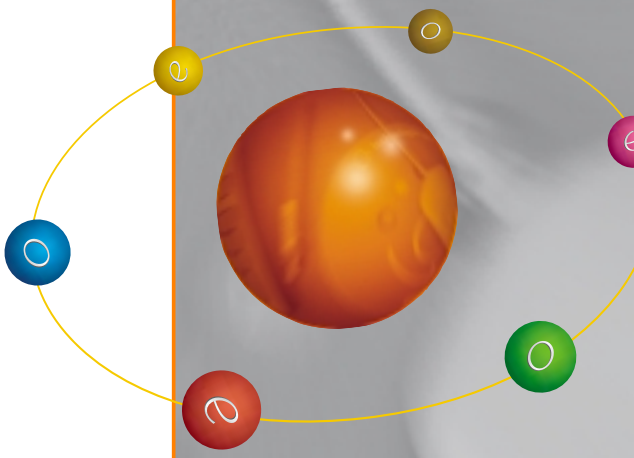
Objecteering/UML

Objecteering/CORBA 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

Contents

Chapter 1: Introduction

Overview of Objecteering/CORBA

Introduction

Welcome to *Objecteering/CORBA*!

Objecteering/CORBA consists of 2 modules: *UML Profile for CORBA* and *CORBA* itself. The *UML Profile for CORBA* module is not subject to a license, and is used to model in accordance with the "*UML Profile for CORBA V1.0*" specification. The *CORBA* module is used to generate IDL in accordance with the CORBA 2.3 specification.

The *Objecteering/CORBA* modules allow the modeling, generation and compilation of an IDL description from an Objecteering/UML model.

Functions

The *Objecteering/CORBA* module groups together the following features:

- IDL code generation
- generated code compilation

By working in conjunction with the reverse modules (C++ or Java) available in the Objecteering/UML range, the user can obtain client and server executables through the following operations:

- reverse engineering C++ or Java skeletons of implementation classes generated by the ORB
- entering code for implementation class operations previously reversed on the server side
- generating C++ or Java code
- generating the production line and compiling

Structure of the Objecteering/CORBA user guide

The *Objecteering/CORBA* user guide is intended for users of the *Objecteering/CORBA* modules. It guides you through the modeling and the realization of an application, and constitutes a reference manual, which will help you understand and use dedicated stereotypes, tagged values and notes. This user guide is divided into the following chapters:

- *Chapter 2*: This chapter describes how to prepare for working with the *Objecteering/CORBA* modules.
- *Chapter 3*: This chapter provides the user with "*First Steps*", which demonstrate how to obtain an IDL file.
- *Chapter 4*: This chapter details the *UML Profile for CORBA* module and provides information on dedicated stereotypes, tagged values and notes.
- *Chapter 5*: This chapter describes IDL code generation.
- *Chapter 6*: This chapter explains how to generate IDL without opening Objecteering/UML.

Glossary

- *IDL generation work product*: object that can be created on a package and which possesses *Objecteering/CORBA* code generation and compilation features.
- *External edition*: operation which allows the entry of notes with an editor other than the Objecteering/UML text editor. The text entered between pre-positioned markers is re-incorporated into the Objecteering/UML repository when the editing is completed.

Chapter 2: Working with the Objecteering/CORBA module

Installation information

Prerequisites

The *Objecteering/CORBA* 2.0 module requires that Objecteering/UML already be installed, and that the OBJING_PATH environment variable be positioned.

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

Installation directories

Module data can be found in the $OBJING_PATH/modules/CorbaModule and $OBJING_PATH/modules/CorbaProfileModule directories, which contain the following elements respectively:

- res and FirstSteps
- res and CORBA

Using the module

In order to use the *Objecteering/CORBA* module, you simply have to select the module for your UML modeling project (for further details on selecting modules in UML modeling projects, please refer to the "*Using the Objecteering/CORBA module in your UML modeling* " section in the current chapter of this user guide).

Using the Objecteering/CORBA module in your UML modeling project

Introduction

Before the *Objecteering/CORBA* module can be used, the following steps must be carried out:

1 - Create a UML modeling project.

2 - Select the module.

Creating a working UML modeling project

For information on how to create a UML modeling project, please refer to the "*Creating or opening a UML modeling project*" section in chapter 3 of the *Objecteering/UML Modeler* user guide.

Selecting the CORBAModule module for the new UML modeling project

Launch the *Objecteering/UML Modeler* editor on your newly-created UML modeling project. The  "*UML modeling project modules*" icon launches the window used to select the module (as shown in Figure 2-2).

