OpenAmeos

Advanced Modeling Concepts
Advanced Modeling Concepts

Trademarks

Aonix and its logo, Software through Pictures, StP, RAVEN, and ObjectAda are either trademarks or registered trademarks of Aonix. All rights reserved. Note that this product includes software developed by the Apache Software Foundation (www.apache.org).

HP, HP-UX, and SoftBench are trademarks of Hewlett-Packard Company. Sun and Solaris are registered trademarks of Sun Microsystems, Inc. SPARC is a registered trademark of SPARC International, Inc. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. Windows, Windows NT, Windows 2000, and Windows XP are either trademarks or registered trademarks of Microsoft Corporation in the United States and other countries. Adobe, Acrobat, the Acrobat logo, and PostScript are trademarks of Adobe Systems, Inc. Sybase, the Sybase logo, and Sybase products are either trademarks or registered trademarks of Sybase, Inc. DOORS is a registered trademark of Telelogic. Continuus and Continuus products are either trademarks or registered trademarks of Telelogic. Rational Rose and ClearCase are registered trademarks of Rational Software Corporation. SNiFF+ and SNiFF products are either trademarks or registered trademarks of Wind River Systems, Inc. Segue is a registered trademark of Segue Software, Inc. All other product and company names are either trademarks or registered trademarks of their respective companies.
Table of Contents

Chapter 1  An Overview of Ameos

Purpose of This Manual ..........................................................................................................................7
What is Covered Here ..............................................................................................................................7
What is Not Covered Here .......................................................................................................................7
About Ameos ...........................................................................................................................................7
The Ameos Desktop ..............................................................................................................................8
Ameos/UML Applications ......................................................................................................................8
The Repository .......................................................................................................................................9
ACD, MDA-Based Model Transformation ..........................................................................................10

Chapter 2  Ameos/UML Profile Support

Profile Editor .............................................................................................................................................12
How to Model Profiles in Ameos .........................................................................................................13
Using Predefined Profiles ......................................................................................................................13
Reusing Profile Definitions ...................................................................................................................14
Default Profiles for a New System .......................................................................................................14
Usage Note ...........................................................................................................................................14
Predefined Ameos/UML Stereotypes ....................................................................................................14

Chapter 3  The Script Manager

Script Manager Layout ............................................................................................................................18
Scripts Pane ...........................................................................................................................................19
Parameters Pane ...................................................................................................................................20
Using the Script Manager ......................................................................................................................20
Running a Script from the Script Manager .........................................................................................20
Terminating a Script .............................................................................................................................21
Specifying Values for Parameters .......................................................................................................21
Copying a Script ....................................................................................................................................21
Creating a Script ....................................................................................................................................22
Editing a Script .......................................................................................................................................23
Deleting a Script .....................................................................................................................................23
Directory Structure for Scripts ...........................................................................................................23
Adding a Site Scripts Location ..............................................................................................................24
Preferences for the Script Manager .....................................................................................................24
Chapter 4  Report Generation

Generating Reports with the Script Manager .................................................................26
  UML Reports ...........................................................................................................26
  HTML Reports .......................................................................................................27

Generating Reports with the Report Maker Editor ...................................................28
  Overview of RME Usage .......................................................................................28
  Creating RME Diagrams .......................................................................................29
  Generating the Report ..........................................................................................30
  RME Objects and Annotations ............................................................................32
  Additional Information ..........................................................................................35

Chapter 5  Reverse Engineering

Parsing Java Source ....................................................................................................36

Parsing C++ Source ..................................................................................................38
  Source File Suffixes ...............................................................................................39
  Microsoft Extensions ............................................................................................39
  Using the Makefile Reader ....................................................................................40
  Using Parser Preprocessor Options .......................................................................40
  Extracting Comments ...........................................................................................41
  Increasing the Speed of the Parser .......................................................................43

Parsing Ada 95 Source ..............................................................................................44

Other Parsing Topics ................................................................................................45
  Deciding Which Files to Parse ..............................................................................45
  Mixing Languages ..................................................................................................45
  Using Incremental Mode ......................................................................................45
  Parsing Source Code Containing Embedded Code ..............................................46
  Common Parsing Problems ..................................................................................47
  Checking Semantic Model Locks (Ada 95 and C++) ............................................48
  Checking Semantic Model Consistency (Ada 95 and C++) ....................................48

Generating a Model fromParsed Files ......................................................................48
  Retaining User Modifications if a Class Exists ......................................................48

Model Elements Generated by Reverse Engineering .............................................49
  Package Diagrams ..................................................................................................49
  Class Diagrams ......................................................................................................50
  Tabular Class Content ............................................................................................55
  Annotations ............................................................................................................57

Updating a Class from the Editors ..........................................................................58

Chapter 6  Importing and Exporting XMI With Ameos/UML

Prerequisites ...............................................................................................................59
  Compatibility with Language Specifications .........................................................59
Compatibility with Previous Code Generators ................................................................. 59
Exporting XMI ....................................................................................................................... 60
Importing XMI ....................................................................................................................... 60
Class Diagrams ..................................................................................................................... 61
Sequence diagrams .............................................................................................................. 62
State Diagrams ..................................................................................................................... 62
Other import capabilities .................................................................................................... 63

Chapter 7  Baselining and Version Control

Creating, Restoring, and Comparing Baseline Systems .......................................................... 64
A Typical Scenario .................................................................................................................. 64
Creating a Baseline ............................................................................................................... 65
Restoring a Baseline .............................................................................................................. 66
Comparing Baselines ........................................................................................................... 67
Using Version Control ........................................................................................................ 68
Using External CM Systems ................................................................................................. 68
CVS ...................................................................................................................................... 68
Other Integrations ................................................................................................................. 72

Chapter 8  Model Management

Using Ameos Model Management—An Overview ................................................................. 74
About Subsystems ................................................................................................................ 74
The Model Management Process .......................................................................................... 75
Using the Model Management Tree Browser ........................................................................ 75
Defining a Subsystem ........................................................................................................... 76
Representing the Subsystem ................................................................................................ 77
Assigning Model Elements to a Subsystem .......................................................................... 77
Propagating Subsystem Membership .................................................................................... 78
Assigning Diagrams to Subsystems ....................................................................................... 78
Managing Subsystems .......................................................................................................... 81
Checking Semantics for Subsystems .................................................................................... 82
Creating a Private Workspace or Baselining a Subsystem .................................................... 83
Locking and Unlocking a Subsystem ..................................................................................... 84
Restoring a Subsystem ......................................................................................................... 84
Reconfiguring External Model Elements ............................................................................. 85
Known Issues and Limitations ............................................................................................. 86

Chapter 9  Third-Party Integrations

DOORS ................................................................................................................................. 87
How DOORS Stores Ameos Objects .................................................................................... 87
Enabling the Ameos/DOORS Integration ............................................................................ 88
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the Ameos/DOORS Integration</td>
<td>89</td>
</tr>
<tr>
<td>DOORS Limitations</td>
<td>90</td>
</tr>
<tr>
<td>Integrating SNiFF+ or Eclipse with Ameos/UML</td>
<td>90</td>
</tr>
<tr>
<td>Setting Up Integrations</td>
<td>91</td>
</tr>
<tr>
<td>Navigating to SNiFF+</td>
<td>91</td>
</tr>
<tr>
<td>Navigating to Eclipse</td>
<td>92</td>
</tr>
<tr>
<td>Navigating to Ameos/UML</td>
<td>92</td>
</tr>
</tbody>
</table>

**Chapter 10  Miscellaneous Topics**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Renaming</td>
<td>93</td>
</tr>
<tr>
<td>Objects and Object References</td>
<td>93</td>
</tr>
<tr>
<td>Renaming an Object Systemwide</td>
<td>95</td>
</tr>
<tr>
<td>Local Rename</td>
<td>97</td>
</tr>
<tr>
<td>Multiple Global Rename</td>
<td>97</td>
</tr>
</tbody>
</table>

**Lock Utilities**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>How Locking Works in Ameos</td>
<td>98</td>
</tr>
<tr>
<td>Using Locking Commands</td>
<td>99</td>
</tr>
<tr>
<td>Opening a Locked File</td>
<td>100</td>
</tr>
</tbody>
</table>
1 An Overview of Ameos

Purpose of This Manual

This is the Advanced Modeling Concepts document for Ameos/UML. It contains information about various aspects of modeling with Ameos/UML.

What is Covered Here

This manual provides information on:

- Profile support in Ameos/UML (Chapter 2 on page 11)
- The script manager (Chapter 3 on page 17) and report generation (Chapter 4 on page 26) facilities of Ameos
- Reverse engineering (Chapter 5 on page 36)
- Importing and exporting XMI (Chapter 6 on page 59)
- Baselining (Chapter 7 on page 64) and model management (Chapter 8 on page 74) facilities of Ameos
- Third-party integrations (Chapter 9 on page 87)
- Miscellaneous topics such as renaming and lock facilities (Chapter 10 on page 93)

What is Not Covered Here

This manual does not provide:

- Step-by-step instructions on using Ameos. For a tutorial on using the basic features of Ameos, refer to A Quick Tour of Ameos/UML.
- Command reference material (an item-by-item description of menu items and dialogs in the Ameos desktop and editor). For such information, refer to the online help (select Help > Help on Commands from the Ameos desktop or Ameos editors).
- Information on code generation. For such information, refer to the ACD Programming Guide and Using ACD Templates.
- Other reference information:
  - For information on administering Ameos, including how to manipulate the repository, refer to the Ameos Administration Guide.
  - For information on the Ameos object management system (OMS), a set of standards and functions for defining the interface between applications and the Ameos repository, refer to Object Management System.
  - For information on the Ameos query and reporting system (QRS), which allows you to use scripts written in QRL (query and reporting language), refer to Query and Reporting System.

About Ameos

Ameos is a next generation modeling tool with support for UML, UML 2.0 profiles and MDA based model-transformation.
Ameos is composed of the following parts:

- The Ameos desktop, which provides a foundation for all product user-interface elements, including diagram and table editors
- Ameos applications, product-specific components to support the modeling notation
- The Ameos shared repository
- Architecture Component Development (ACD), the Ameos model transformation engine

In addition, Ameos provides a model browser, plus tools for managing models (see Chapter 8 on page 74), importing to and exporting from programming environments (see Chapter 9 on page 87), as well as version control and baselining (see Chapter 7 on page 64).

**The Ameos Desktop**

The Ameos desktop is used for starting all Ameos product-specific editors and utilities. It also provides access to general Ameos features, including system administration and repository administration commands.

In addition, the Ameos desktop provides a model browser, plus access to tools for managing models (see Chapter 8 on page 74) importing to and exporting from programming environments (see Chapter 9 on page 87), as well as version control and baselining (see Chapter 7 on page 64).

**Find Window**

Part of the desktop is a find window that allows you to examine the contents of the repository in terms of the persistent data model (described in “The Repository” on page 9). Based on queries you construct, repository data appears in a table where you can sort the information, navigate from one object to another, and browse to related objects. For information about using the repository browser, refer to the online help (see "About the repository browser" on the Index page).

**Script Manager**

The script manager is also part of the desktop and the primary interface to the query and reporting system (QRS). QRS provides the query and reporting language (QRL), which processes data extracted from the repository. You can also use the QRL to format output text and graphics. For general information on printing diagrams and tables or sending formatted output to a file, see the online help ("About printing" on the Index tab). For detailed information on designing and generating reports, see the Creating Documents chapter of Query and Reporting System. For information on the script manager, refer to Chapter 3, “The Script Manager” on page 17.

**Ameos/UML Applications**

An Ameos/UML application comprises product-specific editors and utilities. Ameos applications allow you to create models of your system with the methods and notations supported by Ameos/UML. The information that is generated from the models is stored in the repository to ensure consistency across the entire project.

Ameos offers a set of graphical editors for modeling information, systems, and software. All include facilities for annotating objects, adding extensibility, and specifying information about a symbol in tabular format. For general information about Ameos editors, refer to the online help ("About Ameos editors" on the Index page).

**Ameos Systems and Projects**

Before you create or edit a model, you must set the current Ameos project and system.
A “project” is a directory that contains subdirectories for one or more Ameos systems. When creating a system, you can either create a new project directory or specify an existing one to hold the new system directory.

A “system” is the basic unit of organization for Ameos activity and has two main components:

- A directory of ASCII files for that system, called the “system directory,” in which each file represents a diagram, its annotations, or its tabular information as created with a particular editor (see “Ameos ASCII Files” below)
- A database storage area called the “repository,” which contains consolidated information about individual objects created by users with Ameos editors

When you save the contents of an Ameos editor, the contents are written both to an ASCII file in the system directory and to the system’s repository. Diagrams and tables must be syntactically correct for Ameos to store their contents in the repository. Otherwise, the data is stored only in the system’s ASCII files.

The Repository

When you save your model, Ameos writes information (application types) to the repository, a central pool of data that Ameos can access in a consistent manner. The repository stores application types as persistent objects. Each object in the repository is unique. An object contains all the information about a construct in the design domain.

The Ameos object management system (OMS) provides services for interactions between Ameos products and the repository.

The OMS comprises:

- The persistent data model (PDM)—The schema for all objects in the repository
- The application programming interface (API)—A library of functions for manipulating objects in the repository that can be linked with C and C++ programs
- OMS query language—A comprehensive language for querying the repository
- Type extension—Methods for extending the semantics of data stored in the repository

For details about the OMS, see Object Management System.

Windows and UNIX Users Sharing Data

Ameos supports Windows users and UNIX users sharing the same repository and the same data. However, there is the possibility for data conflict in this situation because Windows is case-insensitive and UNIX is case-sensitive.

Do not use case as a means for distinguishing the name of a new Ameos object in either the Windows environment or in a shared environment where Windows and UNIX users share a repository.

