item implies copying all the document items it references.

When two document items to be copied (in the case of a multiple selection) reference the same document item, then this document item is copied only once and the two items to be copied then reference the same child item.

Reorganizing the document items

It is possible to rapidly restructure a document by moving the document items towards the top or towards the bottom.

The  and  buttons allow you to reorganize the document items.

Referencing document items

The document template editor can re-use in the same document template document items used for other parts of the document.

Furthermore, a document item can reference itself. For example, to recursively describe packages, the document item that describes the packages must reference itself, so the package description describes the packages which make it up.

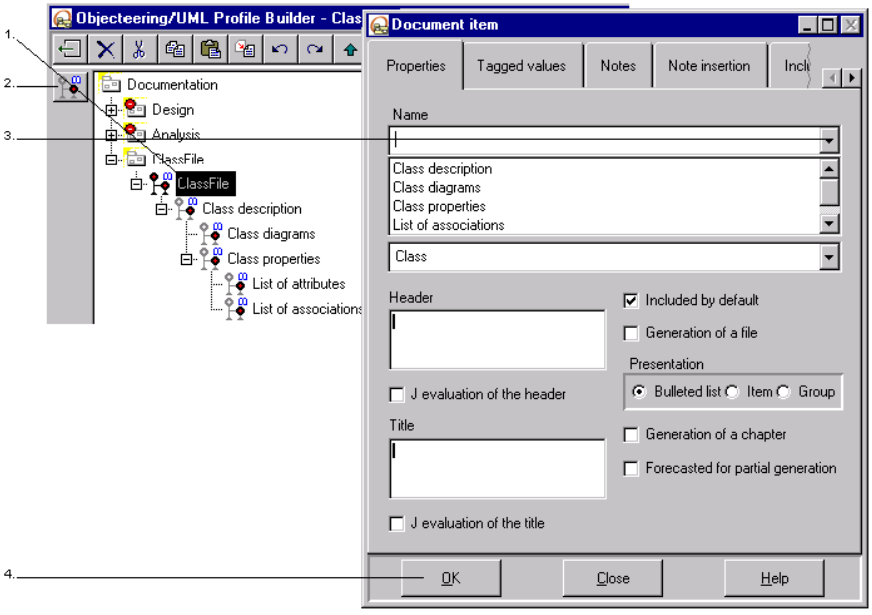


Figure 3-10. Referencing a document item

Steps:

1 - Select a document item.

2 - Click on the  "*Create a document item*" button.

3 - Select the document item to be referenced.

4 - Confirm.

Unreferencing document items

It is possible to unreference a document item. The document items that referenced it will then no longer launch the generation of this document item.

A document item can only be unreferenced if it is referenced by at least two document items.

To unreference a document item, select the document item in question and then click on the right mouse button and select the "*Unreference*" option from the context menu which appears.

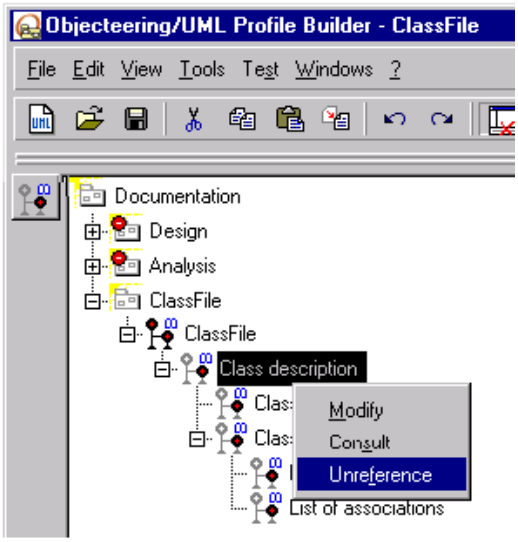


Figure 3-11. Unreferencing a document item

Note: When a document item is referenced only once, it is necessary to delete it so that the document item that previously referenced it no longer references it.

Deleting document items

Deleting a document item provokes the destruction of all the document items it references, even if they are referenced by another document item that is not deleted.

Note: To prevent a document sub-item from being deleted, you must first unreference it.

Rules

Overview

Each document item may define one or more rules. These rules can be used to obtain a dynamic behavior according to the current context, as well as to increase the amount of information to be generated in the document.

It is thus possible to filter the model items browsed by the document item or to insert additional information on these model items in the document.

There are two kinds of rules:

- filter rules
- generation rules

Selecting a rule

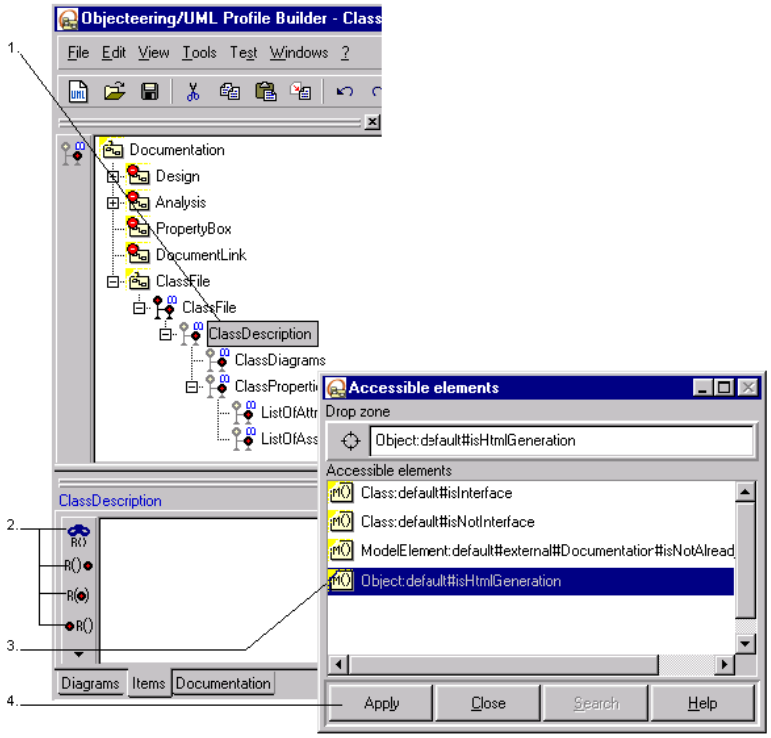


Figure 3-12. Selecting a rule

Steps:

1 - Select a document item in the explorer.