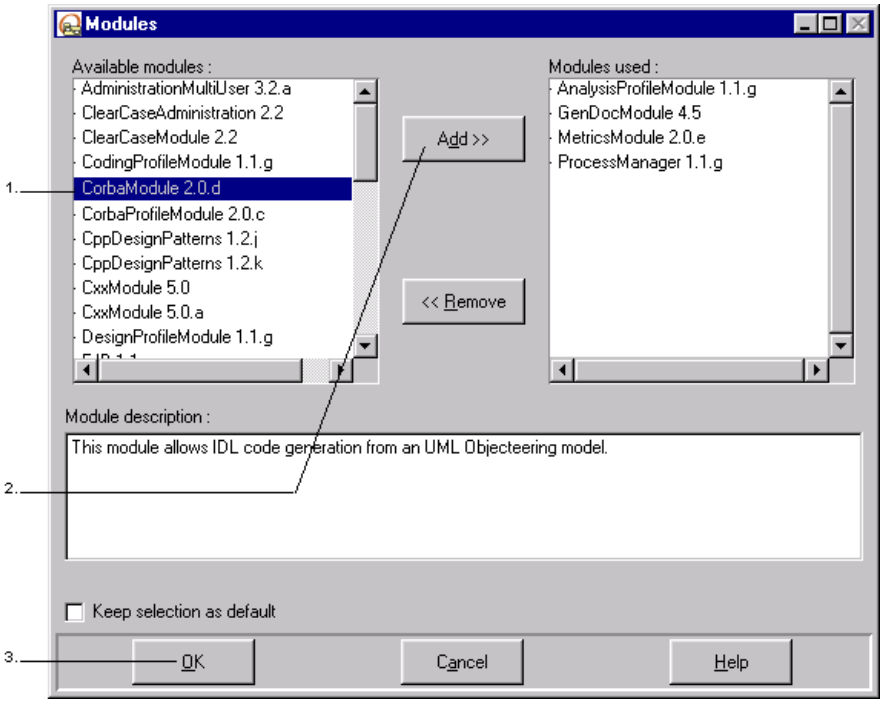


Figure 2-1. Selecting the CORBA module

Steps:

1 - Select the "*CORBA*" module from the available modules list on the left-hand side of the screen.

2 - Click on the "*Add*" button. The "*CORBA*" module then appears in the right-hand "*Modules used*" column.

3 - Click on "*OK*" to confirm. If the "*Keep selection as default*" box is checked, the "*CORBA*" module will automatically be available during future Objecteering/UML sessions.

For further information on this operation, please refer to the "*Selecting modules in the current UML modeling project*" section in chapter 3 of the *Objecteering/Introduction* user guide.

Chapter 3: First Steps

Overview of First Steps

Introduction

By following an example of a UML modeling project, you will discover step by step the different features of the *Objecteering/CORBA* module.

Sources

The example used is a climate control system application, extracted from "*Advanced CORBA Programming with C++*"*, by Michi Henning and Steve Vinoski, Addison-Wesley*.

Initializing the First Steps UML modeling project

To initialize your First Steps UML modeling project, follow the steps described in the "*Using the Objecteering/CORBA module in your UML modeling project*" section in chapter 2 of this user guide.

Getting the example model

Figure 3-1 shows the steps which should be carried out, in order to import the "*FirstSteps*" UML modeling project.

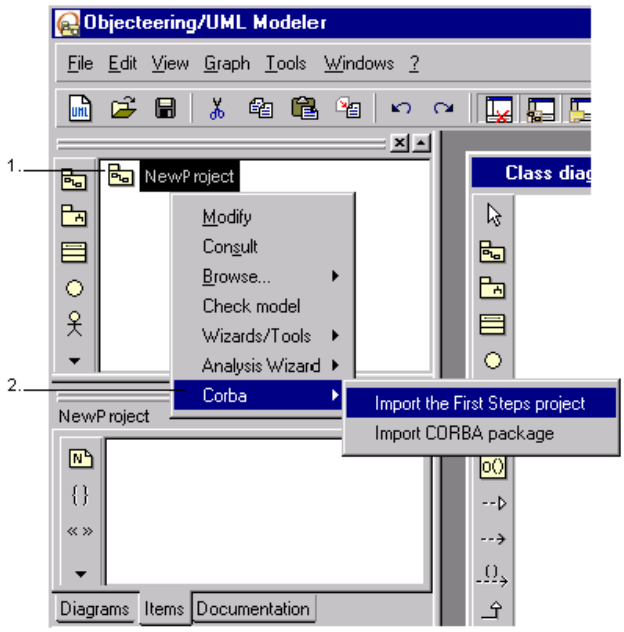


Figure 3-1. Importing the "*FirstSteps*" UML modeling project

Steps:

1 - Select the UML model root package using the right mouse button.

2 - Select the "*Corba/Import the First Steps project*" commands from the context menu which appears.

Creating an IDL generation work product

Overview

In Objecteering/UML, the IDL generation work product provides commands for generating code and compiling. It can also be used to manage files which have been produced. Thus, if you destroy the generation work product, you will also destroy the generated files.

Creating an IDL generation work product

Using this example (shown in Figure 3-2), we are going to create an IDL generation work product for the "*NewProject*" package.