For example, if you have an object labeled “email,” do not distinguish a second object by labeling it “Email.” Ameos generates file names for certain objects based on the object name. Because Windows is case-insensitive, it considers the files “email” and “Email” as identical. The existence of these two files would result in data conflict.

Ameos ASCII Files

The Ameos application editors store the information for each diagram and its associated annotations and tabular content in an intermediate ASCII file format. Each file contains all information necessary to load that diagram into the Ameos editor. When you save a diagram in a particular editor, the editor creates or writes to this ASCII file, then executes the necessary Ameos program to load the file into the repository.
A diagram must be syntactically correct for Ameos to write the information from the ASCII file to the repository.

You do not need to manipulate the ASCII files directly to use Ameos. However, you will see informational messages displayed by the editors indicating that these files are being saved or loaded. Also, some system and repository administration procedure descriptions mention these files. For more information about these procedures, see "Backup and Recovery" in the Repository Manager Maintenance chapter of the Ameos Administration Guide.

**ACD, MDA-Based Model Transformation**

Architecture Component Development (ACD) is a transformation engine of Ameos that provides a more powerful and flexible approach for automatically generating solutions from your Ameos/UML models. Using ACD, a majority of the implementation code for your application can be generated automatically from your model. ACD implements a template-based approach that allows you to specify the architectural aspects of the system under construction.

ACD is designed to let you model your system from a design perspective, rather than focus on implementation details that can be handled by the templates.

For an overview of using ACD, refer to the online help ("About ACD templates" on the Index tab). For detailed information on ACD, see the ACD Programming Guide and Using ACD Templates manuals.
Profiles are an easy way to extend standard UML notation to project specific needs. The UML 2.0 super-structure defines how profiles can be defined by using UML notation.

Ameos/UML profile support consists of:

- A profile editor
- A set of predefined stereotypes and tagged values
- A semantic checker for the profile

The profile editor is invoked as is any other editor, from the desktop.

Figure 1 shows a typical profile editor display.

**Figure 1: The Profile Editor**
Profile Editor

The Ameos profile editor can be used to define new UML profiles and to extend predefined ones.

The model and notation elements that support profiles are stereotypes, tagged values, and constraints; they are implemented through property sheets and display marks and symbols.

The node symbols available in the profile editor are:

- Profile
- Stereotype
- Metaclass
- Attribute
- Enumeration

The links available are:

- Extension
- Composition

Figure 2 shows a section of the UML standard profile. This profile is provided in the example system <Ameos>/examples/uml_profile.

Figure 2: Portion of the Standard UML Profile Model
How to Model Profiles in Ameos

To model a profile, you use the profile symbol, which is very similar to a package symbol in class diagrams, label it with the profile name, and group all other elements in the profile symbol to show they are defined in that specific profile. The <<profile>> stereotype is automatically assigned the profile symbol.

Note for Sybase users. If you are using Sybase, you may need to initialize the uml_profile system manually before you attempt to use the profile editor.

On Unix, simply run the INIT_EXAMPLES script and select uml_profile as the system to be initialized.

On Windows, you need to destroy, then re-create uml_profile under Sybase. In other words:

1. Run System > Repository > Destroy System Repository on the uml_profile system.
2. Run System > Repository > Recover System Repository (using Sybase as database type)

Define a Stereotype

To define a specific stereotype, draw a stereotype symbol and label it with the name of the stereotype you want to define. It is automatically assigned a <<stereotype>> stereotype.

The stereotype needs to be attached to a specific element of the UML metamodel such as classes, use cases, etc. To do this, first add a metaclass symbol (which is automatically assigned a <<metaclass>> stereotype) and label it with the type of UML symbol for which you want the stereotype defined (e.g., UmlClass, UmlAttribute, UmlActor, etc.).

To assign a stereotype to a UML symbol type, draw an “extension” link from the stereotype to the metaclass symbol. This defines the stereotype as being valid for the specific UML symbol can now be used in other UML diagrams.

Define Tagged Value for a Stereotype

To define valid tagged values for a stereotype, you add the names of the tags as attributes (add attribute symbols to the stereotype symbol). You can set valid values by setting the attribute type (the attribute type is defined by you). You can also set a default value that will be set for attributes.

Define a List of Valid Values

To define a list of valid values, add an enumeration symbol to the profile. It is automatically assigned an <<enumeration>> stereotype. Add the valid values as attributes to the enumeration, then set the type of the tagged value definition to the name of the stereotype. The values of the stereotype will show up as a choice list in the extensibility property of the UML editors.

Define a Composition of Stereotypes

The profile editor also provides a composition link to define the relationship of stereotypes. Composition from S to P means that S can only be set for an element SE, if P is set for SE’s parent element PE. If SE is a UmlAttribute and PE is a UmlClass, you could define that a stereotype S is only valid for the attribute, provided the attribute’s class has stereotype PE.

Using Predefined Profiles

Predefined profiles can be imported into Ameos systems for further re-use. When you create a new system, you are presented with a list of existing profiles, where you can either select one or many to be included in the new system. Profiles can also be imported at any time by using a right mouse click on the profile editor icon in the Ameos desktop and choosing Import Profile(s).
A collection of profiles, including a predefined profile that defines the predefined/common stereotypes and tagged values, is distributed in a separate Ameos system. By default, this system is called `uml_profile`, but this can be changed in the preferences of Ameos by setting "uml_profiles" to a different Ameos system. Preferences are specified by choosing **Admin > Preferences** from the desktop.

### Reusing Profile Definitions

Profile support in the individual editors is implemented by using the properties window. You can now go back to, e.g., the class editor, select the class symbol, and open its extensibility page in the properties window.

**Figure 3: Extensibility Property Sheet**

![Extensibility Property Sheet](image)

Reusing a set of stereotypes, tagged values, and constraints defined in a profile is implemented mainly with picklists.

### Default Profiles for a New System

To avoid adding new profiles to your system manually, you can create an Ameos system that contains all the profile diagrams that your new systems should have from the start. If you set Ameos’s preference variable to point to that system, all profile diagrams in that system will be copied to your newly created systems.

### Usage Note

When you delete stereotypes or their attributes in a profile diagram, or delete extension links there, you need to run a semantic check from the Ameos desktop. You can do this with **System > Check > Check Semantics for Whole Model**. This reports any stereotypes and tagged values that were orphaned by modifying the profile diagram. You should then manually clean up those orphaned stereotypes and tagged values in the **Extensibility** tab of the element’s property window.

### Predefined Ameos/UML Stereotypes

Predefined stereotypes add specific semantics to the elements they are attached to. Consequently, predefined stereotypes have an impact on code generation (user-defined stereotypes do not).

Ameos/UML supports the predefined stereotypes listed in Table 1.
**Note:** An asterisk next to an item indicates that the item is an OMG 1.4 model element, but is not classified as a stereotype.

### Table 1: Ameos/UML Predefined Stereotypes

<table>
<thead>
<tr>
<th>Base Element/Stereotype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td></td>
</tr>
<tr>
<td>actor*</td>
<td>External element that interacts with the system</td>
</tr>
<tr>
<td>enumeration*</td>
<td>Domain consisting of a set of identifiers (primitive type)</td>
</tr>
<tr>
<td>interface*</td>
<td>Interface of the element</td>
</tr>
<tr>
<td>metaclass</td>
<td>Type that is a metaclass</td>
</tr>
<tr>
<td>powertype</td>
<td>Type that is a powertype of a generalization</td>
</tr>
<tr>
<td>utility</td>
<td>Type that has no instances</td>
</tr>
<tr>
<td><strong>Component</strong></td>
<td></td>
</tr>
<tr>
<td>application*</td>
<td>Executable program</td>
</tr>
<tr>
<td>create</td>
<td>Event denoting that the instance receiving the even has just been created</td>
</tr>
<tr>
<td>destroy</td>
<td>Event denoting that the instance receiving the event is being destroyed</td>
</tr>
<tr>
<td>document</td>
<td>Generic file that is not a source or executable file</td>
</tr>
<tr>
<td>executable</td>
<td>Program file that can be executed on a computer</td>
</tr>
<tr>
<td>file</td>
<td>Document containing source code</td>
</tr>
<tr>
<td>library</td>
<td>Static or dynamic library</td>
</tr>
<tr>
<td>page*</td>
<td>Page associated with the internet</td>
</tr>
<tr>
<td>table</td>
<td>Database table</td>
</tr>
<tr>
<td><strong>Dependency</strong></td>
<td></td>
</tr>
<tr>
<td>access</td>
<td>Target element whose public contents are accessible to the source</td>
</tr>
<tr>
<td>become</td>
<td>Element that is both the source and the target of the dependency</td>
</tr>
<tr>
<td>bind*</td>
<td>Target type or collaboration that is bound to the source type or collaboration</td>
</tr>
<tr>
<td>call</td>
<td>Source operation that invokes a target operation</td>
</tr>
<tr>
<td>copy</td>
<td>Two elements that are identical</td>
</tr>
<tr>
<td>derive</td>
<td>Two elements where one is derived from another</td>
</tr>
<tr>
<td>friend</td>
<td>Extends the visibility of the import dependency between two elements</td>
</tr>
<tr>
<td>implementation</td>
<td>Specifies that the child inherits the implementation of the parent</td>
</tr>
<tr>
<td>implicit</td>
<td>Specifies that the association is only conceptual</td>
</tr>
<tr>
<td>import</td>
<td>Source element that can reference the public contents of the target element</td>
</tr>
<tr>
<td>instantiate</td>
<td>Target element that is an instance of the source element (for example, an object is an instance of a class)</td>
</tr>
<tr>
<td>metaclass</td>
<td>Target element that is a metaclass of the source element (type)</td>
</tr>
<tr>
<td>powertype</td>
<td>Target element (type) that is a powertype of the source element (generalization)</td>
</tr>
<tr>
<td>realize</td>
<td>Target element that implements the source specification</td>
</tr>
<tr>
<td><strong>Generalization</strong></td>
<td><strong>Class</strong></td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
</tr>
<tr>
<td>extend*</td>
<td>implementationClass</td>
</tr>
<tr>
<td>include*</td>
<td>interface*</td>
</tr>
<tr>
<td>subclass*</td>
<td>metaclass</td>
</tr>
<tr>
<td>subtype*</td>
<td>powertype</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Refine</strong></th>
<th><strong>Role</strong>*</th>
<th><strong>Send</strong></th>
<th><strong>Trace</strong></th>
<th><strong>Uses</strong>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target element that expands the specifications of a source element (for example, an instantiated class is a refinement of a parameterized class)</td>
<td>Association behavior</td>
<td>Source operation that sends a signal to the target operation</td>
<td>Dependency between a source element in one model and a target element in the same or different model</td>
<td>Source use case that includes the behavior of the target use case</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Use Case</strong></th>
<th><strong>Type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>facade</td>
<td>Package that references but never owns other elements</td>
</tr>
<tr>
<td>framework</td>
<td>Package containing elements that specify a reusable system architecture</td>
</tr>
<tr>
<td>metamodel</td>
<td>Model indicating that the model is an abstraction of another model</td>
</tr>
<tr>
<td>stub</td>
<td>Package that is not completely transferred</td>
</tr>
<tr>
<td>subsystem*</td>
<td>System that is contained in another system</td>
</tr>
<tr>
<td>systemModel</td>
<td>Model containing a collection of models of the same physical system</td>
</tr>
<tr>
<td>topLevel</td>
<td>Package denoting the highest level in a containment hierarchy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Note</strong></th>
<th><strong>Constraint</strong>*</th>
<th><strong>Requirement</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Semantic condition or restriction</td>
<td>Responsibility or obligation</td>
</tr>
</tbody>
</table>
3 The Script Manager

The script manager manages the generation of code and documents through "scripts."

A script can be seen as a program that generates code or documentation from the models. Each script starts in file named main, which is stored in a directory named like the script. As is the case with a program, a script can have parameters (sometimes called externals) and can include other files.

This chapter describes the general use of the script manager interface. It does not discuss details about code or report generation. For information on report generation, refer to Chapter 4, "Report Generation," on page 26. For detailed information on creating scripts and on using the query and reporting language (QRL) for document generation, refer to Query and Reporting System.

**Note:** Although this chapter describes how to access, create, modify, or run a script, or to assign values to its parameters, it is not necessary to use the script manager to perform these tasks. A script can be created by using any text editor anywhere on the system. Once a script is created, you run it from the script manager, or you can run it from the command line in a shell window using the qrp or acd command. The way in which you create, edit, and run scripts is a matter of personal preference.

The script manager is part the Ameos desktop. (Script manager commands are discussed in "Using the Script Manager" on page 20.) If the script manager window is not visible, choose View > Show Script Manager from the desktop. For code generation, you can select a programming language from the Code menu to view the script manager.
Script Manager Layout

The script manager window consists of a code view and a documents view. The views are laid out in a **Code** tab and a **Documents** tab. Each tab has a scripts pane (left) and a parameters pane (right). The left pane is a tree structure listing all code/document targets. The right pane allows users to edit external variables associated with the selected script or template.
For information on report generation, refer to Chapter 4, "Report Generation," on page 26.

Scripts Pane

The Code/Documents tabs of the script manager window provide a means for navigating through the directory hierarchy where the ACD/QRL scripts reside.

This directory hierarchy contains general script areas, categories of scripts within script areas called script families, and scripts themselves, which reside in script families. Areas and families are directories (represented by folder icons); scripts are files (represented by parameter icons).

By default, there are Product Scripts, Site Scripts, and System “uml_email” Scripts. The Product Scripts area in Figure 2 is open, revealing script families.

To display scripts that are not located in the default Ameos location, use the scriptman_user_scripts_location in the Ameos preferences. For more information, see "Adding a Site Scripts Location" on page 24.
### Parameters Pane

The parameters pane of the script manager window displays the parameters associated with a selected script. Parameters are placeholders whose values can be assigned at run time, either by using the script manager or the command line. The parameters pane is blank if the selected script has no parameters.