2 - Click on one of the rules in the "*Items*" tab of the properties editor.

3 - Select the item.

4 - Confirm.

Filter rules available on the "Documentation" UML profile

The filter rules allow you to obtain dynamic information on model items in order to orient generation.

The scanned model items are therefore filtered according to their properties and according to the document items already generated.

The *Objecteering/UML Profile Builder* module makes it possible to define new rules in the J language.

The ... rule	on ...	determines if ...
isAbstract	members	the member is abstract.
isActivitydiagram	diagrams	the diagram is an activity diagram.
isClass	members	the member is class.
isClassDiagram	diagrams	the diagram is a class diagram.
isCollaborationDiagram	diagrams	the diagram is a collaboration diagram.
isComponentDeploymentDiagram	diagrams	the diagram is a deployment diagram.
isHtmlGeneration	objects	the generation format is HTML.
isIn	parameters	the parameter is in.
isInOut	parameters	the parameter is in/out.
isInstanceDeploymentDiagram	diagrams	the diagram is a deployment instance diagram.
isInterface	classes	the class is an interface class.
isNotAlreadyGenerated	model items	the current item has already been generated (this operation is linked to the recordGenerationFact rule, which allows the recording of the fact that an item has been generated).
isNotInterface	classes	the class is not an interface class.

The ... rule	on ...	determines if ...
IsNotRoot	states	the state is not the root state.
isNotSubSystem	packages	the package is not a sub-system.
isObjectDiagram	diagrams	the diagram is an object diagram.
isOut	parameters	the parameter is out.
isPrivate	members	the member is private.
isProtected	members	the member is protected.
isPublic	members	the member is public.
isStateDiagram	diagrams	the diagram is a state diagram.
isSubSystem	packages	the package is a sub-system.
isToDescribe	operations	the operation to be described. An operation is only described if it has a *description* type note.
isUseCaseDiagram	diagrams	the diagram is a use case diagram.

Generation rules

Generation rules allow you to increase the amount of information to be inserted into the generated document, according to the modeling item which is in the process of being generated. Document items only generate the strict minimum.

Generation possibilities are thus countless.

The ... rule	on the ...	generates ...
generateCompleteSyntax	attributes	the complete attribute's syntax.
generateCompleteSyntax	parameters	the complete parameter's syntax.
generateConstraints	items	the list of constraints.
generateDescriptionSyntax	operations	the operation's syntax.
generateInheritance	actors	the list of actors which the actor specializes.
generateInheritance	operations	the name of the parent operation.
generateInheritance	use cases	the list of use cases which the use case specializes.
generateInheritance	classes	the list of classes which the class specializes.
generateInvariantDescription	classes	the description of the invariants of the class.
generateInvariantDescription	packages	the description of the invariants of the package.
generateItIsAnAbstractPackage	packages	a piece of information on the package's state of abstraction.
generateInOut	parameters	the parameter's passing mode (in, out or in/out).
generateOwner	namespaces	the component name.
generatePostCondition	operations	the description of the post-conditions.
generatePreCondition	operations	the description of the pre-conditions.
generateStereotype	items	the stereotype.
generateSyntax	attributes	the attribute's syntax.
generateSyntax	instance attributes	the instance attribute's syntax.

The ... rule	on the ...	generates ...
generateSyntax	components	the component's syntax.
generateSyntax	enumerates	the enumerate's syntax.
generateSyntax	states	the state's syntax.
generateSyntax	node instances	the node instance's syntax.
generateSyntax	instances	the instance's syntax.
generateSyntax	component instances	the component instance's syntax.
generateSyntax	sequence instances	the sequence instance's syntax.
generateSyntax	sequence messages	the sequence message's syntax.
generateSyntax	nodes	the node's syntax.
generateSyntax	operations	the operation's syntax.
generateSyntax	parameters	the parameter's syntax.
generateSyntax	roles	the role's syntax.
generateSyntax	transitions	the transition's role.
generateTaggedValues	items	the list of tagged values.
generateUseCaseExtend	use cases	the list of use cases used by the use case by extend.
generateUseCaseUses	use case	the list of use cases used by the use case.
includeDiagram	diagrams	the diagram.
recordGenerationFact	items	the recording of the current item as being a described item (this operation is linked to the isNotAlreadyGenerated filtering rule, which only allows the generation of an item if it has not yet been generated.

Chapter 4: Generation principles

Principles

Overview

The document template groups together a consistent set of document items that each generate a specific document zone.

The document template also generates the header and footnote, as well as the table of contents (if the "*Contents*" box has been checked). All this information is generated in the document items contained in the document template.

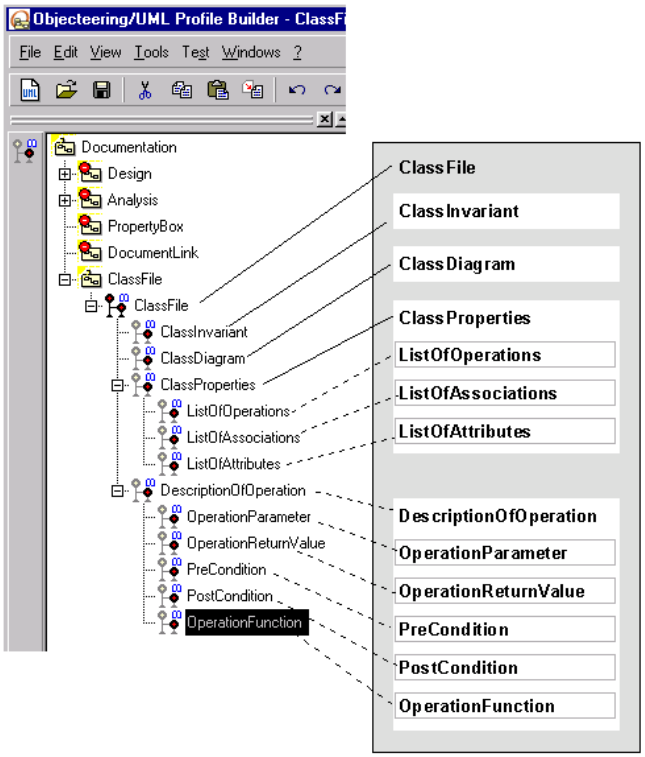


Figure 4-1. Example of a representation of a generated document in comparison with the document template

Generation order of a document item

A document item generates several zones following a predefined order.