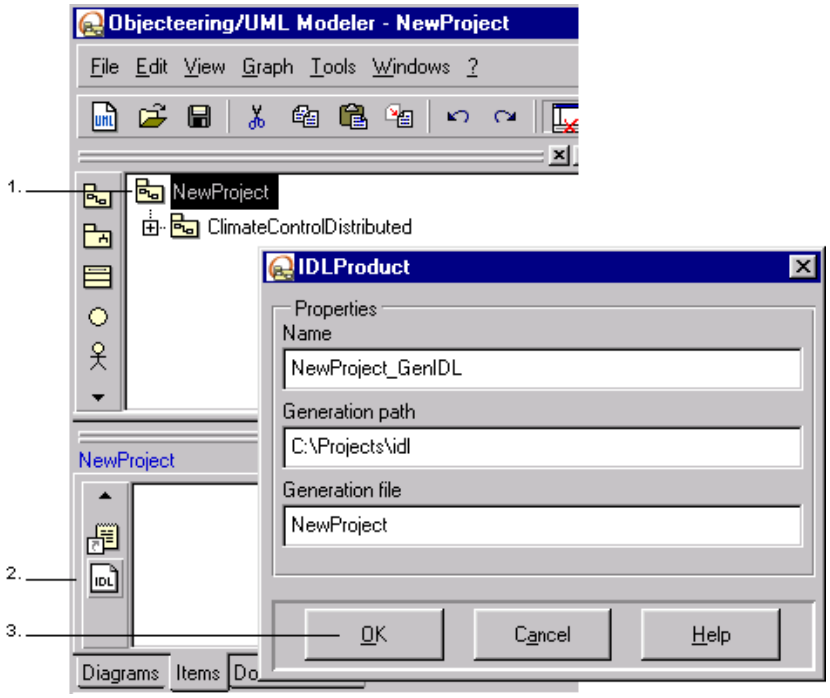


Figure 3-2. Creating an IDL generation work product

Steps:

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

2 - Click on the  "*IDL generation*" icon in the "*Items*" tab of the properties editor.

3 - Press "*Return*" to confirm the details in the dialog box which then opens.

Description of an IDL generation work product

Figure 3-3 shows the IDL generation work product dialog box.

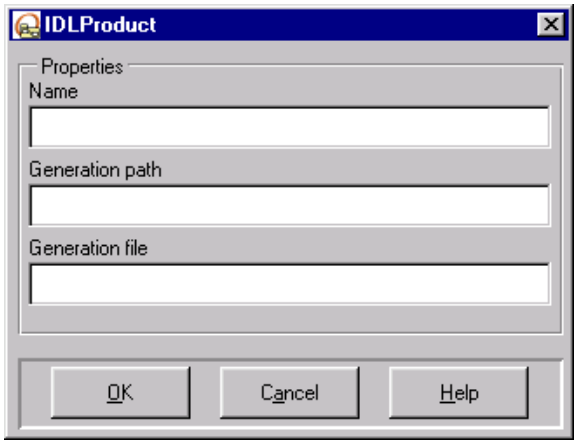


Figure 3-3. The IDL generation work product dialog box

Key:

- "*Name*": This entry field indicates the name of the generation work product as it appears in the "*Items*" tab of the Objecteering/UML properties editor.
- "*Generation path*": This entry field indicates the positioning of the IDL file generation directory.
- "*Generation file*": This entry field indicates the name of the generated file. The ".idl" extension is automatically added.

IDL generation work product menus

Overview

The IDL generation work product has commands for generating IDL code and compiling the generated files (see Figure 3-4).

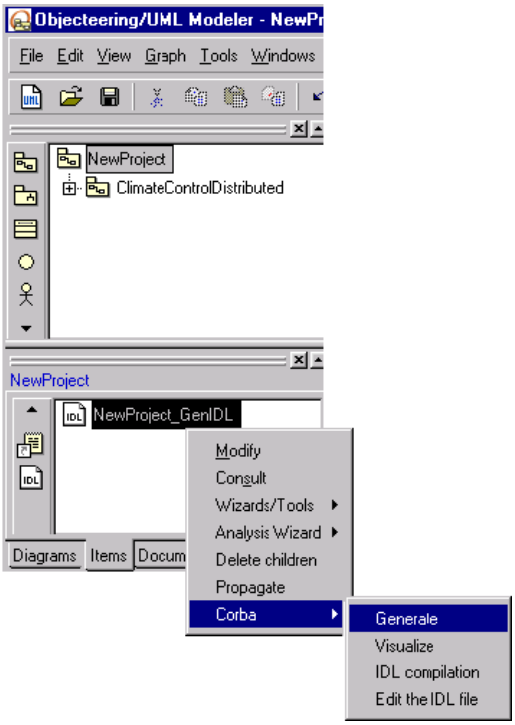


Figure 3-4. Generation work product services

All the commands used to edit and generate IDL are grouped together in the "*Corba*" menu. We will look at these commands one after another.

Contents of the services

The ... menu	is used to ...
Generate	generate the IDL code for root package components.
Visualize	visualize the generated IDL code.
IDL compilation	run the IDL compilation of the generated file. The "*Command for invoking IDL compilation*" parameter in the "*IDL compilation*" module parameterization item is used to specify the IDL compiler.
Edit the IDL file	edit the generated code. The editor used is the one defined by the "*External editor*" parameter in the CORBA generation module.