When a selected script has parameters, each parameter is listed and preceded by an icon, as shown in Figure 3. For information on editing default values, see “Specifying Values for Parameters” on page 21.

### Using the Script Manager

Script manager commands are accessible from the **Script** menu on the desktop; appropriate script manager commands are also accessible from shortcut menus of selected items the script manager window.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>For Details, See</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Run Script</strong></td>
<td>Runs the selected script using your specifications for output file, format file, and whether to view the output file after the run.</td>
<td>“Running a Script from the Script Manager” below</td>
</tr>
<tr>
<td><strong>Terminate Script</strong></td>
<td>Stops a script that is currently running.</td>
<td>“Terminating a Script” below</td>
</tr>
<tr>
<td><strong>Rescan</strong></td>
<td>Updates the currently selected script or template in the Code/Documents lists. For instance, if you are modifying a script by adding parameters, this command updates the Parameters list to show the newly defined parameters. Also updates the Code/Documents lists if outside scripts are copied into the scripts folder.</td>
<td></td>
</tr>
<tr>
<td><strong>Edit Script</strong></td>
<td>Displays the selected script in a text editor window.</td>
<td>“Editing a Script” on page 23</td>
</tr>
<tr>
<td><strong>Create Script</strong></td>
<td>Creates a copy of a script template in a text editor window. The new script is named according to your specifications.</td>
<td>“Creating a Script” on page 22</td>
</tr>
<tr>
<td><strong>Copy Script</strong></td>
<td>Copies the selected script to the specified name and location.</td>
<td>“Copying a Script” on page 21</td>
</tr>
<tr>
<td><strong>Set to Default Value</strong></td>
<td>Restores the value of the selected parameter to the default.</td>
<td>“Deleting a Parameter’s Value” on page 21</td>
</tr>
<tr>
<td><strong>Delete Script</strong></td>
<td>Deletes the selected script area, script family, or script.</td>
<td>“Deleting a Script” on page 23</td>
</tr>
<tr>
<td><strong>Edit Value in External Editor</strong></td>
<td>Allows you to enter a value for the selected parameter in a text editor window.</td>
<td>“Specifying Values for Parameters” on page 21</td>
</tr>
<tr>
<td><strong>Show Path</strong></td>
<td>Displays the name of the path for the selected script in the message log.</td>
<td></td>
</tr>
</tbody>
</table>

### Running a Script from the Script Manager

This section explains how to run a script from the script manager. It uses the script `all_files` as an example.

To run a script:

1. With the script manager window open, select the script you want to run from the scripts pane. If the script includes parameters, they appear in the parameters list.
2. (Optional) If the script includes parameters, make any desired changes to their values. Procedures are described in “Specifying Values for Parameters” on page 21.

3. Right-click on the script and choose Run Script. You may also choose Run Script from the desktop (Script > Run Script). The results of running the QRL script or ACD template are displayed in the message log.

**Terminating a Script**

To terminate a script that is running, choose Terminate Script from the selected script, then press OK in the confirmation box. The script terminates.

**Specifying Values for Parameters**

A variable declared in a script for which you can assign values at run time is called a “parameter.” This section describes how to set values for parameters at run time by using the script manager.

For information on how parameters are defined in a script, see the “Ameos Query and Reporting Language” chapter of *Query and Reporting Language*.

**Specify a Value at Run Time**

When you declare parameters in a script, you can include default values, but this is not required. When you run the script, you must supply a value for a parameter, if:

- No value is defined for the variable in the script.
- You want to use a different value than the one defined in the script.

To change the value of a parameter, double click on the parameter. Depending on the type of the parameter, you can edit the value then or you can choose a value from the list. All specified values are retained until you change them. Changed parameters are marked with a small green square in front of the variable name.

Changed values of parameters are valid Ameos system wide. This means that all users who run the same product scripts for a system are using the same set of externals.

If you want a user (or a group of users) to have individual parameter settings for product scripts, set the preferences variable `scriptman_product_externals_location` to a different location in the file system.

All scripts of Ameos can be run from the command line as well. In this case, the specified values will not be applied.

**Deleting a Parameter’s Value**

Deleting the value of a parameter causes the default value, as declared in the script, to be restored.

To delete a parameter’s value:

1. Select the variable in the Parameters list.
2. Right click, then choose Set to Default Value.
3. In the confirmation box, click Yes. The values revert to the default values.

**Copying a Script**

You can use the script manager to make a copy of a script. This is useful if you want to use the same script with different sets of parameters. For example, you can use the UML report provided with Ameos
To generate a requirements document with use cases or a design document with all classes, just by setting different parameters.

To copy a script:
1. Select a script in the scripts list.
2. From the **Edit** menu, choose **Copy Script**.
   A dialog appears.

3. In the dialog, verify the script you want to copy in the **From** group.
4. If necessary, display the **Area** and **Family** option lists in the **To** group to change the target location.
   Alternatively, type a new family name in the **Family** input field.
5. In the **Script** input field, type the destination script’s file name.
6. Select **Edit After** if you want to edit the script after the copying is completed.
7. Select **Copy Externals** if you want to copy the original script’s parameter values to the new script.
8. Click **OK**. The script is copied into the specified script area and family. If you elected to edit the script, a text editor window appears with the new script.

### Creating a Script

Creating a script with the script manager is similar to copying a script, since the initial step copies a template script, `simple_script`, to a target location. The `simple_script` is included in the Product Scripts area of the scripts pane.

To create a script, you must be familiar with QRL. For information about QRL, see the "**Ameos Query and Reporting Language**" chapter of *Query and Reporting Language*.

To create a script:
1. From the desktop, choose **Script > Create Script**. A dialog appears.

2. In the dialog, specify where you want to save the script when you finish creating it. If necessary, display the **Area** and **Family** option lists to change the target location. Alternatively, type a new family name in the **Family** input field.

3. To name the script, type the name of the script in the **Script** field, replacing the current name.

4. If you want to edit the script, check **Edit After**.

5. Click **OK**. A text editor window appears, displaying a copy of *simple_script*. You can now create a script, using the *simple_script* template.

The name of the new script appears in the area specified.

**Editing a Script**

To edit a script, you must be familiar with QRL. For information about QRL, see the "Ameos Query and Reporting Language" chapter in *Query and Reporting System*. To edit a script from the script manager:

1. In the scripts list, select the script you want to edit.
2. From the **Edit** menu, choose **Edit Script**.
3. Make the desired changes to the script, save the file, and exit the editor.

**Deleting a Script**

To delete an area, script family, or script:

1. In the scripts pane, select the area, script family, or script you want to delete.
   
   **Note:** To delete a script area or family, the area or family icon must be open and empty.

2. From the shortcut menu for the item, choose **Delete Script**.
3. In the confirmation box, click **Yes**.

**Directory Structure for Scripts**

In order for a script (QRL or TDL) to be displayed by the script manager, it needs to be located in defined directories, with a predefined name for the file that contains the main entry point or function and the external parameter definition.

For QRL, here is the structure for HTML reports; other reports have a similar structure.

- **templates/uml/qrl/** – Base directory for all product scripts
- **templates/uml/qrl/Reports/** – Directory containing the script family
- **templates/uml/qrl/Reports/HTML/** – Directory for the specific script type
- **templates/uml/qrl/Reports/HTML/main.qrl** – The “entry-level” QRL file containing the parameter definition and the main function (this file generally includes other QRL files)

All **main.qrl** files located in the above structure – inside **templates/uml/qrl** (or the same structure inside the currently opened system’s qrl_files directory) – are picked automatically by the script manager.

For TDL (and ACD), the structure is similar. Here is the structure for Java templates; other templates have a similar structure.

- **templates/uml/tdl/** – Base directory for all product templates
- **template/uml/tdl/Languages/** – Directory containing the script family
- **templates/uml/tdl/Languages/Java/** – Directory for the specific script
- **templates/uml/tdl/Languages/Java/main.tdl** – The “entry-level” TDL file containing the parameter definition and the main procedure (usually this file includes other TDL files)

All **main.tdl** files located in the above structure – inside **templates/uml/tdl** (or the same structure inside the currently opened system’s qrl_files directory) – are picked automatically by the script manager

## Adding a Site Scripts Location

If you want to add site-specific QRL scripts and TDL templates to the script manager, set the following variables in the **Edit Preferences** dialog appropriately.

```plaintext
site_scripts_path
site_templates_path
```

The site-specific scripts and templates appear under the Site Scripts category of the appropriate script manager window (code or documents).

## Preferences for the Script Manager

There are several variables that are relevant to the operation of the script manager. These variables can be set from the Ameos desktop with **Admin > Preferences**. All the preferences are stored in file named `ToolInfo.<platform>`, located in the Ameos product folder. (Note that “preferences” are sometimes referred to as “ToolInfo variables.”)

Preferences of the script manager are listed in Table 2.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>script_editor</strong></td>
<td>String</td>
<td>For Windows—notepad $(scriptpath) &amp; For UNIX—dtpad $(scriptpath) &amp;</td>
<td>Specifies the editor to be used by the script manager for editing scripts.</td>
</tr>
<tr>
<td><strong>scriptman_ascii_view</strong></td>
<td>String</td>
<td>For Windows—notepad $(ascii_file) &amp; For UNIX—xterm -e view $(ascii_file) &amp;</td>
<td>Specifies the command to use to view ASCII output from the script manager.</td>
</tr>
<tr>
<td><strong>scriptman_product_externals_location</strong></td>
<td>String</td>
<td>Optional. No default value.</td>
<td>Points to the directory for product externals.</td>
</tr>
</tbody>
</table>
### Table 2: Script Manager Preferences

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scriptman_system_externals_location</code></td>
<td>String</td>
<td>Optional. No default value. Points to the directory for system externals.</td>
<td></td>
</tr>
<tr>
<td><code>scriptman_run_xtterm</code></td>
<td>String</td>
<td>xterm -title ${script_title} -sb -e ${script_cmd}</td>
<td>Controls the xterm window that runs a QRL script. (UNIX only)</td>
</tr>
<tr>
<td><code>scriptman_script_process</code></td>
<td>String</td>
<td>qrp -p ${projdir} -s ${system} ${scriptpath} ${targetargs}</td>
<td>Specifies the process that executes QRL scripts from the script manager.</td>
</tr>
<tr>
<td><code>site_scripts_path</code></td>
<td>String</td>
<td>Optional. No default value. Points to the location of &quot;site scripts,&quot; which are distributed in a hierarchy similar to that in <code>templates/qrl</code> or <code>templates/tdl</code>. The scripts are displayed in the script manager, in the Site Scripts category, below Product Scripts.</td>
<td></td>
</tr>
<tr>
<td><code>site_templates_path</code></td>
<td>String</td>
<td>Optional. No default value. Points to the location of &quot;site scripts,&quot; which are distributed in a hierarchy similar to that in <code>templates/qrl</code> or <code>templates/tdl</code>. The scripts are displayed in the script manager, in the Site Scripts category, below Product Scripts.</td>
<td></td>
</tr>
</tbody>
</table>
This chapter discusses the Ameos report generation facility.

There are two ways to generate reports:

- Through the script manager (below) with predefined scripts
- Through the report maker editor (page 28) with creating new scripts

### Generating Reports with the Script Manager

You generate "UML reports" and "HTML reports" through the script manager interface. The script manager interface is discussed in Chapter 3, "The Script Manager" on page 17. Please refer to that chapter for usage information. Figure 1 shows the documents view of the script manager window, through which you generate UML and HTML reports.

#### Figure 1: Documents View of the Script Manager Window

This section discusses the elements of UML reports (below) and HTML reports (page 27). The purpose of these two report types is quite different. The output of the UML report is a single document – e.g., in RTF format, with several chapters to provide a printable document of the modeled system. The HTML report generates a set of linked HTML pages that can be used as online documentation of the modeled system.

### UML Reports

The UML report provides information pertaining to an entire UML model. The report includes several chapters; they are described in the table below.

<table>
<thead>
<tr>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Includes the front page, table of contents (RTF only), and introduction. To view the table of contents in RTF, update the table of contents in Microsoft Word.</td>
</tr>
</tbody>
</table>

| Users View |
Displays use case diagrams in alphabetical order. Includes detailed information on actors, packages, use cases, and use case scenarios.

System Architecture
Displays the system architecture in logical (package hierarchy) and physical (component and deployment) implementations.

Package Report
Displays information pertaining to packages, including package hierarchy, contents, and connections to collaboration and sequence diagrams.

Class Report
Displays classes in alphabetical order and includes class table descriptions and table diagrams. Includes related class information in sequence, collaboration, and activity diagrams.

Profiles
Displays profile diagrams in alphabetical order. Includes detailed information from profile annotations.

When running a report, you can use external parameters (sometimes called external variables) to customize its scope.

**Changing External Parameters**

You can replace the default format file with the name of any other existing format file (for the chosen format). If your format file contains paragraph and character formats other than those defined in `ameosreport.<extension>`, you have to adjust their names in the template file `ALL_Formats.inc`. For information on format files, see the "Printing Text" section in Chapter 4 of *Query and Reporting System*.

**Running UML Reports from the Script Manager**

To run a report from the script manager:

1. In the script manager window, open the *Product Scripts* folder, then the *Reports* folder.
2. Right-click on the UML report desired (e.g., RTF_English – see *Figure 1 on page 26*).
3. Change external parameters as desired.
4. Choose *Run Script* (from the UML report you selected in the documents window).

The progress of document generation is reported by messages in the message log of the Ameos desktop.

**Viewing UML Reports**

By default, all generated reports are displayed in the desktop’s model browser, in the Document View. When you click on Microsoft Word files, all UML reports will be listed in the right pane of the browser. The report can be opened by a double click. You can also view UML reports by choosing *File > Open Report* from the desktop.

**HTML Reports**

HTML report generation you to document a UML system in a set of linked HTML pages. The structure of these pages is similar to those in the JavaDoc standard.