The information to be generated is taken from the document item (title, header, etc.) from its generation rules and from the model items browsed (name, notes, etc.).

The information generated by a document item is subdivided into several parts. The first contains the value of the header defined in the document item. The model items the document item runs through generate the other parts.

When a model item is browsed, you generate the pre-generation title and rules, the notes, whose types are defined in the document item, the generation rules, the document items it references, and the post-generation rules.

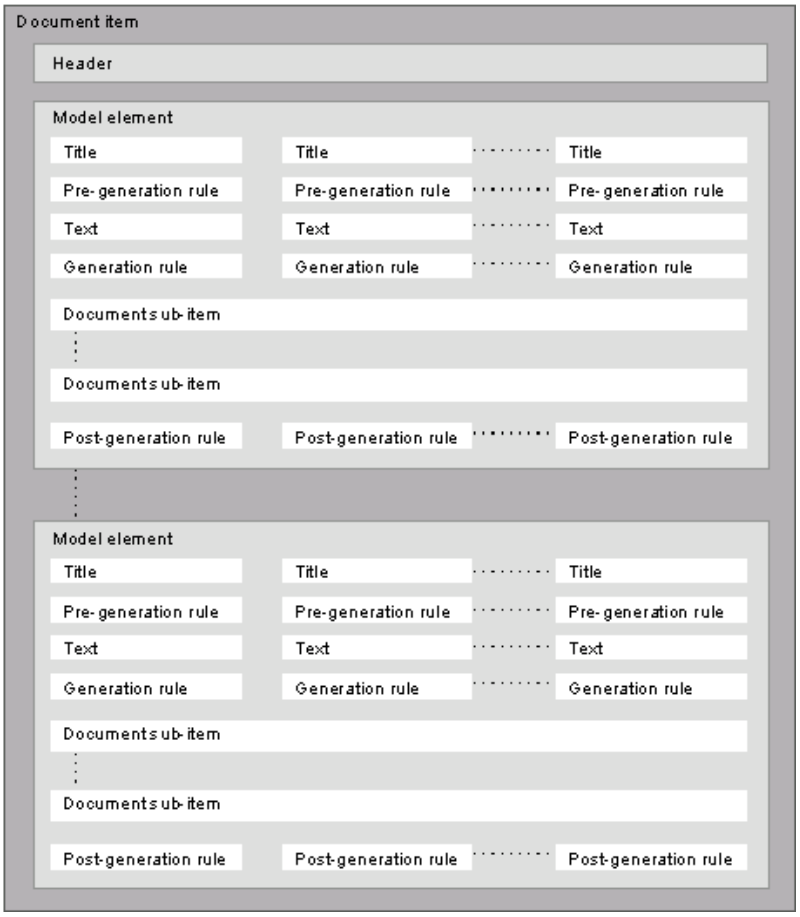


Figure 4-2. Generation order of a document item

Including an item

There are two ways to include the model item a document item runs through, either taking into account or not taking into account the annotations.

When the "*Included by default*" box is checked, the model items run through are systematically generated, except if they own a tagged value defined in the "*Excluded Tag Type*" tab. This box allows you to choose whether or not to take into account the content of the "*Inclusion*" tab.

When the "*Included by default*" box is not checked, the model items run through are not generated, except if they are annotated with a tagged value defined in the "*Included Tag Type*" tab.

Excluding an item

A document item does not generate a model item annotated with a tagged value defined in the "*Excluded Tag Type*" tab.

Note: The exclusion tagged value is stronger than the inclusion tagged value. If a model item is annotated by two opposed tagged values, then the item is not generated.

J evaluation of the header and title

When the "*J evaluation of the header*" or "*J evaluation of the title*" are checked, it is assumed the "*Header*" or "*Title*" zones contain J instructions (please refer to the *Objecteering/The J Language* user guide for further information). They are then evaluated dynamically. This mechanism allows you, amongst other things, to evaluate the J methods or to run through metaclass attributes (Name, etc.).

However, not all J instructions are allowed. Only character strings expressions separated by commas are authorized. J evaluates expressions and concatenates the string results separated by commas. The result is a string inserted in the documentation.

Example 1:

When the "Title" zone contains:

```
getMulMessage ("analysis.fr", "Class", Name)
```

and the "*J evaluation of the title*" is selected, the generation gives the following result when the document item runs through the "*Training*" class:

```
Class Training
```

Note: The J method getMulMessage allows to search a message in an internationalization file ("*analysis.fr*") according to an identifier ("*Class*").

Example 2:

When the "Title" zone contains:

```
"Class", Name
```

This will provide the same result, but without the internationalization service.

Filter rules

Filter rules allow the dynamic interrogation of a model element browsed by the document item.

For each documentation item, the filter rules are evaluated in the order in which they are displayed in the explorer. The documentation item is then only generated if the filter rules are applied.

If a rule is not respected, the following rules are systematically not evaluated.

Generation rules

Each type of rule (pre-generation, generation, post-generation) inserts a piece of information extracted from the browsed model element.

For each documentation item, the generation rules are evaluated in the order in which they are displayed in the explorer. They can modify the current context by keeping the information that will be useful for the generation that will follow. They can also include diagrams, retrieve information on the user model or recapture messages defined in an external file.

HTML generation

In HTML generation, the "*Generation in a file*" box allows the generation of the document in a separate file from the master file.

When the box is checked, each model item is run through by the document item stored in a file named after the model item's identifier.

Hypertext links are always generated, even when the link's target file does not exist. It is necessary for the files containing the description of the packages and classes to exist, so that the links towards these files be in working condition. This is why it is necessary to produce document items that generate packages, and classes in the files.

Note: The generation of the hypertext links and file names cannot be modified by the editor of document templates. To carry out this modification, you have to use the *Objecteering/UML Profile Builder* tool.

Overview of a generated item

Introduction

Many document items become repetitive when documentation is generated (package classes, class attributes, etc).

All document items are generated with a title and a content. The presentation of a generated item determines the layout of the title and content.

This presentation is only possible if the document item does not generate a chapter (the "*Generate a Chapter*" box is not checked).