Generating IDL code

Launching IDL code generation

To launch IDL code generation, simply click on the generation work product using the right mouse button, and run the "*Corba/Generate*" commands from the context menu which appears.

An *.idl* file is generated for the package referenced by the IDL work product on which the "*Generate*" command is run. The code of inner packages is incorporated in the same file. However, work products can also be created for inner packages, and separate idl generated for them. In this case, the *.idl* file for the enclosing package contains the necessary include clause.

Visualizing the generated IDL code

Generated IDL code can also be visualized (as shown in Figure 3-5). This command can only be launched on a work product to which an IDL file is associated, that is to say, on a work product of a package for which code has been generated.

To visualize generated IDL code, simply click on the generation work product belonging to the "*ClimateControlDistributed*" class using the right mouse button, and run the "*Corba/Visualize*" commands from the context menu which appears. This command opens a window containing the generated IDL code. It is not possible to modify the code directly in this window. You can, however, easily modify notes on certain model elements, by clicking on the blue code sections.

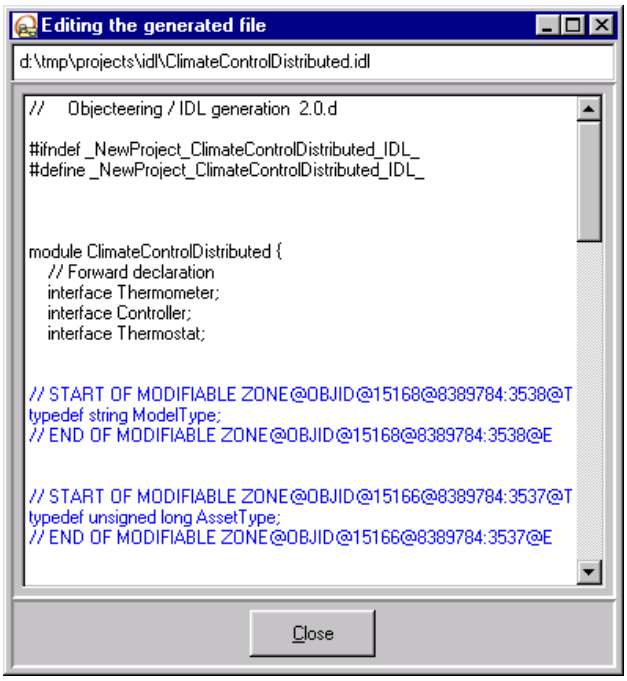


Figure 3-5. Window displaying the IDL code of the "*ClimateControlDistributed*" package

Editing the generated code

The generated IDL code for a package can be edited using the editor chosen, as shown in Figure 3-6 (see the "*External editor*" parameter in the "External edition" group). Zones represented between markers can be modified. Modifications are directly incorporated into the model when the editor is closed.

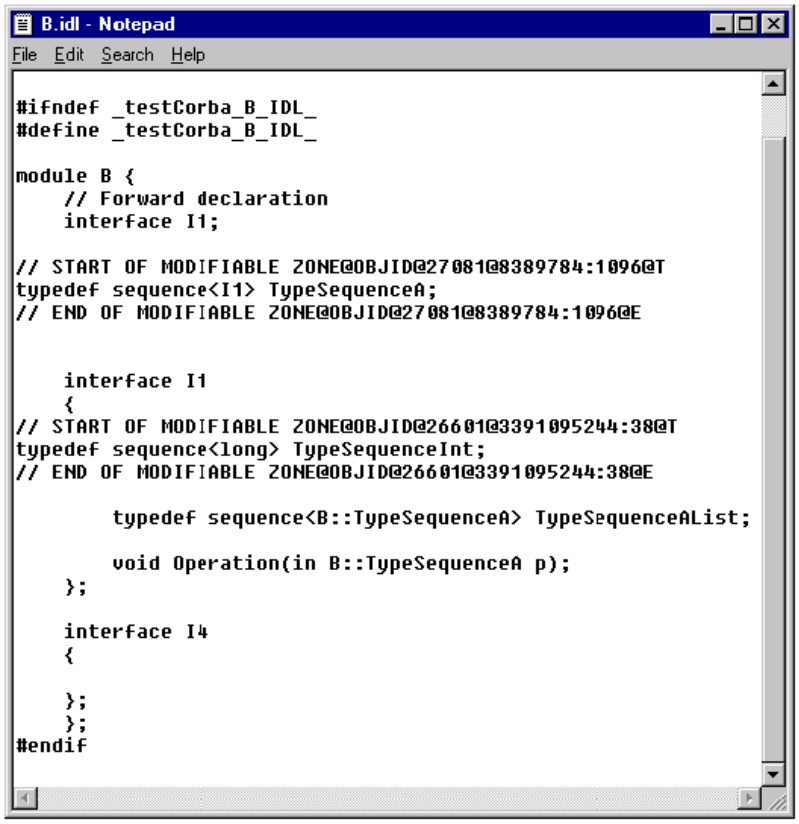


Figure 3-6. Editing the generated code using an external editor

Compiling

Parameterizing the IDL compilation command

Once the file has been generated, the user may call the IDL compiler, in order to check IDL syntax and generate C++ or Java classes from IDL interfaces generated by Objecteering/UML. Options available when calling the compiler (for example, the interface implementation mode or generation paths) must be specified in the "*Command for invoking IDL compilation*" parameter entry field.