The report consists of a master index that links to subindex pages such as Actor, Use Case, Package, Class Index, as well as to an index that lists all diagrams in the system. Each diagram listed links back to its graphical representation.
High level UML elements are documented textually and are cross-linked with their references in diagrams. The report also provides hyperlinks from symbols in diagrams back to their textual description.

To generate an HTML report for your model:

1. In the script manager window, open the Product Scripts folder, then the Reports folder.
2. Right-click on HTML, then choose Run Script.

By default all generated reports will show up in the desktop’s model browser, in the Document View. To view an HTML report, click on HTML Files and then double click on index.html. You can also view HTML reports from by choosing File > Open HTML Report from the desktop.

### UML Report-Generation Files

Files involved in UML report generation are listed below.

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format files</td>
<td>/templates/ct/print_format/&lt;publishing tool&gt;/ameosreport.&lt;extension&gt;</td>
</tr>
<tr>
<td>Format include file</td>
<td>/templates/ct/qrl/include/ALL_Formats.inc</td>
</tr>
<tr>
<td>Ameos scripts</td>
<td>/templates/uml/qrl/Reports/</td>
</tr>
<tr>
<td></td>
<td>RTF_English</td>
</tr>
<tr>
<td></td>
<td>RTF_French</td>
</tr>
<tr>
<td></td>
<td>RTF_German</td>
</tr>
<tr>
<td></td>
<td>HTML</td>
</tr>
<tr>
<td>HTML reports</td>
<td>&lt;projdir&gt;/&lt;system&gt;/html_files/</td>
</tr>
<tr>
<td></td>
<td>Index.html</td>
</tr>
<tr>
<td></td>
<td>&lt;file&gt;.html</td>
</tr>
<tr>
<td>UML reports</td>
<td>&lt;projdir&gt;/&lt;system&gt;/qrl_files/</td>
</tr>
<tr>
<td></td>
<td>UML_Report_&lt;language&gt;.&lt;extension&gt;</td>
</tr>
<tr>
<td></td>
<td>UML_Report_&lt;language&gt;.html (.rtf or .asc)</td>
</tr>
<tr>
<td>Script settings</td>
<td>&lt;projdir&gt;/&lt;system&gt;/qrl_files/product.ext</td>
</tr>
</tbody>
</table>

### Generating Reports with the Report Maker Editor

The Report Maker Editor (RME) is a tool that allows you to describe in graphical terms a report consisting of Ameos diagrams, tables, text, and database objects. Sections of the report are assembled into a diagram, which is then used to generate the set of QRL commands that make up the report. This capability allows even novice users to easily define a report structure and to generate finished documents without much knowledge of QRL.

This section contains the following topics:

- “Overview of RME Usage” (immediately below)
- “Creating RME Diagrams” on page 29
- “Generating the Report” on page 30
- “RME Objects and Annotations” on page 32
- “Additional Information” on page 35

### Overview of RME Usage

The RME is accessed from the Ameos desktop’s Document View. To generate reports with the RME, you:
Create a new diagram. On the Ameos desktop, go to the Document View category, right-click on Report Maker Diagrams, then select **New**. The report maker editor opens. Figure 2 shows the RME at startup, minus the panner/find and message log windows.

- Graphically define the structure for your report. To do this, insert document icons from the Symbols toolbar and then connect the icons with arcs.
- If desired, add or modify annotations and notes associated with the icons. To do this, select the object, then set the annotation items as desired in the **Annotations** tab of the properties window.
- Generate the QRL for the report (**Reports > Generate QRL Report**).
- Run the generated QRL (**Reports > Run Report > Run Generated QRL <output_type>**).
- View or print the report as desired.

**Figure 2: RME Window**

Creating RME Diagrams

To create a report with the RME, you begin by graphically defining the structure of the report.

A report can have the following elements:

- A cover page
- A single diagram or a set of diagrams
- A single table or a set of tables
- A set of text paragraphs
- A set of database objects
Each of these items is represented by an icon. By connecting the icons to each other, you define the order of the report. By annotating the icons, you define the type of diagram, table, or object, as well as specify information on paragraph styles, orientation, and other attributes.

Here are rules for connecting the icons:

1. You cannot create circular reports. Semantic checking will flag this as an error.
2. The cover page icon can only have an outlink – no inlinks.
3. All other icons can have a maximum of one outgoing link and one incoming link.
4. Links may be either a straight arc or a spline.
5. All icons must have a label.
6. Icons may be reused on different reports; they will retain their properties.

Once you have constructed a diagram and have labeled the icons, you should then save the diagram. At that point you can start annotating the individual icons.

Figure 3 shows a sample diagram. In it, you can see examples of the various icons, each of which represents a document section. The report begins with a cover page followed by two text sets, and then prints the diagram.

Figure 3: Sample Diagram Generated by the RME

Generating the Report

Once you have filled in the attributes for the icons, you can generate the QRL code that will create the report. This function is accessed via the RME’s Reports > Generate QRL Report command. During this process, you will see status messages. When finished, the RME produces a QRL file in the rme_files directory of your system.

The file that was created can then be viewed and manually edited, or run to produce the final report. To view the file, select Reports > Edit Generated QRL. This will allow you to edit the file if you wish.

If you are satisfied with the generated QRL, you can then choose to run the QRL. This is done by selecting Reports > Run Report > Run Generated QRL. You can choose to run the report to one of two target document systems: standard ASCII or RTF. This produces a file of one of the types in the qrl_files directory.

Below is the portion of the QRL code produced by the example.

```qrl
// example.qrl
// This file was automatically generated by Ameos’s Report Maker
// 01/29/04 15:07:56

// GLOBALS
string query = "";
```
void main()
{
    file file_var;
    string inc_file = "";
    target_enum target = target();
    message("Starting report processing...");

    if (target == Ascii) format("rme_report.asc");
    if (target == RTF) format("rme_report.rtf");

    // RME Example
    // CoverPageSection
    message("-- Processing RME Example...");

    if (target == Ascii) paragraph("center");
    if (target == RTF) paragraph("Title");

    print_line("");
    print_line("RME Example");
    print_line("");

    if (target == Ascii) paragraph("center");
    if (target == RTF) paragraph("Author");

    print_line("Project Directory : F:/Ameos9.0/examples/");
    print_line("System Name : Millennium_Clock_Java");
    print_line("Quick Example");
    page();

    // Introduction
    // TextSection
    message("-- Processing Introduction...");
    print_line("Introduction to the report");

    // Class Diagrams
    // MultiDiagSection
    message("-- Processing Class Diagrams...");

diagram_orientation_scale_set(BestFit);
diagram_caption_orientation_set(Portrait);
diagram_caption_paragraph_format_set("Figure");
diagram_frame_height_set(8.000000);
diagram_frame_width_set(16.000000);
query = "file[UmlClassDiagram]";
for_each_in_select(query, file_var)
{
    file_print(file_var, "Diagram: " + file_var.name);
}

    // Class Details
    // MultiObjectSection
    message("-- Processing Class Details...");

    // Declare a set of uniquely named variables each time.
    node node_var1;
    node node_ops_var1;
    node node_attr_var1;
    link in_link_var1;
    link out_link_var1;
    note note_desc_var1;
    item item_val_var1;

    print_line("--- Database report on type : UMLClass ---");
    query = "node[UMLClass]";
RME Objects and Annotations

This section discusses each RME object, along with the specific annotations that apply to it. To see what the icon for some of the available objects looks like, refer to Figure 3 on page 30.

Generic annotation items (those associated with all or most icons) are listed below. Specific annotation items are discussed in sections according to icon type: Cover Page and Text Set (immediately below), Diagrams and Diagram Set (page 33), Table and Table Set (page 34), and Object Set (page 34).

---

**Generic Report Options**

- **Start On New Page**
  - Specifies that generated output begin on a new page. If it is not present, or is set to False, the icon’s information will flow immediately after the last icon’s information on the same page (if possible).

- **Text Report Options**
  - Custom | ASCII | RTF Paragraph Style
  - Specifies paragraph format. Custom allows a user-defined format to be specified.

- **Include File**
  - Allows you to specify an external file to be included in the output text. If a file is specified, it will appear after any note descriptions and will allow you to use the same formatting. You should specify the full path to any file.

- **Object Name**
  - Specifies the name of the object

---

**Note:** Annotations other than those described in the sections below can be ignored; they do not need to be filled in. Examples of such annotations are Requirement, Glossary Definition, and Requirement Allocation.

---

**Cover Page**

The Cover Page icon is typically used to start a document. It provides a convenient way to create a simple cover sheet. Annotation items specific to the cover pages are shown below.

- **Title**
  - Set to icon’s label by default.

- **Author**
  - Set to user log-in by default. If true, the date is placed under Title and Author.

---

**Text Set**

The Text Set icon allows you to insert text paragraphs and headings into your final document. If you look at the document you are reading now, you can see various section headings and text paragraphs that contain descriptions for each section.

Each icon can actually contain multiple sections of paragraphs and heading information. The information for each is contained in an annotation called Text Report Options. The text of the annotation note is displayed in the Description area in the annotations page of the properties window. An easy way to see this text is to select a text report option in the annotations page and then double-click on the note. This will display the editable note description.
Annotation items specific to text sets are described below.

**Custom |ASCII |RTF Paragraph Style**

Specifies paragraph format. Custom allows a user defined format to be specified.

**Include File**

Allows you to specify an external file to be included in the output text. If a file is specified, it will appear after any note descriptions and will use the same formatting. You should specify the full path to any file.

When you define paragraph styles, you are supplied with a set of predefined values specific to that style. The only exception is the custom paragraph style, which allows a user-defined style to be specified. Make sure that the report format file that you are using defines any custom paragraph styles. If you fail to set a paragraph style, the document will continue using the last paragraph style specified.

Predefined styles are found in a “format file.” This is a file that contains paragraph and character formatting information. The RME uses a supplied set of format files called rme_report. If you need to specify a custom format file (versus the default rme_report), you can do so by annotating the diagram itself. There is a special annotation for the diagram that allows you to set your own format file. You will probably want to use the custom paragraph style if you have specified a custom format file.

You can have up to ten (numbered 0 - 9) text report options per Text Set icon. This allows you to put a number of document paragraphs and headings together in one place. If you need more text you can always link up additional Text Set icons. Each one can hold an additional ten text sections.

**Diagrams and Diagram Sets**

Diagrams and diagram sets allow you to include your Ameos diagrams in your documents. The difference between a Diagram icon and a Diagram Set icon is that the Diagram icon produces only a single diagram to output. A diagram set outputs all diagrams of a given type.

Annotation items specific to diagrams and diagram sets are shown below.

**Diagram Type/Name**

Specifies the type/name of the diagram.

**Caption**

Specifies that the caption is to appear at the bottom of the diagram.

**Caption Orientation**

Specifies if the caption is landscape or portrait; default is ‘best fit.’

**Diagram Frame Height/Width**

Sets the height/width of the bounding box for the diagram.

**Use Print Setting**

Instructs the program to use a named print setting.

**Start Each On New Page**

(Diagram sets only) Starts every diagram on a new page.

**Omit Diagram Called**

(Diagram sets only) Excludes a named diagram.

**Sort By**

(Diagram sets only) Sorts the diagrams alphabetically by field (e.g., ‘name’).

The annotations used to support a single diagram differ only in that a specific diagram name is required for a single diagram. By default, this name comes from the label on the icon, but it can be changed.
When you add a Diagram or Diagram Set icon, you need to specify the diagram type (click on **Diagram Type** under Diagram Report Options and select a value from the **Description** list). This identifies the kind of diagram to be printed. If you are printing just a single diagram, you will also need to specify the diagram name.

### Tables and Table Sets

Ameos stores some information, like details of a class and details of a state, in tables. Tables and table sets allow you to include your Ameos tables in your documents. The difference between a Table icon and a Table Set icon is that the Table icon produces only a single table to output. A Table Set outputs all tables of a given type.

Annotation items specific to tables and table sets are shown below.

<table>
<thead>
<tr>
<th>Annotation Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Caption</strong></td>
<td>Sets the caption to appear at the top of the table.</td>
</tr>
<tr>
<td><strong>Orientation</strong></td>
<td>Sets the orientation to portrait or landscape.</td>
</tr>
<tr>
<td><strong>Table Width</strong></td>
<td>Sets the width of the bounding box for the table</td>
</tr>
<tr>
<td>**Row</td>
<td>Column</td>
</tr>
<tr>
<td>**Show Row</td>
<td>Column Indices**</td>
</tr>
<tr>
<td><strong>Use Print Setting</strong></td>
<td>Instructs the program to use a named print setting.</td>
</tr>
<tr>
<td><strong>Start On New Page</strong></td>
<td><em>(Table sets only)</em> Starts every table on a new page.</td>
</tr>
<tr>
<td><strong>Omit Table Called</strong></td>
<td><em>(Table sets only)</em> Excludes a named table.</td>
</tr>
<tr>
<td><strong>Sort By</strong></td>
<td><em>(Table sets only)</em> Sorts the tables alphabetically by field (e.g., ‘name’).</td>
</tr>
</tbody>
</table>

The annotations used to support a single table differ only in that a specific table name is required for a single table. By default, this name comes from the label on the icon, but it can be changed.

When you add a Table or a Table Set icon, you need to specify the table type. This identifies the kind of diagram to be printed. If you are printing just a single table, you will also need to specify the table name.

### Object Sets

The Object Set icon is used to query the Ameos database and report on individual items contained in it. It does not provide clues as to what the queries should be or how to form them. This icon is for more advanced users because you must know about how the Ameos database is organized, and how to formulate OMS queries, in order to use an object set effectively. The resulting QRL code is a looping structure that collects a set of data and provides it for output to the report.

Annotation items specific to object sets are shown below.

<table>
<thead>
<tr>
<th>Annotation Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DB Object Type</strong></td>
<td>Specifies the OMS data type for the object to be reported.</td>
</tr>
</tbody>
</table>
In addition, the same paragraph styles as described under text sets ([page 32](#)) are also available. They allow you to format the output from the generated QRL query.