There are three types of presentation:

- a bulleted list
- an item
- a group

Bulleted list: Description

This form is aimed at short descriptions of model items, which will each be presented on a separate line with a list separator:

```
Title1: content1
Title2: content2
```

Bulleted list: Example

```
"CourseName" parameter : (in) name of the course
"TrainingType" parameter : (in) type of the course
```

Item: Description

This form is aimed at long descriptions of model items, which will be presented under the title:

```
title1
content1
title2
content2
```

Item: Example

```
"Client" class
The clients defined here are the ones managed by marketing
and accounting, and by the system that administrates the
training.
"ManageClients" class
The clients' orders, billing and quality follow-up are
administrated thanks to the training.
```

Group: Description

This form is aimed at model items with no description or very short descriptions, presented on one line separated by commas:

```
title1, content, title2, content
```

Group: Example

```
Name, Given Name, Activity, Age
```

Internationalizing messages

Overview

It is possible to separate the messages to be inserted during documentation generation from the document items and the document template.

This message separation from the document template allows the execution of different documentation generations with the same document template: generation in English, with UML formalism, customizing the generated information, etc.

Using the message file

There are two ways of using messages:

- for generation rules
- in the "*Header*" and "*Title*" text zones of a document item

The definition of the corresponding message identifier in the message file used is required when using predefined generation rules.

For the information stored in the "*Header*" and "*Title*" zones, it is necessary to use a method that will allow you to retrieve a message in a file.

The following J method allows you to retrieve the value of an identifier of a message contained in a file.

```
String getMulMessage (in String messageFile, in String
ident, in String param1, in String param2, ...)
```

The message file defined in the document work product is represented by the MSG_FILE character string. This string is automatically updated when the generation is launched. It is therefore necessary to use this constant.

Example:

The following example returns a message whose identifier is "*Overview*" and which is contained in the file defined in the document work product in the "*Message file*" text zone.

```
getMulMessage (MSG_FILE,"Overview")
```

In the method, it is possible to pass parameters. These parameters are represented in the message in the form of %1, %2, etc.

Example:

```
The class specializes the %1 class.
```

Selecting a message file

All the messages must be grouped together in a file in the directory :
`$OBJING_PATH/modules/<module_name>/<module_version>/res`

A specific message file is chosen from the document item dialog box.

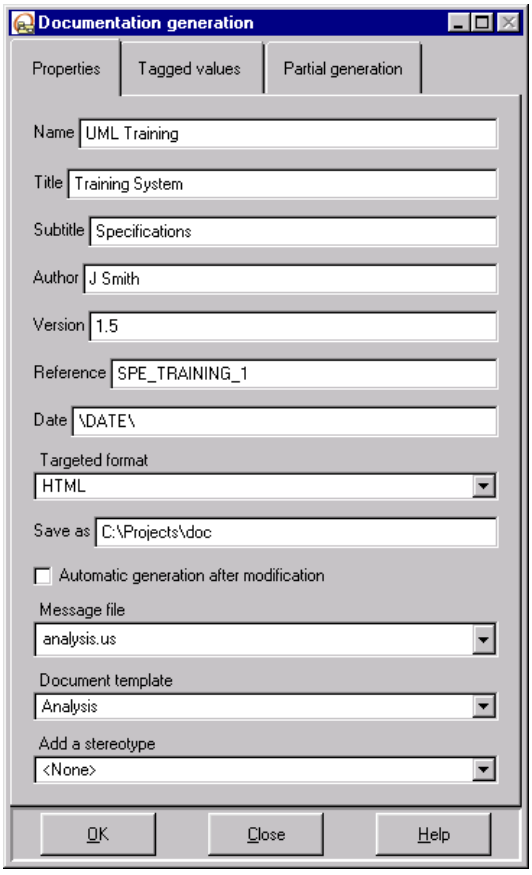


Figure 4-3. Selecting a message file for the document item

Defining a message

In the message file, a message must have the following syntax:

```
Identifier:
This is the translation of the message.
end Identifier
```

Example:

```
Overview:
General presentation
end Overview
```

Index

Overview

In HTML, an index which contains all those elements browsed by the document template can be generated. These elements are grouped by "*NameSpace*" and in alphabetical order within each group.

By clicking over the name which interests you, you can display its description in the right-hand window.

In order for an element to be indexed, the item which describes this element must be annotated using the *{index}* tagged value. All elements describing this item are then added to the index. In order to easily find an element in the generated index, items are grouped by metaclass. The different groups are packages, classes, interfaces, types, enumerations, use cases, actors, components, nodes, signals and those metaclasses which are not *NameSpaces*.

Example

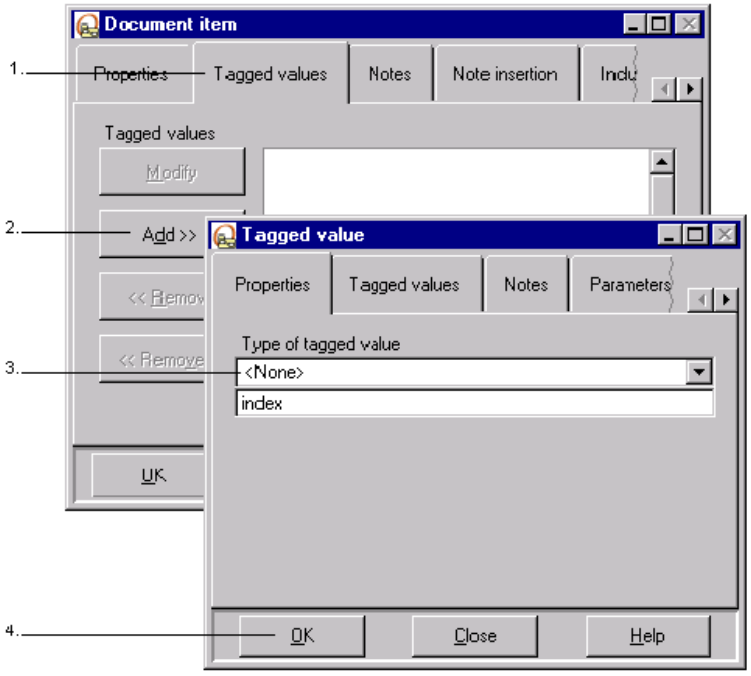


Figure 4-4. Annotating a document item

Steps:

1 - After selecting the "*Modify*" context menu item for the document item in question, click on the "*Tagged values*" tab.

2 - Click on "*Add*".

3 - Click on the "*Type of tagged value*" combo box and select the *{index}* tagged value.

4 - Confirm.