Example: idl –B ClimateControlDistributed.idl (as shown in Figure 3-7).

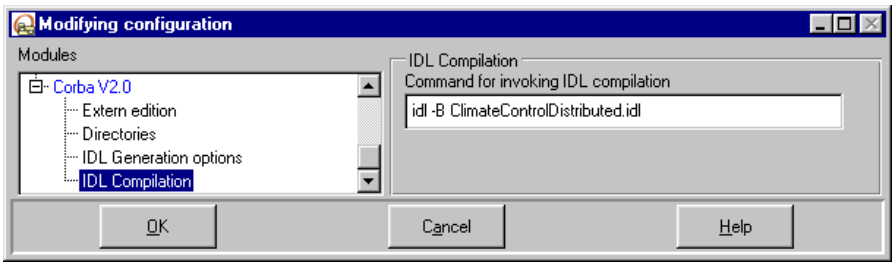


Figure 3-7. IDL compilation command parameter

Triggering compilation

To compile, simply run the "*Corba/IDL Compilation*" commands from the generation work product context menu on the "*ClimateControlDistributed*" package (as shown in Figure 3-8).

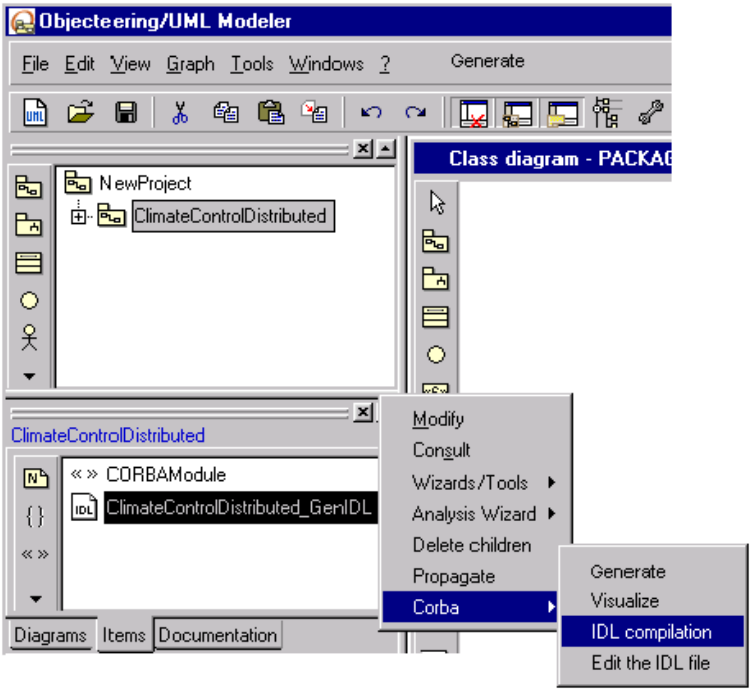


Figure 3-8. Running IDL compilation

Steps:

1 - Select the "*ClimateControlDistributed*" generation work product in the "*Items*" tab of the properties editor, by clicking on the right mouse button.

2 - Run the "*Corba/IDL compilation*" commands from the context menu which appears.

Visualizing the result of the compilation

Where there are no compilation errors, a window similar to that shown in Figure 3-9 is displayed, to indicate that IDL compilation command has been correctly run.

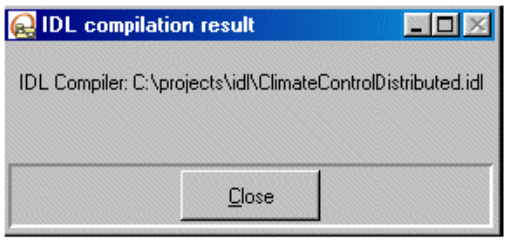


Figure 3-9. Result of error-free compilation

Where errors have occurred, a window like the one shown in Figure 3-10 is displayed, indicating those lines which include syntax errors and an indication with regard to the error in question.