**Object Set example output**

The following is a sample output from an object set query.

```plaintext
--- Database report on type : MultiObjectSection ---
Object Example
  In Link(s) :
    NextSection :

  Out Link(s) :
    NextSection :

  Note Description for GenericObject :
This text is contained in the GenericObject note description for the 'Example Object' of the ReportTutor RME diagram in the ReportMaker system.
```

**Additional Information**

- For those who need to use advanced features, the RME can be used to quickly provide a working prototype and fine tune the output until additional customization is needed. At that point, the QRL code can be manually edited and reworked to create the report format desired.

- Some features will look different when output to different applications. For instance, RTF does not support vertical spanning cells. Instead, the text will be repeated for each of the cells. RTF also does not support a true landscape diagram. Instead it rotates the paper and your footer will appear on the right edge of the page, sideways. For this reason it is best to keep RTF formatted documents simple and to use portrait mode if possible. Similarly, ASCII output is very limited in its capabilities.

- When you define an RME icon on a given report, you can reuse that icon on a different report; all its formatting and specifications will be carried over. You just need to give the icon the same name and make sure it is the same type.

  If you need to use a set of diagrams in a different report, you just need to recreate the icon on the new report. You can even cut and paste between reports.
You can use the Ameos object-oriented reverse engineering tool to generate a new Ameos/UML model or update an existing model from source code written in C++, Java, Ada 95. The reverse engineering process generates both a semantic and a graphical model of your system. The semantic model contains information extracted from source code; it is used to generate the graphical model.

The graphical model contains one or more of the elements listed below.

- A class diagram and corresponding tabular class content for each class
- An annotation file for each class, attribute, and operation

In addition, it generates a class diagram for each of the following:

- Dependency, association, and aggregation relationships
- Nested relationships
- Inheritance relationships
- Interface implementation relationships

These model elements are described in “Model Elements Generated by Reverse Engineering” on page 49.

The main topics addressed in this chapter are:

- How to parse Java source (immediately below)
- How to parse C++ source (page 38)
- How to parse Ada 95 source (page 44)
- Other parsing topics that apply to the supported languages (page 44)
- How to generate a model from parsed files (page 48)
- Model elements generated by reverse engineering (page 49)

### Parsing Java Source

There are several preferences that need to be set before you use the reverse engineering tool of Ameos. Note that several of these preferences are enabled by default.

- **java_vm** – Path to the Java virtual machine. The current implementation is known to work with both Classic and HotSpot virtual machines.
  
  **Examples:**
  
  ```
  JavaVM=$(env.JAVA_HOME)\jre\bin\classic\jvm.dll
  JavaVM=$(env.JAVA_HOME)/jre/bin/java
  ```

  **Note:** Don’t forget to set the environment variable JAVA_HOME. Setting this variable replaces $(env.JAVA_HOME) when Ameos reads the preferences.

- **javare_javac**: – Path to the Java compiler
  
  **Examples:**
  
  ```
  javare_javac=$(env.JAVA_HOME)\bin\javac.exe
  javare_javac=$(env.JAVA_HOME)/bin/javac
  ```

- **javare_javac_outdir** – Output file directory
Example:

javare_javac_outdir=C:\<project>\src\JavaOutput

- **classpath**: Path to user and system libraries. The value of this preferences variable is appended to the `CLASSPATH` environment variable.

  Example:
  
  classpath=C:\jdk1.3.1

- **sourcepath**: Path to the root source path directory, passed as a parameter to the Java compiler.

  Example:
  
  sourcepath=C:\<project>\src

To parse Java source:

1. Make sure you have a Java 2-compatible JDK installed on your machine.
2. Make sure your `JAVA_HOME` environment variable is set to point to the location of the Java compiler.
3. Make sure the appropriate preferences are set (see “Java-specific variables” below).
4. Choose Code > Reverse Engineering > Reverse Engineer Java Code. The Java parse dialog (Figure 1) appears.

**Figure 1: Parse Dialog for Java**

5. On the **Source Files** tab, prepare a parse list.
   - Select the base directory, which is the top-level directory for your source code. Use the **Choose** button next to the **Current Directory** field to do it.
   - If you want to parse files that match a certain pattern, enter the pattern in the **Filter** field, then click the **Filter** button.

The directory’s contents are displayed in the **Directories** scrolling text field.

To display the contents of a subdirectory, double click on a directory name in the scrolling list.
To display the contents of the parent directory, double click **Parent Directory**, which appears at the top of the scrolling list.

- Select the source file(s), and use the right arrow button to move them to the **Parse List**.

6. On the **Parse Code** tab, select the **Source Type** (Java Source, Java Source Precompiled, Java Class/Jar), then press **Parse Source**.

A semantic model of your Java source is generated.

Once the Java source is parsed successfully, you can make a UML model. What is generated in the model is described in “Model Elements Generated by Reverse Engineering” on page 49.

**Parsing C++ Source**

To parse C++ source:

1. Choose **Code > Reverse Engineering > Reverse Engineer C++ Code**. A parse dialog appears (Figure 2).

   **Figure 2:** Parse Dialog for C++

2. Prepare a parse list. There are several ways to do this.
   - Have the parser recursively read tiles from the current directory. You can either:
     - Manually add files from the **Directories** list to the **Parse List** on the **Source Files** tab. Select the file(s), then press the right arrow.... or
Specify a file pattern to read sets of files from the current directory. For example, to add .h and .cpp files, type "*.cpp *.h" in the File Pattern field on the Source Files tab, then press Populate Parse List.

(Windows platforms) Specify a Microsoft Visual Studio Project file in the DSW File field of the Source Files tab, then press Populate Parse List.

Use a "makefile" to populate the list. This is done on the Makefile tab. See “Using the Makefile Reader” on page 40 for details.

Each of these actions populates the Parse List window on the Source Files tab with the files you have chosen.

3. Fill in any needed preprocessor options (Preprocessor tab). See “Using Parser Preprocessor Options” on page 40 for details.

4. Fill in any needed comment-extraction options (Comments tab). See “Extracting Comments” on page 41 for details.


Once the C++ source is parsed successfully, you can continue with extracting comments from the source code, or you can make a UML model. What is generated in the model is described in “Model Elements Generated by Reverse Engineering” on page 49.

Source File Suffixes

The parser recognizes the following suffixes for C++ files:

- .C
- .h
- .hh
- .cc
- .def
- .cpp
- .cxx

If you want to parse a C++ source file that does not have one of these suffixes, you must enter the full list of suffixes into the RE_CXX_suffixes variable of your preferences. For example, to parse files with the '.c++' suffix, add this line to your preferences:

```
RE_CXX_suffixes=.c++.C:.h:.hh:.cc:.def:.cpp:.cxx
```

Microsoft Extensions

If you select Use Microsoft Extensions on the Parse Code tab, it allows the parser to parse C++ files containing these extended reserved words:

- __asm
- __based
- __cdecl
- __declspec
- __except
- __fastcall
- __finally
- __fortran
- __inline
- __int16
- __int32
- __int64
- __leave
- __multiple_inheritance
- __oldcall
- __pascal
- __single_inheritance
- __stdcall
- __syscall
- __try
Each of the extensions listed above includes a double underscore. For compatibility with previous releases of Microsoft's compilers, a single underscore is also allowed in these extensions.

**Using the Makefile Reader**

The reverse engineering tool for processing C++ provides a feature called the “makefile reader,” which automatically finds the files needed to model your software program with the information contained in an Nmake, a GNU, or a UNIX makefile.

The makefile reader uses the same rules as the UNIX `make`, Microsoft `Nmake`, or Free Software Foundation (FSF) GNU `make` commands to read the makefile and determine the prerequisites for the primary target or any subordinate targets specified on the Makefile tab.

Once you have specified the name of a makefile in the Make File field of the Makefile tab, you press Populate Parse List to create a parse list. The makefile reader displays:

- Prerequisite files in the Parse List window on the Source Files tab.
- Directories containing user-defined and system-defined header files in the User Include Search Directories and System Include Search Directories fields on the Preprocessor tab.

You can preprocess and parse the files as displayed or edit the information before parsing. If you wish to change the default options, change them on the Makefile tab before parsing. To access a full list of options available on the parse dialog, see the Command Reference online help. Click on “About reverse engineering” on the Index tab to find the information quickly.

**Invoking make Recursively**

The makefile reader can trace prerequisite files through a hierarchy of directories when `make` is invoked recursively. By default, there is no limit on the number of directory levels the makefile reader searches; however, you can specify a limit using the Make Depth Bound option. Select 1, 2, 3, 4, or 5 directory levels, or use the default setting of No Limit.

**Selecting Subordinate Targets**

By default, the makefile reader finds the files needed to model the primary target or executable identified in the makefile. You can override the default by specifying one or more subordinate targets in the Make Targets field. Whenever this field has a value, the makefile reader generates prerequisites for the specified targets only.

A subordinate target can be any object in the makefile that has a dependency line. If you specify more than one target, make sure to insert a space between each name.

In addition to specifying targets, you can also define macros in the Make Targets field. Type macros in the form `<name>=<value>`. For example:

```
OBJECTS="db_ant.o ant_files.o ant_main.o ant_ids.o"
```

or

```
BIN=$(HOME)/bin
```

**Using Parser Preprocessor Options**

To change the default preprocessor options, you use the Preprocessor tab on the parse dialog. To access a full list of options available on the parse dialog, see the Command Reference online help. Click on “About reverse engineering” on the Index tab to find the information quickly.
Specifying Include File Search Directories

When you click the Populate Parse List button on the Makefile tab, the makefile reader fills the User Include Search Directories and System Include Search Directories windows (Preprocessor tab).

Generally, you should specify directories as absolute paths, although you can specify relative path names or a combination of both types. Relative paths are evaluated from the directory from which Ameos was started.

The specified directories are searched in top to bottom order.

If you wish, you may change the directories provided. You may include as many search directories as you want.

For C++, user include files contain the #include "" statement, and system include files contain the #include <> statement.

On Windows, you do not need to specify files specified in "include" environment variables, because Ameos automatically adds their paths to the end of the System Include Search Directories list.

Specifying Compiler Directives as Defines

If C++ source code was compiled with preprocessor flags, you may want to use the same compiler directives in parsing the code. Specifying such directives tells the parser to create a semantic model for the version of the software built by the compiler. These flags are usually passed to the compiler from the makefile, which holds all preprocessor flags needed to build each version of the software.

To specify preprocessor flags, enter a string containing the necessary compiler directives, separated by spaces, into the Defines field on the Preprocessor tab. This creates a file called Defines in the workspace directory. If the directive includes spaces or punctuation, enclose it within double quotes. For example:

VERSION="6.0 Beta" MACHINE=SUN DEBUG

Normally, you do not need to use the Defines field. Enter compiler directives only if necessary.

Preprocessing Problems

By default, files are preprocessed before they are parsed. Sometimes preprocessing produces the problem that prevents the files from parsing correctly. To see the preprocessed source code, look in the revc_files/code_dir directory in the current system. A copy of all files that fail parsing is left in this directory.

Missing Source and Header Files

If the parser cannot find the required source files or header files, error or warning messages appear. To correct these problems:

1. Type in the locations of the missing files in the Preprocessor tab.
2. Reparse the files.

If only source files are missing, you can reparse incrementally (Parse Source button on the Parse Code tab). If both source and header files are missing, you must reparse in non-incremental mode.

Extracting Comments

Comments represent an important source of information that can enhance the final design documentation. The source code parser alone does not sufficiently handle comments; it stores comments, but it
does not associate them with classes and functions. To place source code comments appropriately in your semantic model, you must run the source code comment extractor (Extract Comments button on the Comments tab).

The comment extractor tries to associate source code comments with data structures, functions, and global variables by using search parameters that you provide. It then concatenates the comments and writes them to the semantic model. The model generator uses the comments in the semantic model to create annotations for diagrams in the graphical model.

The comment extraction phase is optional. If you do not extract comments from the source code, they do not appear as annotations in your model.

The Comments tab allows you to specify search parameters for finding and extracting comments from source code.

You specify comment extraction parameters by setting non-exclusive options for each of the source code object categories in the dialog:

- Class Comment settings—options for extracting comments related to classes
- Member Comment settings—options for extracting comments declared within a class
- Function Comment settings—options for extracting comments related to function definitions (including operation definitions)

To extract source code comments from your source code:

1. On the Comments tab of the parse dialog, make selections for the source code object categories and fill in the fields, as needed.
2. Press the Extract Comments button. This action runs the comment extractor and automatically saves the current settings.

Comment Position Settings

The position settings indicate where you think the comments for that object are most likely to occur within the source code (above, below, or to the right of the object’s definition in the code).

The position specifications are cumulative. If you click all three position options for a comment (above, below, and to the right of an object definition), the comment extractor extracts all comments for that object that appear within the specified gap, no matter where they are positioned relative to the definition.

For example, by selecting options on the Comments tab, you can direct the comment extractor to:

- Look for all comments positioned above a member definition, as well as all comments positioned to the right of the definition.
- Ignore any comments found if there is a gap of more than one blank line between the comment and the object definition.

Gap Settings

The gap options determine the number of non-comment lines (containing either code or white space) that constitute a legal gap between a comment and its associated object. For example, a comment separated from a function by two lines would be associated with that function only if the gap setting is two lines or more; a gap setting of one line would not associate the comment with this function. If you set too large a gap, you may extract irrelevant comments.

The gap setting determines spacing for comments both below and above the target object, as indicated by the position setting. Different gaps can be set for classes, members, and functions.

The gap for functions is measured from the line on which the function appears. For example, the gap is measured from the second line in the following code fragment:
int
stock_item::getStockNo()
{

The gap for classes is measured from the line containing the name.