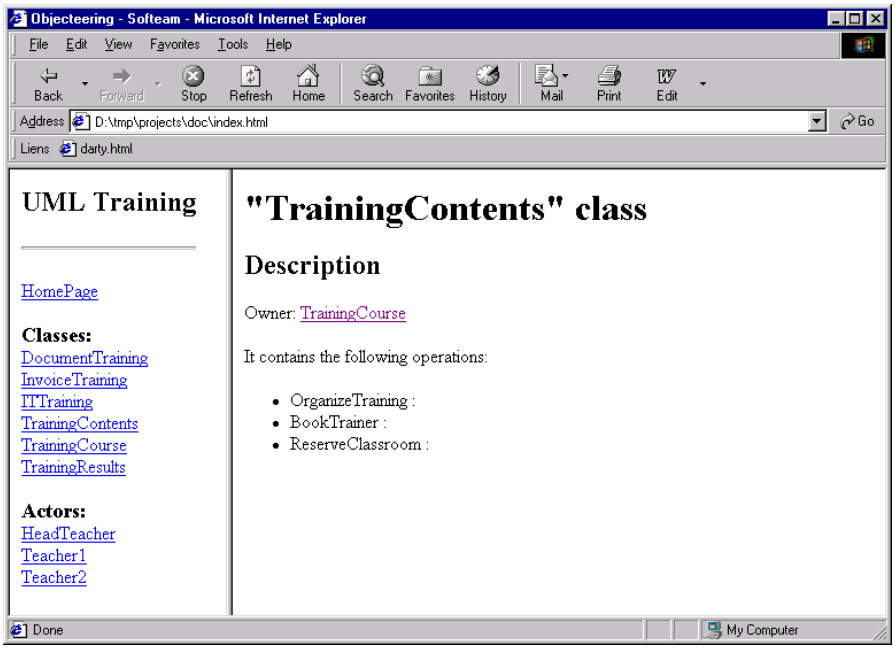


Figure 4-5. Visualizing a generated index

Detailed description of generation rules

Overview

Certain operations use message identifiers that are stored in external files, allowing you to run generations in different languages. Using these operations in a document template implies that this identifier must be translated in the file that will be used for the generation (for example, design.us), i.e. that will be defined in the document work product.

Actor::generateInheritance

This operation allows you to insert the list of actors which specialize the current actor. It runs through the metamodel relation "*ParentGeneralization.<SuperTypeActor*".

The message identifier used is "*SpecializesActor*".

Generation example:

```
I1 specializes the actorParent1, actorParent2 actor.
```

AssociationEnd::generateSyntax

This operation allows you to insert the syntax of an association role in the following form:

```
associationName : (cardinality) as (link) towards
(opposedClass) (opposedCardinality) as (opposedLink)
```

Generation example:

```
Allocation : (0-*) as user towards Station (0-*) as used
```

Attribute::generateCompleteSyntax

This operation allows you to insert an attribute's syntax in the following form:

```
Type: visibility class const unsigned short
array(Multiplicity) of className (TypeConstraint/StringSize)
attributeName=defaultValue
```

Generation example:

```
public class const array(10) of string(25)=125
```

Attribute::generateSyntax

This operation allows you to insert an attribute's syntax in the following form:

```
visibility class const unsigned short size
className[TypeConstraint/StringSize]
attributeName=defaultValue
```

Generation example:

```
public unsigned int number=0
```

AttributeLink::generateSyntax

This operation allows the insertion of the syntax of an instance attribute in the following form:

```
attributeName:className=value
```

Generation example:

```
age:float=32
```

Class::generateInheritance

This operation allows the insertion of the classes which the current class specializes. It runs through the metamodel relation "*ParentGeneralization.<SuperTypeClass*".

The message identifier used is "*SpecializesClass*".

Generation example:

```
This class specializes the ParentClass1 class.
```

Class::generateInvariantDescription

This operation allows the insertion of the contents of all the current class' constraints which have the "*invariant*" stereotype.

ClassifierRole::generateSyntax

This operation allows the insertion of the syntax of a role in the following form:

```
role:className
```

In the case of HTML format generation, it adds a hypertext link to the file which has the unique identifier of the instance class as its name.

Generation example:

```
object:C1
```

Component::generateSyntax

This operation allows the insertion of the component name.

ComponentInstance::generateSyntax

This operation allows the insertion of the component instance name.

Diagram::includeDiagram

This operation allows the insertion of the current *diagram* and a legend. It inserts a reference to a file containing the diagram and whose name is unique. The file is stored in a directory whose name is the identifier of a document work product, followed by a chain of "*.img*" characters.

The format of the file depends on the type of format requested (HTML, Postscript, Rtf) and on the platform (Windows, Unix).

In HTML, generated diagrams contain "clickable" zones, which are used to open the description of an element by clicking on the element in question in the diagram. This method is, therefore, used to generate zones in the image which are sensitive to clicking. Links are defined by the method which returns the link. (For further information, please refer to the "*Diagrams*" section of chapter 3 of the *Objecteering/Documentation* user guide).

In RTF, pages containing diagrams which are too wide are generated in "*landscape*" mode. The "*Image with for landscape mode*" module parameter is used to specify the minimum width for which the page is generated in "*landscape*" mode.

Formatting in ...	**generates in UNIX a file in ... format**	**generates in Windows a file in ... format**
Ascii	n.a.	n.a.
HTML	Graphics Interchange Format (GIF)	Graphics Interchange Format (GIF)
Postscript	Encapsulated PostScript (EPS)	n.a.
RTF	encapsulated PostScript (EPS)	Windows Metafile (EMF)

Note: Generation in Ascii format does not generate files containing diagrams.

The message identifiers used for the diagram legend are:

The message ... identifier	corresponds to a ... diagram
ActivityDiagram	activity
CollaborationDiagram	collaboration
DeploymentDiagram	deployment
InstanceDeploymentDiagram	deployment instance
ObjectDiagram	object
SequenceDiagram	sequence
StateDiagram	state
StaticClassDiagram	class
UseCaseDiagram	use case

Enumeration::generateSyntax

This operation allows the insertion of the syntax of an enumerated type in the following form:

```
typeName (val1, val2, val3)
```

Instance::generateSyntax

This operation allows the insertion of an instance in the following form:

```
instanceName:className
```

In the case of HTML format generation, it adds a hypertext link to the file which has the unique identifier of the instance class as its name.

Generation example:

```
object:C1
```

ModelElement::recordGenerationFact

This operation allows the recording of the fact that a modeling operation has already been described. From the moment that a modeling element is recorded as being described, the "*isNotAlreadyGenerated*" filtering operation will ensure that this modeling element is not inserted again.