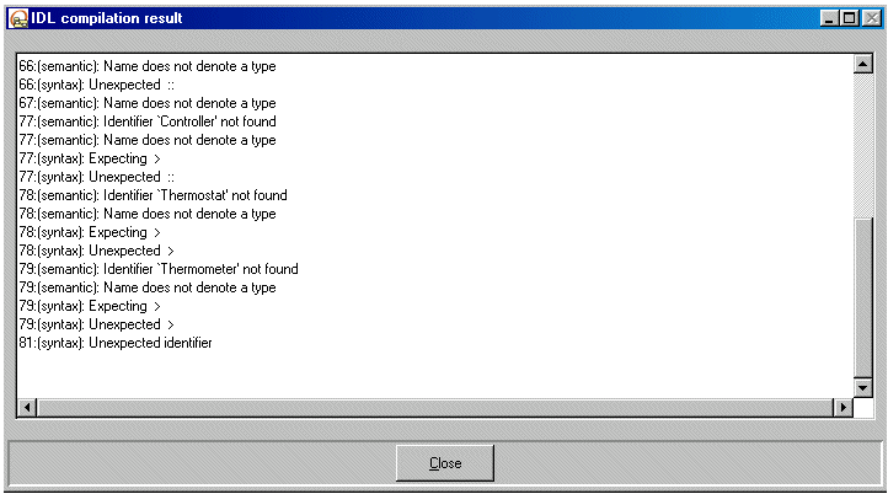


Figure 3-10. Result of compilation indicating syntax errors

The information presented in this screen differs according to the IDL compiler used.

Chapter 4: The UML Profile for CORBA module

Overview of the UML Profile for CORBA module

The "*UML Profile for CORBA*" module defines all the stereotypes, tagged values and notes available on certain metaclasses, and a "*CORBA*" package containing some basic classes, as detailed in the "*UML Profile for CORBA specification*" document.

This module is free of charge and allows the user to model his application according to OMG specifications. The *CorbaModule* IDL generator takes into account most of the afore-mentioned notations.

The CORBA package

The CORBA package contains the following classes stereotyped <<CORBAPrimitive>>:

- short
- long
- long long
- double
- unsigned short
- unsigned long
- unsigned long long
- any
- Boolean
- String
- octet
- void
- char
- wchar
- float
- wstring
- typecode
- native

These classes are required when modeling a "*typedef*" or "sequence" etc. The following examples will provide further details.

The CORBA package also contains a template class, "*fixed*", which has two template parameters. This class is used to model the instantiation of CORBA fixed template.

Examples

The "*UML Profile for CORBA specification*" indicates that in order to obtain the following idl code:

```
typedef sequence<short> mySeqShort;
```

you should create a class named "*mySeqShort*" and stereotyped <<CORBASequence>>, and then create an association (with multiplicity of 1) from this class to the "*CORBA::short*" class.

The "*UML Profile for CORBA specification*" indicates that in order to obtain the following idl code:

```
typedef unsigned long ulong;
```

you should create a class named "*ulong*" and stereotyped <<CORBATypedef>>, and then make this class specialize the "*CORBA::unsigned long long*" class.

In Objecteering/UML, this association and this generalization require that "*short*" and "*unsigned long long*" be classes. To distinguish them from ordinary classes, they are stereotyped <<CORBAPrimitive>>.

Stereotypes

The table below provides details on those stereotypes which exist in the *Objecteering/UML Profile for CORBA* module.

The ... stereotype	on the ... metaclass	is used to ...
CORBAModel	Package	generate a module
CORBAInterface	Class	generate an interface
CORBAEnum	Class	generate an enum
CORBAStruct	Class	generate a struct
CORBAUnion	Class	generate a union
CORBAException	Class	generate an exception
CORBASequence	Class	generate a typedef sequence <...>
CORBAArray	Class	generate a typedef type Name [...]
CORBATypedef	Class	generate a typedef type typeName
CORBAConstants	Class	generate constants in the module
CORBAAnonymousSequence	Class	generate the sequence <...> used in struct/exception
CORBAAnonymousArray	Class	generate the array used in struct/exception
CORBAFixed	Class	instantiate the fixed
CORBAValue	Class	generate the value*

The ... stereotype	on the ... metaclass	is used to ...
CORBACustomValue	Class	generate the CustomValue*
CORBABoxedValue	Class	generate the BoxedValue*
CORBAReadonlyEnd	AssociationEnd	generate the read only attribute
CORBAReadonly	Attribute	generate the read only attribute
CORBAConstant	Attribute	generate a constant in a UserDefinedType
CORBAOneway	Operation	generate a one way operation
CORBAValueFactory	Operation	generate a ValueFactory*
CORBAValueSupports	Generalization	generate a ValueSupports*
CORBATruncatable	Generalization	generate a truncatable*

*Not taken into account for IDL generation in the current version of the module.

Tagged values

The table below provides details on those tagged values which exist in the *Objecteering/UML Profile for CORBA* module.

The ... tagged value	on the ... metaclass	is used to ...
CORBAIDLOrder	Package	generate the element order in the model.
CORBATypeId	Package	choose a repository ID.
CORBATypePrefix	Package	choose a repository ID prefix.
CORBATypeId	Class	choose a repository ID
CORBATypePrefix	Class	choose a repository ID prefix
CORBABind	Class	give the parameters used to instantiate fixed. .
CORBAName	ModelElement	represent the generated name for the element. .
CORBACase	Parameter, AssociationEnd, Attribute	generate the case in Union.
CORBAArray	Parameter, AssociationEnd, Attribute	generate the use array for the attribute of n multiplicity.
CORBATypeName	Parameter, AssociationEnd, Attribute	suggest the name in typedef for the attribute of * multiplicity.
CORBABind	Parameter, AssociationEnd	give the parameters used to instantiate an association end or parameter, which has fixed as opposite/parameter type.