**Formatting Options**

You can select one of the following formatting options:

- **Copy Text Exactly**—Copies comment text exactly as it appears in the code, without reformatting.
- **Left Justify Text**—Reformats comments by left-justifying the text with a ragged right margin.
- **Fully Justify Text**—Reformats comments by justifying both left and right margins.

If you use this option, select the maximum line length allowed in the **Line Length** list. This number determines where line breaks occur.

**Evaluating and Improving Comment Extraction**

Unless you wrote the source code being parsed, you do not necessarily know where comments will be found. You may need to experiment with the option settings to see what combination yields the best results.

Each time you run the comment extractor, Ameos displays diagnostic statistics. These statistics indicate the percentage of user-defined classes and functions with associated comments, given your search specifications.

If the percentages are not satisfactory, you can rerun the comment extractor with new options to increase the percentage of comments found. Comment extraction replaces any comments in the semantic model with the latest ones found. Note that comments are extracted only from files that contain legal source, either header (*.h, *.h++, ...) or implementation files (*.cpp, *.cxx, *.C, ...).

You can also use the comment extraction program to check comment usage standards within an organization. For example, if every function should have a comment above its definition, make sure that you achieve a 100% rate of extracting comments for functions.

**Detection of Inappropriate Language in Comments**

If inappropriate language appears in any of the comments, reverse engineering replaces it with a random collection of and informs you of the number of words it replaced. Replacement characters are:

*#@?^%

**Increasing the Speed of the Parser**

If you wish to increase the speed of the parser, you can:

- Parse header files only.
- Switch off preprocessing.

By parsing header files only, you can increase the speed of the parser by ten to fifteen percent. Since the majority of the information required for building a model is extracted from header files, you can eliminate source files from parsing without losing a significant amount of information. However, by parsing header files only, you cannot include operation bodies during model generation (see “Microsoft Extensions” on page 39).

In addition, associations that represent a local variable of another class type that are declared within an operation body of another class do not appear on the generated diagrams. These associations can be created only if the source code containing the operation definitions is parsed.
You can also speed up the parser by not preprocessing the files to be parsed; however, if your code is complex, it may be essential to preprocess it.

To increase the likelihood of successful parsing without preprocessing:

- Add *.h to the File Pattern field on the Source Files tab.
  
  This action causes all header files associated with the specified target to be placed in the Parse List.
- Change the order of the files in the Parse List so that header files are parsed before the files that depend on them.

**Parsing Ada 95 Source**

To parse Ada 95 source:

1. Choose Code > Reverse Engineering > Reverse Engineer Ada95 Code. The Ada 95 parse dialog appears (Figure 3).

**Figure 3: Parse Dialog for Ada 95**

2. Prepare a parse list. There are three ways to do this:
   - Manually add files from the Directories list to the Parse List on the Source Files tab. Select the file(s), press the right arrow, then press Populate Parse List.
   - Specify the name of a file in the ObjectAda File field of the Source Files tab, then press Populate Parse List. (This file contains the path pointing to the Ada source to process, so you do not have to create a parse list separately.)
Specify a file pattern to read sets of files from the current directory. For example, to add .h and .cpp files, type “*.cpp *.h” in the File Pattern field on the Source Files tab, then press Populate Parse List.

Each of these actions populates the Parse List window on the Source Files tab with the files you have chosen.


After you have parsed your Ada 95 source to create a semantic model, you can generate the graphical model from it. Go to “Generating a Model from Parsed Files” on page 48 for information on generating a graphical model.

Other Parsing Topics

This section addresses parsing topics that apply to parsing Java, C++, and Ada 95.

Deciding Which Files to Parse

When you choose the source files to be parsed, you determine the content of the diagrams produced by Ameos. You can parse an individual source file, several selected source files, or all source files used to build the executables that form an entire system.

Processing several or all executables at once allows you to create a model for a particular part or all of the software system. Generally, you should parse as a group all related executables that share common data definitions. This action causes identical information for different executables to be diagrammed only once, rather than reproduced in identical diagrams and definitions for each executable.

The source files to be parsed can reside anywhere on the file system. They are not necessarily in the same directory. Although you can parse files that are incomplete or have syntax errors, you may receive warning messages.

If you inadvertently omit necessary source files from the list of source code files to parse, the generated model may be incomplete.

Ameos does not extract the source code from version control systems. If your code resides within a version control system, you must manually extract readable copies of the source into a directory where Ameos can process them. If you choose to run Ameos incrementally with such code, make sure that only changed files are copied over their previous instances.

Mixing Languages

When you create or update a model, you can mix languages. For example, you can parse C++ and Java source code files at once, or you can add information from Java source code files to a model that was created from C++ or Ada 95 source code files.

Using Incremental Mode

Select Incremental Mode (on the Parse Code tab) to parse only those specified files that have been modified since the last time they were parsed. The parser uses the file system modification date stamp to determine when the files were last modified.

When you select Incremental Mode, a file is parsed only if one of the following apply.

- It has been modified since last being parsed.
- It failed to pass prior parsing correctly.
It includes an `#include` file that has been re-parsed incrementally (for C++)

It has just been added to the source file list.

Generally, you should use incremental mode for parsing files except when:

- Many files have changed.
- A frequently included file has changed.

In these cases, parsing in non-incremental mode is usually faster. However, the results of parsing incrementally and non-incrementally are the same.

### Parsing Source Code Containing Embedded Code

If the dialect of source code used in your organization supports the embedding of SQL, assembler, or other languages, you can configure the reverse engineering parser to search for and ignore the embedded code. This action allows you to reverse engineer the original source code without special preprocessing that would create extra code you may not want in your source code.

To tell the reverse engineering tool to ignore embedded code, you specify in a file the starting and ending tokens that surround the code to be ignored. The tokens should be used by the embedded language to signify sections of embedded code, such as declaration sections and SQL commands. The parser then looks for the user-specified tokens in the source code and ignores all code found between the matching starting and ending tokens.

For example, when embedded in C++ code, some dialects of SQL use the token `EXEC` to indicate the beginning of an embedded SQL command, and a semicolon (`;`) to indicate the end of the embedded command. By specifying `EXEC` as the start token and a semicolon as the end token, you instruct the reverse engineering tool to search for and ignore each embedded SQL command.

To specify tokens, you add them to a file that the `re_token_file` preference variable points to.

### Token Search Patterns

The token strings you choose and the order in which you enter them in the file affect the results of the parser search. Precedence of start tokens is by the order in which the token pairs occur in the file.

For example, suppose your token definition file contains these token specifications:

```sql
"EXEC SQL BEGIN DECLARE SECTION;" "EXEC SQL END DECLARE SECTION;" "EXEC" ";
```

When parsing the following SQL code embedded in C++ code, the parser first finds and ignores all of the SQL declarations in the declaration section; then it finds and ignores the embedded SQL command.

```cpp
// Some C++ code here...
EXEC SQL BEGIN DECLARE SECTION;
  // Some SQL declarations here...
EXEC SQL END DECLARE SECTION;
// More C++ code here...
EXEC <embedded SQL command>;
// Additional C++ code here...
```

However, suppose you entered the tokens in the file in the reverse order:

```sql
"EXEC" ";"
"EXEC SQL BEGIN DECLARE SECTION;" "EXEC SQL END DECLARE SECTION;"
```

In this case, the parser finds the first `EXEC` token at the beginning of the `BEGIN DECLARE SECTION` in the preceding code and ignores code only until it encounters the semicolon token at the end of the `BEGIN DECLARE SECTION` on the same line. It then stops ignoring code and tries to parse the SQL declarations between the first end token and the next `EXEC` token in the `EXEC SQL END DECLARE SECTION`.
Likewise, the reverse engineering parser would not be able to identify a nested token pattern, such as this:

```cpp
// Some C++ code here...
EXEC <embedded SQL command>
    EXEC <embedded SQL command>;
    // More SQL code
;
// More C++ code here...
```

The parser would ignore all code between the first `EXEC` token and the first semicolon (`;`) end token it finds, thus ignoring the nested `EXEC` as well. Having ignored the second `EXEC` token, the parser would not look for its paired semicolon end token either. Thus, the parser would try to parse the SQL code between the two semicolons.

The preceding examples show how important it is to choose appropriate tokens and enter them in the token file in an appropriate order, so the parser parses the source code correctly, while ignoring the embedded language code.

**Specifying Embedded Tokens**

You can specify as many pairs of tokens as needed to identify all of the embedded code to be ignored. Token matching is case sensitive. User-specified tokens must exactly match the tokens used by the embedded language, or the parser cannot identify the code to ignore.

To specify the start and end tokens:

1. Create a file (for instance, `embedded_tokens`) to contain the token specifications.
2. Add a new line to the file for each pair of start and end tokens you want the parser to use.
   - Enclose each token within double quotes and separate the start token from the end token with a single space. For example:
     ```
     "EXEC SQL BEGIN DECLARE SECTION;" "EXEC SQL END DECLARE SECTION;"
     "EXEC" ";"
     ```
3. In your preferences, set the `re_token_file` variable to point to the file containing the token specifications.
   - For example, if your specifications are in the `embedded_tokens` file, set the value of the `re_token_files` variable as follows:
     ```
     re_token_file=/usr/Ameos/template/embedded_tokens
     ```

When you run the parser, it looks for the file and the tokens you have specified.

**Common Parsing Problems**

You will have problems when parsing source code if:

- There are locks on the semantic model.
- Some files fail preprocessing.
- The parser cannot locate required source or header files.
- Case and spelling of source file names is incorrect.
- The directory containing the source files is not specified as a path.
- Paths to files are specified incorrectly.
Checking Semantic Model Locks (Ada 95 and C++)

Reverse engineering uses locking to prevent users from simultaneously performing operations on the same data in the semantic model database. For example, you cannot have two parsers executing at the same time, updating the same semantic model.

Because of the massive number of records stored in the semantic model and the large numbers of contiguous records that are removed and replaced during parsing, it is not appropriate to lock individual records. Instead, complete tables are locked exclusively by all reverse engineering processes.

Semantic model locks do not affect users of the Ameos repository, which is always available to multiple readers and writers.

To find out the lock status of a semantic model, choose Code > Reverse Engineering > Check RE Semantic Model Locks from the desktop.

The command removes any redundant locks and reports active locks by listing the name of the person and the machine holding the lock, as well as the time the lock was applied.

Checking Semantic Model Consistency (Ada 95 and C++)

It is possible for the semantic model’s indexes to become corrupted if a process that is writing it is killed before completing the task. Also, if the data on the disk is corrupted due to a disk crash, for example, the data fields in the semantic model may be invalid or out of range.

If you suspect that the semantic model has been corrupted, check it by choosing Code > Reverse Engineering > Check RE Semantic Model Consistency from the desktop.

This results in a message being displayed stating whether the semantic model can be used or if it has been corrupted. If it is corrupted, you should destroy the semantic model by rerunning the parser on all the source files in non-incremental mode. This action removes the current semantic model and rebuilds an entirely new one.

Generating a Model from Parsed Files

Pressing Make Model on the parse dialog creates a graphical model from the semantic model produced by the parser.

The generated model is stored in the repository for the current Ameos project and system. When you generate a model where a particular class already exists, you can regenerate the class and/or retain user modifications.

To generate a model:
1. From the Code menu on the Ameos desktop, choose one of the Reverse Engineer commands from the Code menu.
2. On the Options tab, specify desired options.
3. To generate the model, press Make Model.
   Progress and results display in the Ameos desktop message hog.

Retaining User Modifications if a Class Exists

The following information pertains to using the If Class Already Exists option of the Options tab.

If a class exists, and you do not check Regenerate, the existing class table, diagram, and annotations remain as they were before you pressed Make Model.
If you select **Regenerate**, but do not select **Retain User Modifications**, the **Make Model** command:

- Completely rewrites the class table and removes any added rows that are not in the source code. The command also blanks out any extra columns.
- Completely rewrites the class diagram. Therefore, any modifications to all the existing `<classname>_RE_[AGGR | ASSOC | NESTED | INH]` diagrams will be lost.
- Updates annotations, or creates them if they do not exist.

If you select both **Regenerate** and **Retain User Modifications**, the **Make Model** command:

- Updates the class table. Any user modifications to places that were not created by the reverse engineering tool will still exist. If you have modified parts of the table that the reverse engineering tool has created, these will be overwritten by the correct values in the source code.
- Creates a class diagram if it does not already exist. If the diagram does exist, it is neither recreated nor updated.
- Updates annotations, or creates them if they do not exist.

### Model Elements Generated by Reverse Engineering

#### Package Diagrams

During the reverse engineering process, Ameos searches for each package that contains one or more classes. If such packages exist, Ameos creates the diagrams noted below.

- `<package>_DEPEND_RE` – Dependencies (described below)
- `<package>_CONT_RE` – Package hierarchy. For each package that contains one or more classes, a diagram called `<package>_CONT_RE.uclassd`, which shows this by graphical containment, is generated.
- `PACKAGE_HIERARCHY_RE` – The package hierarchy for the entire system

#### Package Dependency Diagrams

Package dependency files (`<package>_DEPEND_RE`) are created for each package that contains a class referencing classes from other packages.

*Example code (Java)*:

```java
package myPackage;
import otherPackage.*;
class myClass extends otherClass // otherClass is from otherPackage
{
}
```

*Generated diagram for myPackage_DEPEND_RE*:

![Package Dependency Diagram](image)

#### Package Containment Diagrams

Package containment diagrams (`<package>_CONT_RE`) are created for each package containing classes and/or packages.

*Example code (Java)*:

```java
package myPackage;
```

Advanced Modeling Concepts
import myPackage.subPackage*;

class myClass {
}

Diagram for myPackage_CONT_RE:

```
myPackage
   
subPackage
   
myClass
```

Class Diagrams

During the reverse engineering process, Ameos searches for a class’s relationships. If these relationships exist, Ameos creates the class diagrams noted below. In each case the class is a visual representation of relationships that were captured from the code.

Note: Not all elements are generated for all languages.