For example, the document item which allows the generation of the description of classes is referenced several times in the document template. By adding the "*isNotAlreadyGenerated*" operation as a filtering rule and the "*recordGenerationFact*" generation rule, the modeling element run through by this document item will only be generated once. In the case of HTML generation, this allows you to only generate once a file which contains the description of a modeling element.

Note: This operation returns an empty chain of characters and therefore inserts no information into the generated document. All it does is to inform the generator of the existence of a modeling element.

ModelElement::generateTaggedValues

This operation generates the list of *tagged values* defined on the described element.

The message identifier used is "*TaggedValues*"

Generation example:

```
Tagged values : {analysis}, {const}
```

ModelElement::generateStereotype

This operation generates the stereotype defined on the described element.

The message identifier used is "*Stereotype*".

Generation example:

```
Stereotype : {extend}
```

ModelElement::generateConstraints

This operation generates the list of constraints defined on the described element.

The message identifier used is "*Constraints*".

Generation example:

```
Constraints : {invariant}
```

NameSpace::generateOwner

This operation allows the insertion of the owner element. In the case of generation in HTML format, it adds a hypertext link to the file which has the unique identifier of the owner element as its name.

The message identifier used is "*Owner*".

Generation example:

```
Owner : Pack
```

Node::generateSyntax

This operation allows the insertion of the node's name.

NodeInstance::generateSyntax

This operation allows the insertion of the node instance.

Operation::generatePostCondition

This operation allows the insertion of the contents of all the operation's constraints which have the "postcondition" stereotype.

Operation::generatePreCondition

This operation allows the insertion of the contents of all the operation's constraints which have the "precondition" stereotype.

Operation::generateSyntax

This operation allows the insertion of the syntax of an operation in the following form:

```
visibility abstract final class operationName
(syntaxOfParameters)
return syntaxOfParameters
```

Generation example:

```
public abstract op (p1 in String) return boolean
```

Operation::generateDescriptionSyntax

This operation allows the insertion of the syntax of an operation in the following form :

```
visibility abstract final class operationName (
            syntaxOfParameters)
      return syntaxOfParameter
```

Generation example:

```
public abstract op (
        p1 in String
        p2 in boolean)
    return boolean
```

Operation::generateInheritance

This operation allows the insertion of the name of the parent operation and its container if the operation is a redefinition. In the case of HTML format generation, it adds a hypertext link to the file which has the unique identifier of the container as its name.

The message identifier used is "*SpecializesOperation*".

Generation example:

```
It specializes the "create" operation of "Class1".
```

Package::generateInvariantDescription

This operation allows the insertion of the contents of all the current package's constraints which have the "*invariant*" stereotype.

Package::generateItIsAnAbstractPackage

This operation allows the insertion of a piece of information where the package is abstract.

The message identifier used is "*ItIsAnAbstractPackage*".

Generation example:

```
It is abstract.
```

Parameter::generateInOut

This operation allows the insertion of a parameter's passage mode.

The message identifiers used are:

The message ... identifier	corresponds to the ... mode
ItIsInInput	in
ItIsInOutput	out
ItIsInInputOutput	in/out

Parameter::generateCompleteSyntax

This operation allows the insertion of the syntax for a parameter in the following form:

```
Type : array(<Multiplicity>) of
<TypeGeneralClass>(<TypeConstraint>)=<Value>
```

Generation example:

```
Type : array(17) of string(28)=212
```

Parameter::generateSyntax

This operation allows the insertion of the syntax for a parameter in the following form:

```
parameterName=defaultValue in size typeName[TypeConstraint/StringSize]
```

Generation example:

```
p1 in Set(*) int
```

SequenceMessage::generateSyntax

This operation allows the insertion of the syntax of a sequence message in the following form:

```
fromInstance to toInstance : messageName
```

Generation example:

```
:Accountant to :UserCompany : invoice
```

State::generateSyntax

This operation allows the insertion of the syntax for a state in the following form:

```
parentName::stateName
```

Generation example:

```
s1::s2::s3
```

Transition::generateSyntax

This operation allows the insertion of the syntax of a transition in the following form:

```
sourceStateName to targetStateName
```

Generation example:

```
s1 to s2
```

UseCase::generateInheritance

This operation allows the insertion of the list of use cases which generalize the current use case.

It runs through the metamodel relation "*ParentGeneralization.<SuperTypeUseCase*".

The message identifier used is "*SpecializesUseCase*".

Generation example:

```
It specializes the case1, case2 use case.
```

UseCase::generateIncludedUseCases

This operation allows the insertion of the list of use cases which are included by the current use case.

The message identifier used is "*IncludedUseCase*".

Generation example:

```
It includes the case1, case2 use case.
```

UseCase::generateExtendedUseCases

This operation allows the insertion of the list of use cases which are extended by the current use case.

The message identifier used is "*ExtendedUseCase*".

Generation example:

```
It includes the case1, case2 use case.
```

Chapter 5: Using a document template

Module configuration

Editing the configuration

The module containing the document template inherits from the documentation generation module that is part of standard Objecteering/UML delivery. It contains and uses the same parameters.

Whatever the document template used, the document work product parameters will be included in the module created by the edition of the document templates.

For further information, see chapter 4, "*Documentation generation parameters*", of the *Objecteering/Documentation* user guide.

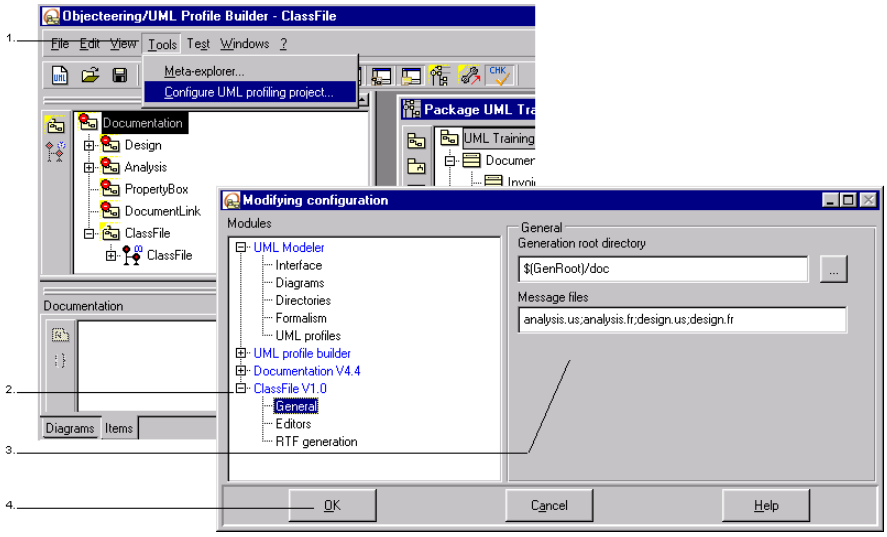


Figure 5-1. Configuring the module

Steps:

1 - Select the "*Configure the UML profiling project*" command in the "*Tools*" menu. The "*Modifying database configuration*" window then appears.