The ... tagged value	on the ... metaclass	is used to ...
CORBAIDLOrder	Feature, DataType, Enumeration	generate the element order in the user defined type.
CORBANocode	Feature	generate no IDL code. .
CORBAContext	Operation	combine with "int", "float", "char" in mapping to the corresponding C.ORBA basic types.
CORBAUnsigned	Attribute, Parameter	combine with "int", "float", "char" in mapping to the corresponding CORBA basic types. .
CORBALong	Attribute, Parameter	combine with "int", "float", "char" in mapping to the corresponding CORBA basic types. .
CORBALongLong	Attribute, Parameter	combine with "int", "float", "char" in mapping to the corresponding CORBA basic types. .
CORBAOctet	Attribute, Parameter	combine with "int", "float", "char" in mapping to the corresponding CORBA basic types. .
CORBAAny	Attribute, Parameter	combine with "int", "float", "char" in mapping to the corresponding CORBA basic types. .

Notes

The table below provides details on those notes which exist in the *Objecteering/UML Profile for CORBA* module.

The ... note	on the ... metaclass	is used to ...
CORBAInheritance	Class	insert free text as parent class.
CORBAHeader	Package	insert free text at the beginning of the idl file. .
CORBAIDL	Datatype	insert free text associated with a UML type. .

Mapping UML types into IDL types

The mapping of Objecteering/UML base types in IDL is shown in the following table:

Objecteering/UML type	Tagged value	IDL type
Integer		short
Integer	{CORBAUnsigned}	unsigned short
Integer	{CORBALong}	long
Integer	{CORBALongLong}	long long
Integer	{CORBAUnsigned}, {CORBALong}	unsigned long
Integer	{CORBAUnsigned}, {CORBALongLong}	unsigned long long
real		float
real	{CORBALong}	double
Char		char
Boolean		boolean
Integer	{CORBAOctet}	octet
Integer	{CORBAAny}	any
String		string

Importing the CORBA package

To import the "*CORBA*" package, carry out the steps illustrated below (Figure 4-1).

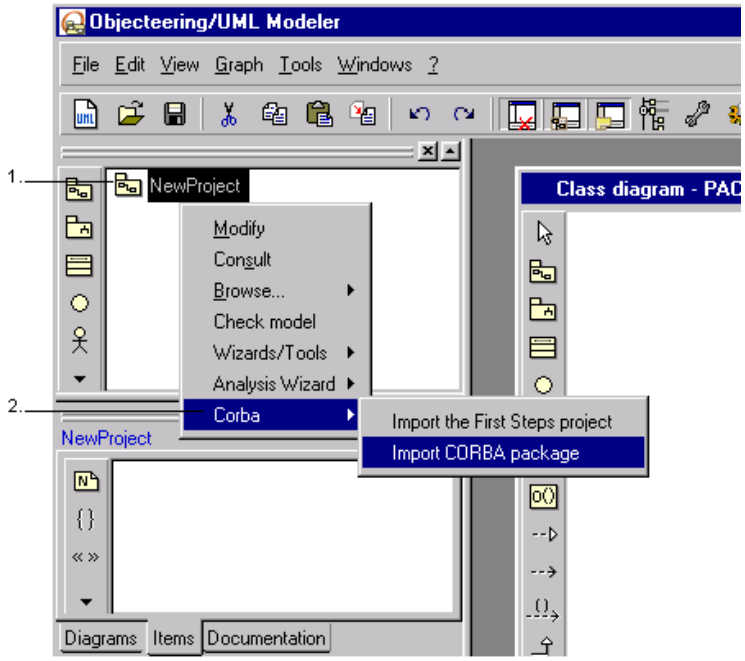


Figure 4-1. Importing the "*CORBA*" package

Steps:

1 - Select the UML model root package using the right-mouse button.

2 - Select the "*Corba*/*Import CORBA package*" command from the context menu which then appears.

Chapter 5: Code generation

Overview of code generation

Code generation and model consistency checks

Code may be generated from the UML model regardless of whether consistency checks are active or inactive. However, when generation is launched, a message informs the user that he is in the process of generating code on a model which may potentially not conform to the UML modeling rules checked by Objecteering/UML (as shown in Figure 5-1).

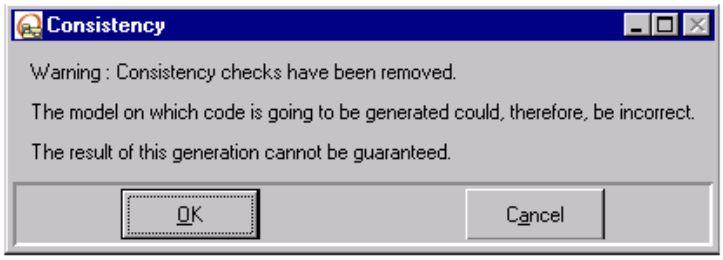


Figure 5-1. Message informing the user that consistency checks have been removed

Note: It should be noted that code generation in command line mode is assured, whatever the state of the consistency checks at the time of code generation.