- `<class/interface>_INH_RE` – Inheritance (generalization) relationships (see below)
- `<class/interface/package>_ASSOC_RE` – Associations (page 51)
- `<class>_AGGR_RE` – Aggregations (page 52)
- `<class/interface>_DEPEND_RE` – Dependencies (page 52)
- `<class>_IMPL_RE` – Interface implementations (page 53)
- `<class/package>_NESTED_RE` – Nested relationships. When reverse engineering finds a nested relationship, it creates an annotation item (page 53)

When you use reverse engineering, Ameos places the generated diagrams in the `uclassd_files` subdirectory of the current system/project directory and includes them in the list of files you can load into the class editor.

**Class Diagrams For Generalizations**

Ameos creates a class diagram for each distinct generalization hierarchy that exists in the classes captured with reverse engineering. Each generated diagram is a simple drawing using orthogonal arcs of the subclasses inheriting from the superclass, as shown in Figure 4.

**Figure 4: Generalization Diagram**

```
Shape

Superclass

Circle  Rectangle

Subclasses
```

Inheritance information is placed in the repository as an annotation specified on the inheritance link joining the subclass and the superclass.

Reverse engineering can create a generalization diagram even if you capture only a subclass. For example, if you capture just the `Circle` class shown in Figure 4, Ameos creates a file named `Shape_INH.uclassd`, showing `Circle` and `Shape` in an inheritance relationship. If you then capture just
Rectangle, the Shape_INH.uclassd file is enhanced to show both subclasses. However, the Shape class attributes and operations are not defined in the repository until you capture the class.

**Class Diagrams for Associations**

Ameos creates association diagrams when:

- An attribute points to another class (as in UsesB3 in the example code shown below)
- An attribute is the return type of a member function or a parameter of a member function (as in the second constructor method for A)
- A local variable of another class type is declared within any member function (as in A::A in the example code below)

Association diagrams created by reverse engineering show role names and multiplicity, as described in:

- "Roles in Reverse Engineering Diagrams" on page 53
- "Multiplicity in Reverse Engineering Diagrams" on page 53

**Example code (C++):**

```cpp
class B1
{
};
class B2
{
};
class B3
{
};
class A
{
    public:
        A();
        A(B1 &usesB1);
    private:
        B3 *UsesB3;
        B2  hasB2;
};
A::A()
{
    B2 usesB2;
}
```

**Diagram for A_ASSOC_RE:**

```
  A
 /|
/ | 
B1|B3|B2
```

Role names

```
UsesB3
```

```
hasB2
```
**Class Diagrams for Aggregations**

Ameos creates aggregation diagrams from source code files when:

- A class contains attributes that are instances of another class (if the attribute is a pointer to an object, the relationship is considered an association).
- An attribute is typed as being an array of elements, each of which is another class.
- An attribute's type is a recursive pointer to the class in which it is defined, as shown in the following C++ code example:

```c++
class IntList {
    IntList *next;
    int      data;
};
```

Ameos/UML supports both aggregations and composition aggregations. Reverse engineering can create both types of aggregations. Composition aggregations appear as a solid diamond; aggregations appear as a hollow diamond.

Aggregation diagrams created by reverse engineering show role names and multiplicity, as described in:

- "Roles in Reverse Engineering Diagrams" on page 53
- "Multiplicity in Reverse Engineering Diagrams" on page 53

**Example code (C++):** (See code shown in "Class Diagrams for Associations" on page 51)

**Diagram for A_AGGREG_RE (with role name):**

```
Display mark for a composition aggregation.

Role name hasB2
```

**Class Diagrams for Dependencies**

Class dependency diagrams (<class>_DEPEND_RE) are generated for each attribute with a own class as type.

**Example code (Java)**

```java
class myClass {
    otherClass data;
};
```

**Diagram for otherClass_DEPEND_RE:**

```
myClass

otherClass
```
**Class Diagrams for Interface Implementations**

Class implementation diagrams (\texttt{<class>_IMPL\_RE}) are generated when a class implements an interface.

*Example code (Java):*

```java
class myClass implements otherInterface {
    
};
```

*Diagram for otherInterface\_IMPL\_RE:*

```
myClass

«interface»  
otherInterface
```

**Class Diagrams for Nested Relationships**

Reverse engineering creates a nested diagram when it finds:

- C++ namespaces
- Java packages

In Ameos/UML, when a class is nested within another class, reverse engineering does not create a diagram, because this action breaks the UML syntax checks.

Instead, reverse engineering represents these relationships by using the Enclosing Scope item of the Class Definition note, described in "Annotations" on page 57.

When creating nested diagrams, reverse engineering:

- Draws C++ namespaces and Java packages as packages in class diagrams.
- Completes the Enclosing Scope item of the Class Definition note.
- Does not create tabular information for these classes.

**Roles in Reverse Engineering Diagrams**

Role names automatically appear on association and aggregation diagrams created by reverse engineering. The role name is taken from an attribute name, where the attribute is mapped into an aggregation or association.

Role names are placed in the repository as contexts on links.

If you do not want to show role names on your diagram, you can:

- Use a filter to hide them from view (see the online help – “About diagram filters” on the Index tab – for information on applying filters).
- Select and then delete them from the diagram.

The diagram shown in "Class Diagrams for Aggregations" on page 52 shows a role name on an aggregation diagram created by reverse engineering. For a more detailed example, see “Example code” below.

**Multiplicity in Reverse Engineering Diagrams**

Multiplicity display marks automatically appear on association and aggregation diagrams created by reverse engineering if the multiplicity display mark is set for display.
Information about multiplicity is placed in the repository as an annotation on the link.

**Rules for determining multiplicity**

Reverse engineering uses the following rules to determine multiplicity for aggregations:

- An aggregation that is an array (but not an array of pointers) is created with Many multiplicity.
- An aggregation created from a recursive pointer is created with Optional Multiplicity.
- All other aggregations are created as Exactly One multiplicity.

All associations are created with the default multiplicity (Exactly One), unless the association is a pointer to an array or a double pointer (or triple pointer or more). These associations are created as Many multiplicity.

If reverse engineering finds more than one identical association or aggregation within a class, it merges their multiplicity. The basic rules for merging are:

- Many + <anything else> = Many
- One or More + Exactly One = Many
- One or More + One or More = One or More
- One or More + Numerically Specified = Many
- Numerically Specified + Optional = Many
- Numerically Specified + Exactly One = Many
- Optional + Exactly One = One or More
- Exactly One + Exactly One = Numerically Specified (with a value of 2)

In addition, if one role name has less than three characters and the other has three or more characters, reverse engineering uses the longer name.

**Example code**

```java
class E
{
};

class F
{
    E myE[100];
};
```

**Aggregation diagram generated showing roles and multiplicity:**

Notice that the role name, `myE`, appears on the end of the link opposite the class from which the role name was extracted (`F`).
Interfaces in Reverse Engineering Diagrams

Reverse engineering generates interfaces from Java source files. Interfaces appear as class symbols with a stereotype of <<Interface>> on Ameos/UML association, generalization, and nested diagrams. For each class, the stereotype of the Extensibility Definition note is set to interface.

Viewpoints in Reverse Engineering Diagrams

When reverse engineering creates an Ameos/UML class diagram, it attaches the viewpoints to nodes on diagrams, as described in Table 1. Since viewpoints are invisible, they do not alter the appearance of the diagram.

For additional information about nodes, see Node Subtype in the "Understanding the Persistent Data Model" chapter of Object Management System. For additional information about viewpoints, see the online help – click “About display marks” on the Index tab.

Table 1: Viewpoints on Reverse Engineered Diagrams

<table>
<thead>
<tr>
<th>Viewpoint</th>
<th>Attached to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Aggregation</td>
<td>All nodes on <code>&lt;class&gt;_AGGR_RE</code> diagrams</td>
</tr>
<tr>
<td>All Generalization</td>
<td>All nodes on <code>&lt;class&gt;_INH_RE</code> diagrams</td>
</tr>
<tr>
<td>All Association</td>
<td>Root node of any <code>&lt;class&gt;_ASSOC_RE</code> diagrams</td>
</tr>
<tr>
<td>Package containment</td>
<td>All nodes, except the root node, on <code>&lt;class&gt;_NESTED_RE</code> diagrams</td>
</tr>
</tbody>
</table>

Tabular Class Content

Ameos creates tabular information in the property window of each class that is created during the reverse engineering process. Table 2 shows the tabular information produced by the reverse engineering process.

Table 2: Tabular Class Content Provided by RE

<table>
<thead>
<tr>
<th>Table Section/Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td></td>
</tr>
<tr>
<td>Class members</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Attribute name</td>
</tr>
<tr>
<td>Type</td>
<td>Attribute type</td>
</tr>
<tr>
<td>Default Value</td>
<td>Initialization (Java only)</td>
</tr>
<tr>
<td>Analysis Items</td>
<td>(Provided for C++ and Java source code files only)</td>
</tr>
<tr>
<td>Visibility</td>
<td>(Public, Private, or Protected)</td>
</tr>
<tr>
<td>Class Attr?</td>
<td>True/False. Set to True if the attribute is static.</td>
</tr>
<tr>
<td>C++ Items</td>
<td>True/False</td>
</tr>
<tr>
<td>Const?</td>
<td>C++ visibility</td>
</tr>
<tr>
<td>Visibility</td>
<td>Public/Protected/Private</td>
</tr>
<tr>
<td>Volatile?</td>
<td></td>
</tr>
<tr>
<td>Java Items</td>
<td>Public/Protected/Private. Java default visibility is private</td>
</tr>
<tr>
<td>Visibility</td>
<td>True/False</td>
</tr>
<tr>
<td>Final?</td>
<td></td>
</tr>
</tbody>
</table>
Reverse engineering creates tabular class content for the Shape class, as shown below. (Note that the table in the figure shows only a partial view of the Analysis Items for the Shape class.)

**Tabular class content generated for class Shape:**

```plaintext
Example Code:

class Shape
{
public:
    Shape(int x, int y);
    virtual void draw() = 0;
    virtual void erase() = 0;
protected:
    int locx, locy;
};
```

---

### Table 2: Tabular Class Content Provided by RE (Continued)

<table>
<thead>
<tr>
<th>Transient?</th>
<th>True/False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volatile?</td>
<td>True/False</td>
</tr>
</tbody>
</table>

| Ada 95 items   | (no items are generated) |

#### Operations

<table>
<thead>
<tr>
<th>Class Members</th>
<th>Operation name</th>
<th>Operation arguments</th>
<th>Operation return type</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Analysis Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Provided for C++ and Java source code files only.)</td>
</tr>
<tr>
<td>Visibility</td>
</tr>
<tr>
<td>Class Op?</td>
</tr>
<tr>
<td>Abstract?</td>
</tr>
<tr>
<td>Throws</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C++ Items</th>
<th>True/False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visibility</td>
<td>Public/Protected/Private</td>
</tr>
<tr>
<td>Virtual?</td>
<td>True/False</td>
</tr>
<tr>
<td>Inline?</td>
<td>True/False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Java Items</th>
<th>True/False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visibility</td>
<td>Public/Protected/Private. The Java default visibility is private</td>
</tr>
<tr>
<td>Final?</td>
<td>True/False</td>
</tr>
<tr>
<td>Native?</td>
<td>True/False</td>
</tr>
<tr>
<td>Synchronized?</td>
<td>True/False</td>
</tr>
</tbody>
</table>

| Ada 95 items   | (no items are generated) |
## Annotations

During the reverse engineering process, when Ameos finds the appropriate information, it creates annotations for each class. Table 3 shows annotations created for Ameos/UML.

### Table 3: Ameos/UML Annotations

<table>
<thead>
<tr>
<th>Note/Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object</strong></td>
<td><em>(C++, Java)</em> Contains source code comments for a class, attribute, or operation.</td>
</tr>
<tr>
<td><em>(Appears in Note Description dialog)</em></td>
<td></td>
</tr>
<tr>
<td><strong>C++ Declarations</strong></td>
<td><em>(C++)</em> Contains locally defined typedefs, friend classes, or friend functions, or source for operations</td>
</tr>
<tr>
<td><em>(Appears in Note Description dialog)</em></td>
<td></td>
</tr>
<tr>
<td><strong>Class Definition</strong></td>
<td><em>(C++)</em> Indicates nested classes. This item contains the name of the parent class that the annotation class is nested in.</td>
</tr>
<tr>
<td><strong>Enclosing Scope</strong></td>
<td><em>(Java)</em> Indicates the package in which the class resides.</td>
</tr>
<tr>
<td><strong>Abstract Class</strong></td>
<td><em>(C++, Java)</em> Indicates that a class is abstract.</td>
</tr>
<tr>
<td><strong>Class Java Definition</strong></td>
<td><em>(Java)</em> Indicates that the class was declared “final” when implemented in Java.</td>
</tr>
<tr>
<td><strong>Final</strong></td>
<td><em>(Java)</em> Indicates Java visibility (public, private, protected). If visibility is set in the Class Definition, this setting overrides it.</td>
</tr>
<tr>
<td><strong>Visibility</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Class Ada 95 Definition</strong></td>
<td><em>(no items are generated)</em></td>
</tr>
<tr>
<td><strong>C++ Inheritance Definition</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Inheritance is Virtual</strong></td>
<td><em>(C++)</em> Indicates if the C++ inheritance is virtual.</td>
</tr>
<tr>
<td><strong>Inheritance Visibility</strong></td>
<td>Indicates if the inheritance is public, protected, or private</td>
</tr>
<tr>
<td><strong>Extensibility Definition</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Stereotype</strong></td>
<td><em>(C++, Java)</em> Specifies a stereotype name for an object.</td>
</tr>
<tr>
<td><strong>Role Definition</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Aggregation Type</strong></td>
<td><em>(C++)</em> Indicates if the relationship is an aggregation or a composition aggregation.</td>
</tr>
</tbody>
</table>
Updating a Class from the Editors

You can use reverse engineering commands to update a class or create a generalization hierarchy from the class diagram editor. (See Table 4.) These commands, which are available on the UML menu of the class diagram editor, are greyed out except when a semantic model exists.