2 - Expand the "*ClassFile*" heading. Three sub-categories of parameter then appear.

3 - Enter the module's values.

4 - Confirm.

Test project

The test project

The created document template is stored in a UML profiling project.

The test project is a project specifically associated to a UML profiling project. Any modification in the UML profiling project automatically affects the test project. This allows the immediate testing of any modification of the document template on the test project. There is no module installation procedure to carry out.

Selecting a test project

The test project must be selected by the user.

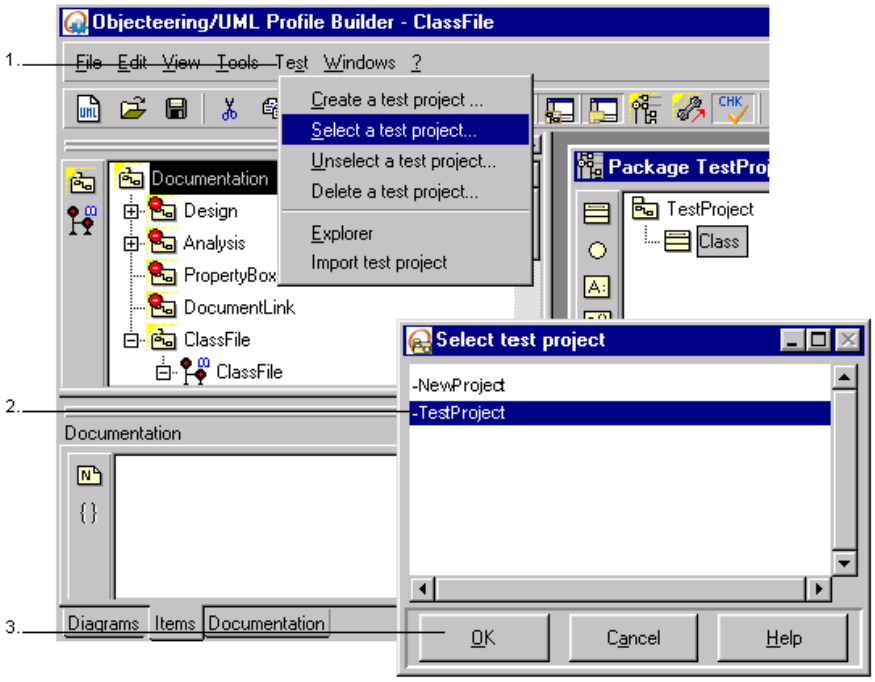


Figure 5-2. Selecting a test project

Steps:

1 - Select the "*Select a test project*..." option from the "*Test*" menu.

2 - Select the test project.

3 - Confirm.

Packaging and delivering

Overview

In order to be able to use the document template you have developed in other databases, it must be packaged and delivered to the user site in question. These packaging and delivery operations are carried out in command line only.

Packaging

The packaging operation consists of creating a *.prof* file, containing all necessary module information. This operation can only be carried out in command line mode, and is used as follows:

```
objing -package <database name> <UML profiling project
name> <module name>
[-version <target version>] [-lock] [-hide]
[-prj <UML modeling project name>]
<deliverModuleFilePath>
```

For example:

```
objing -package MyBase ClassFile ClassFile -version0.0
C:\Program Files\Objecteering
```

Delivering

This operation is used to render the newly developed document template available in any database

For further information on this operation, please refer to the "*Detailed view of the Configuration menu*" section in chapter 3 of the *Objecteering/Administrating Objecteering Sites* user guide.

Selection

In a new database, create a new UML modeling project and select the "*ClassFile*" module.

Documentation generation

Overview

The use of the document template is the same whether you use the document template in a test project or after having installed the module.

For further information, see the *Objecteering/Documentation* user guide.

Creating a document work product

To be able to generate documentation using the new document template, it is first necessary to define the module that the product must use (Figure 5-4), and then to select the new document template in the product's dialog box (Figure 5-5).

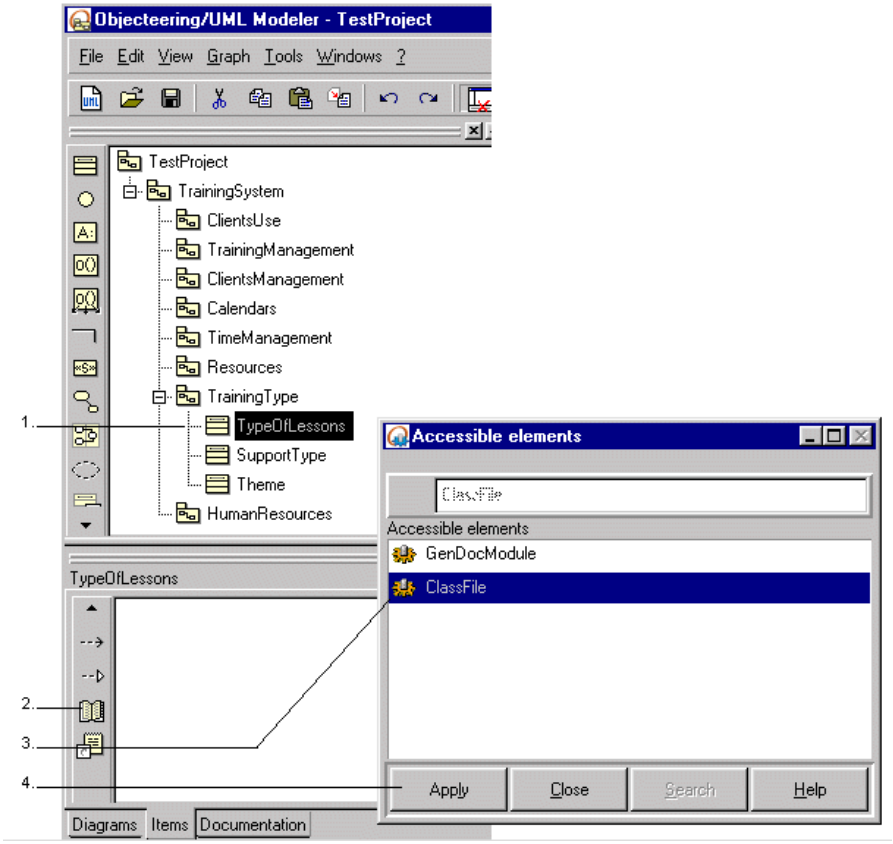


Figure 5-4. Creating a document work product

Steps:

1 - Select the "*TypeOfLessons*" class in the explorer.