The generation work product

Before generating IDL code, an IDL generation work product must be created (see Figure 5-2). This object is created in a package.

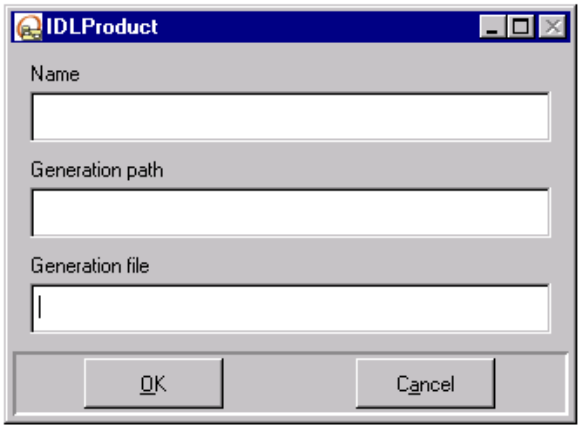


Figure 5-2. Dialog box for an IDL generation work product

An IDL work product has the following properties:

The ... property	**is used to ...**
Name	display the name of the work product.
Generation path	display the directory for generating the .idl files
Generation file	display the name of the file generated. The ".idl" extension is systematically concatenated with the value of this entry field.

Generating IDL elements

Generating an IDL module

An IDL module is deduced from a UML package. To allow IDL generation, the package must be stereotyped <<CORBAModule>>. An empty UML package, that is to say a package without classes, type definitions or enumerates, does not give rise to IDL generation for module definition.

Embedded UML packages are mapped in the form of nested IDL module definitions.

Generating an IDL interface

An IDL interface is deduced from a UML class with public visibility. To allow IDL generation, the class must be stereotyped <<CORBAInterface>>>>. An IDL interface cannot be embedded in an IDL interface. Furthermore, UML classes embedded in a class must include annotations, which allow them to be mapped in the form of IDL exceptions or IDL structure.

Generating an IDL structure

An IDL structure is deduced from a UML class with public visibility and is stereotyped <<CORBAStruct>>. The UML class must not include operations.

Generating an enumerate

An IDL enumerate can be directly deduced from a UML enumerate.

An IDL enumerate can also be modeled as a class, stereotyped <<CORBAEnum>>. The enum item is modeled by an attribute.

Generating an IDL constant

An IDL constant is modeled by an attribute stereotyped <<CORBAConstant>>, with the constant value expression represented by the attribute's initial value expression. A class' attribute is generated as a constant, within the scope of the IDL interface. For constants defined within a module, the attribute must be contained in a special class named *Constants* and stereotyped <<CORBAConstants>>.

Generating an IDL union

An IDL union is defined by a "*CORBAUnion*" class. This class must have one and only one attribute or association. Its name is the same name as the class, with the "_switch" suffix added. The other attribute or associationEnd must be annotated with the {CORBACase(label)} tagged value.

Generating an IDL type definition

An IDL type definition can be modeled in two ways:

- through a UML type to which a "CORBAIDL" note is associated
- through a class stereotyped <<CORBATypedef>>

The definition of the type must be entered in its entirety by the user in the "CORBAIDL" note. Type definition which depends on another class must be modeled through generalization. The dependency decides the generation order.

Generating an IDL exception

An exception is modeled in the form of a UML class stereotyped <<CORBAException>>. This class, which represents an exception, can be embedded in an IDL interface. For a class stereotyped <<CORBAException>>, it must not:

- be in a generalization graph
- have associations with other classes
- have operations

To specify that an operation can raise an exception, a use link from the operation to the class representing the exception must be modeled. This class is stereotyped <<CORBAException>>. An operation can raise several exceptions.

Generating an asynchronous IDL operation

A UML operation is transformed into an asynchronous operation when the operation is stereotyped <<CORBAOneway>>. For this type of operation:

- the parameters must all be in "In" mode
- the operation must not have return parameters
- the operation must not raise exceptions

Compatibility

Version 2.0 of the *Objecteering/CORBA* module does not take into account the following notes and tagged values defined in its previous versions.

The ... note	on the ... metaclass
Corba::moduleMember	Package
Corba::interfaceMember	Class
Corba::structureMember	Class
Corba::exceptionMember	Class
Corba::idl	Class

The ... tagged value	on the ... metaclass
Corba::root	Package
Corba::GlobalVariable	Package

Chapter 6: Calling module on-line commands

Calling on-line commands

Overview

Module commands which do not require an interface can be launched through a command line, using the *objingcl* delivered with Objecteering/UML.

Calling commands

An on-line command is called using the following instruction:

```
objingcl-prj <project_name>
-db base
-mdl CorbaModule
-cmd <command_Name>
<metaclass>:<object_name>
```

Commands which can be invoked

The ... command	On the ... metaclass	is used to ...
generate	Idl Product	generate code

Index