2 - Click on the  "*Create a generation work product*" icon in the "*Items*" tab of the properties editor.

3 - Select the "*ClassFile*" module.

4 - Click on "*Apply".*

5 - Continue with the steps represented in Figure 5-5.

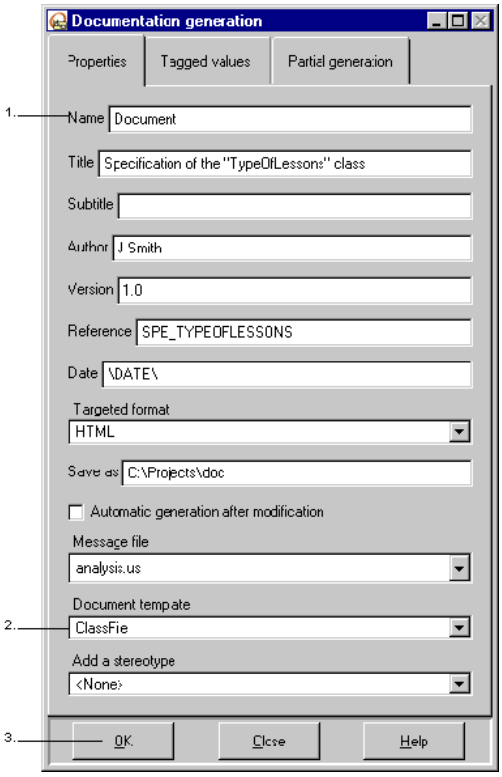


Figure 5-5. Document work product dialog box

Steps:

1 - Enter the values for the document work product.

2 - Select the "*ClassFile*" document template.

3 - Confirm.

Generating the document

The document work product can now be generated like any other work product (Figure 5-6) and the result can be visualized (Figure 5-7).

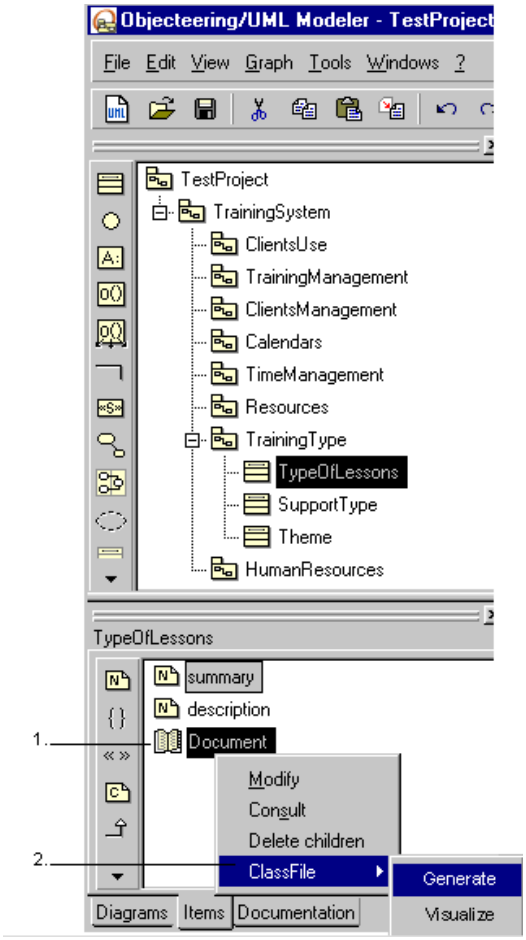


Figure 5-6. Generating the document

Steps:

1 - Select the document work product in the "*Items*" tab of the properties editor.

2 - Run the "*Generate*" command.

Visualizing the generated document

The generated document can be visualized by selecting the "*Visualize*" menu of the document work product (Figure 5-7).

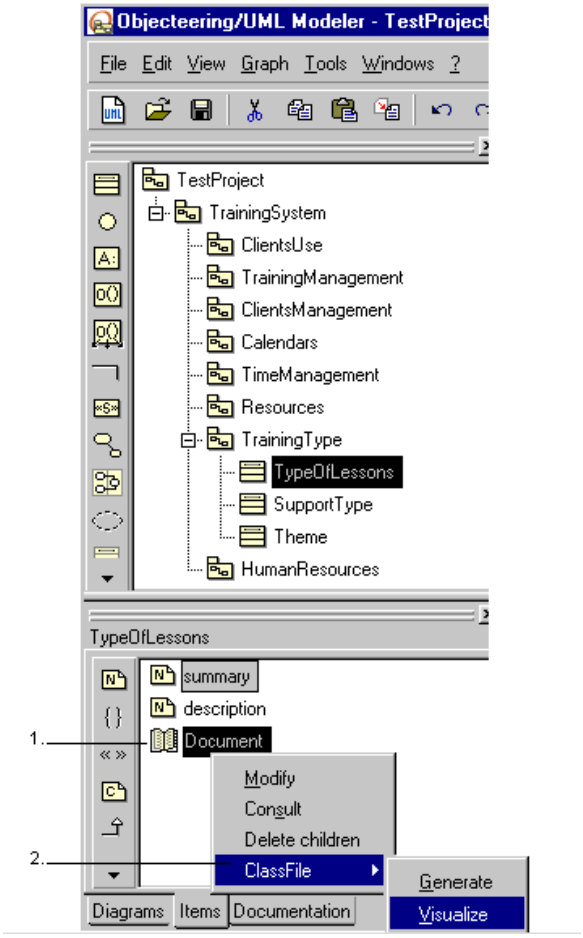


Figure 5-7. Visualizing the document

Steps:

1 - Select the document work product in the "*Items*" tab of the properties editor.

2 - Run the "*Visualize*" command.

Index