Table 4: RE Commands for Updating Classes

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML &gt; Attributes and Operations &gt; Construct From Reverse Engineering</td>
<td>Gets the definition of the selected class from the source code files and updates the attributes and operation for the selected class.</td>
</tr>
<tr>
<td>UML &gt; Generalization Hierarchy &gt; Create Subclass Hierarchy from Reverse Engineering</td>
<td>Builds a generalization hierarchy consisting of classes beneath the selected class.</td>
</tr>
<tr>
<td>UML &gt; Generalization Hierarchy &gt; Create Superclass Hierarchy from Reverse Engineering</td>
<td>Builds a generalization hierarchy consisting of classes that the selected class inherits from.</td>
</tr>
<tr>
<td>UML &gt; Generalization Hierarchy &gt; Create Whole Hierarchy from Reverse Engineering</td>
<td>Builds a generalization hierarchy consisting of classes above and below the selected class.</td>
</tr>
</tbody>
</table>

Table 3: Ameos/UML Annotations (Continued)

| Multiplicity | See “Multiplicity in Reverse Engineering Diagrams” on page 53. |
6 Importing and Exporting XMI With Ameos/UML

This chapter discusses import and export of XMI streams for Ameos/UML models. XMI, short for XML Metadata Interchange, is an XML-based standard defined by the Object Management Group (OMG) consortium that facilitates exchange of UML models between various tools.

Ameos supports both import and export of XMI 1.2 streams based on the UML 1.4 metamodel. Both import and export can be customized in order to allow bridging of additional user-defined extensions (which are part of the XMI standard).

XMI export is implemented with an ACD template (similar to a code generation template). XMI import uses XSLT (Extensible Stylesheet Language Transformation) to translate XMI into Ameos's internal common input format (called DDL), which is then used to create UML diagrams.

**Note:** It is not possible to bridge a model from one tool to another without losing diagram layout information. **And Ameos does not yet support the upcoming XMI diagram interchange information (like diagram layout).** However, when generating a UML model for an XMI stream, Ameos does perform an "auto-layout" to generate the supported diagram types.

**Prerequisites**

All software required to import or export XMI is part of the Ameos package. However, you need to be licensed as an Ameos Developer or Ameos Analyst to access the Ameos XMI facilities.

Importing an XMI stream uses Xalan-C++, an XSLT processor from the Apache XML Project (see [http://xml.apache.org/xalan-c/index.html](http://xml.apache.org/xalan-c/index.html)). Most recent versions of Xalan-C++ (and the corresponding XML Parser Xerces-C++) are distributed with the Ameos package. At the moment, these are Xalan-C++ version 1.5 and Xerces-C++ version 2.2.0.

The main advantages of using Xalan-C++/Xerces-C++ are that it provides:

- Robust implementation of all W3C (World Wide Web Consortium) recommendations for XSLT and XPath
- Platform independency – runs on all platforms supported by Ameos (Windows, Linux, Solaris, HP-UX)
- A complete API for performing transformations in C++ code

**Compatibility with Language Specifications**

Both XMI import and export are compatible with OMG’s XMI definition 1.2 and the UML 1.4 metamodel. Part of the distribution is the XML document type definition (DTD) that defines that standard.

**Compatibility with Previous Code Generators**

The XMI version supported in 9.x is not compatible with the XMI 1.0 streams supported in the 8.x releases of StP.
Exporting XMI

XMI export can be invoked in two ways:

1. By choosing Code > XMI > Generate XMI by ACD Template. This opens the script manager, with XMI highlighted. From there you choose Run Script from the context menu to generate XMI as shown in Figure 1.

   **Figure 1: Using the Script Manager to Export XMI**

2. From context menus in the desktop’s tree browsers for diagrams, packages, or classes (choose Generate XMI for <object> by ACD Template)

The following diagram types (including all contained objects and annotations) are supported:

- Class diagrams
- Sequence diagrams
- State diagrams

When exporting an XMI stream, Ameos also copies the UML 1.4 DTD file into the same directory. The presence of the DTD file allows verification of the XMI file with various XML tools.

Importing XMI

Incremental update is supported for tagged values only. This will not change diagrams, just the tagged values (see “Other import capabilities” on page 63). Importing an XMI stream with diagram generation is not incremental and will simply overwrite data already present in the model in case of namespace overlap.

The following diagram types (including all contained objects and annotations) are supported:

- Class diagrams
- Sequence diagrams
- State diagrams
XMI import is invoked by choosing **Code > Reverse Engineer XMI > Import XMI Model**. This opens the **Parse XMI** dialog (Figure 2).

**Figure 2:  Parse XMI Dialog**

In the **Parse XMI** dialog, the following items can be defined:

- **On the Parse XMI tab:**
  - The XMI file to be imported
  - The target system it should be loaded into.
  - The name of a different XSL file (optional). For example, you can specify an XSL file that maps additional user defined extensions.

  Use the **Parse XMI File** button on this tab to import the XMI file. Once importing is accomplished, you can specify items on the **Options** tab to select which diagrams should be created.

- **On the Options tab:** Generate options, such as **Class Diagrams**. For an incremental update of tagged values, you can choose **Update tagged values in model only**. This will not change diagrams, just the tagged values (see "Other import capabilities" on page 63).

Ameos uses its own sophisticated algorithm to perform "layouting." Nevertheless, the layout of generated diagrams can be very different from the layout of source diagrams.

**Class Diagrams**

When importing an XMI stream, Ameos can generate the types of class diagrams noted below.

- **Generalization diagram(s).** Generalization diagrams are used to express parent-child relationships between classes/packages. In an XMI stream, such a relationship is represented by the tag UML:Generalization. Generated class diagrams are named `<name>_INH_RE.uclassd`.

- **Aggregation diagram(s).** Aggregation diagrams are used to express class-attribute relationships. In an XMI stream, such a relationship is represented by the tag UML:Attribute scoped in a tag UML:Class. Generated class diagrams are named `<name>_AGGR_RE.uclassd`.

- **Association diagram(s).** Association diagrams are used to express class-class relationships. In an XMI stream, such a relationship is represented by the tag UML:Association. Generated class diagrams are named `<name>_ASSOC_RE.uclassd`. 
• **Dependency diagram(s).** Dependency diagrams are used to express class-class dependency. In an XMI stream, such a relationship is represented by the tag UML:Dependency. Generated class diagrams are named `<name>_DEPEND_RE.uclassd`.

• **Implementation diagram(s).** Implementation diagrams are special case of generalization diagrams. In an XMI stream, they are represented by the tag UML:Generalization. The only difference is that parent is an interface (represented by tag UML:Interface). Generated class diagrams are named `<name>_IMPL_RE.uclassd`.

• **Containment diagram(s).** Containment diagrams are used to represent graphical package containment. A separate diagram is created for each package (XMI tag UML:Package), which contains other classes/packages. Generated class diagrams are named `<name>_CONT_RE.uclassd`.

  **Note:** Only one-level containment is represented, i.e., only immediate child classes/packages are included in the parent package. If a child package contains other classes/packages, they are represented in separate containment diagrams.

• **Package overview diagrams.** Package overview diagrams are used to represent the package hierarchy in the entire XMI stream. The package hierarchy is modeled as a tree, with dependency links between parent/child packages. There can be at most one diagram of this type, named `PACKAGE_HIERARCHY_RE.uclassd`.

**Class member information.** Attributes and operations found in the XMI stream are not displayed in any diagrams but just created in the repository. Note: They can be displayed at any time using the classes **Select Members to Display** context menu.

## Sequence diagrams

When importing an XMI stream, Ameos searches for the XMI tag UML:Collaboration. For every such tag Ameos generates one sequence diagram, and places all actors, objects, and messages that are scoped in this collaboration inside the diagram.

The following sequence diagram objects are supported:

• Actors (XMI tag UML:Actor)
• Objects (XMI tag UML:ClassifierRole)
• Messages (XMI tag UML:Message)

No special diagram layout is performed – all actors/objects are placed horizontally from left to right, in the same order as defined in the XMI stream. The diagram is named `<name>.usequenced`, where `<name>` is the name of the XMI tag UML:Collaboration.

  **Note:** There is no way to differentiate between active objects and passive objects, so all imported objects are interpreted as passive objects.

## State Diagrams

When importing an XMI stream, Ameos searches for the XMI tag UML:StateMachine. For every such tag, Ameos generates one state diagram, and places all states and state transition links that are part of this state machine inside the diagram. In addition, separate state diagrams are generated for each composite state.

The layout of state diagrams is the same as for class diagrams. States are represented as nodes in a graph; state transitions are represented as links connecting corresponding nodes; and the graph has a layout determined by the general layout algorithm. The only difference between the layout of class and state diagrams is that, for the latter, there is no decomposition into connected components.

The following types of states are supported:

• Initial states (XMI tag UML:InitialState)
- Final states (XMI tag UML:FinalState)
- Composite states (XMI tag UML:CompositeState)
- Simple states (XMI tag UML:SimpleState)
- Pseudo states (XMI tag UML:PseudoState)

In addition to state diagrams, the following elements are generated in the repository (and can be seen in the state’s property sheet):
- Parent state(s)
- Entry actions
- Exit actions
- Activities

**Other import capabilities**

In addition to importing a UML model from an XMI stream, Ameos offers the ability to update tagged values in an existing model. In this mode, mapping between tagged values and objects in the Ameos repository is done via the object’s GUID. This means that the input XMI stream must be exported from the same Ameos system.
Ameos provides the ability to create baselines for Ameos/UML systems/subsystems and an interface to configuration management (CM) software.

This chapter describes:

- Creating, restoring, and comparing baseline systems (immediately below)
- Using version control (page 68)
- Using external CM systems for version control (page 68)

Creating, Restoring, and Comparing Baseline Systems

The baselining capabilities of Ameos allow you to:

- Create a baseline—Save the state of your Ameos system, or Ameos/UML subsystem, at various points during the development process.
- Restore a baseline—Restore the saved baseline state into Ameos as a separate system in order to manually check what has changed and possibly revert to the previously baselined version.
- Manage the baselines—Use the Ameos version control interface for supported external configuration management (CM) systems, such as CVS, to manage the various baselines you have created.
- Compare two baselines to obtain a list of changes between two versions.

Note that you can perform an Ameos baseline operation with or without an integrated version control/configuration management (CM) system. Also note that version control for individual files within an Ameos system is not supported. However, baselining and version control for Ameos/UML subsystems is supported.

A Typical Scenario

Assume that several users are working simultaneously on a model. The project administrator wants to make sure that, if undesirable changes are made to the model, it will be easy to revert to a state where the model was still valid. The project administrator uses the Ameos baseline and version control capabilities to accomplish this goal.

Here is what might be involved:

1. The project administrator creates a baseline of the system on a regular basis.
2. The project administrator uses the Ameos interface to check the baseline file into a CM system. The CM system keeps track of date and log information about the baseline file(s).
3. A problem occurs: someone made undesired changes to the model.
4. The project administrator checks the most recent version out of the CM system.
5. The project administrator restores the baseline, overwriting the current system. The problem is solved.
Creating a Baseline

To create a baseline:

1. Select one of the following commands from the desktop’s System menu, as appropriate:
   - Baselines > Baseline Current System
   - Model Management > Baseline Subsystem
2. In the Baseline Current System/Subsystem dialog, enter or make selections as described in “Baseline Current System/Subsystem Dialog” (immediately below) and click OK.
   
   Ameos checks for locks, stores the repository information, and archives it all into a zip file, as described in “How Ameos Creates the Baseline” below.

Baseline Current System/Subsystem Dialog

Figure 1 shows the Baseline Current System dialog. (Note: This dialog shows CVS commands, indicating that the CVS CM system has been set up for use with Ameos.) The Baseline Subsystem dialog is nearly identical, but includes a pick list of subsystems in your model and an option to lock the subsystem after baselining.

Figure 1: Baseline Current System Dialog

How Ameos Creates the Baseline

A baseline is a snapshot of a system. Therefore, no users should be working on a system that is being baselined. When a baseline is being created, Ameos checks to see if there are any locks on the system, which indicates that there are active users. If locks are detected, the baselining stops. This behavior can be switched off with the Ignore lock(s) options in the Baseline Current Subsystem/Current System dialog.

If the system uses a Sybase repository, the repository is “dumped” via a Sybase dump function into a temporary directory within the Ameos system directory structure. This is a time-saving feature, as it is quicker to restore a Sybase repository from a dump than to rebuild it from the Ameos system files. For Jet systems, the repository information is stored in a file within the system directory, so a “dump” step is unnecessary.
The system directory (including the Sybase repository dump and/or lock information file) is archived into a single file, using `zip.exe`, a public-domain zip utility provided with Ameos. The file name contains the name and directory location you’ve specified and always ends in `.zip`. If you specify a name without the `.zip` extension, it is added automatically.

**Notes on Creating Baselines**

- When you create a baseline, the baseline contains only files relevant to Ameos. These files are the required diagram/table/annotation files, a few administrative files, and the repository.

  If you want to add additional files, you can provide a list of patterns in the `baseline_include` ToolInfo variable. For example, to add all `.rtf`, `.html`, and `.svg` files generated from some reports, the ToolInfo variable should look like this:

  _On Windows:_
  ```
  baseline_include=*.rtf;*.html;*.svg
  ```

  _On UNIX:_
  ```
  baseline_include=*.rtf:*.html:*.svg
  ```

  If you want all files to be added to the baseline archive, the ToolInfo variable should look like this:

  ```
  baseline_include=*  
  ```

  In addition, there is a ToolInfo variable `baseline_exclude`, which can contain a similar list of files to be excluded. Excluding files takes precedence over including files.

- You can customize your baseline setup by setting the `baseline_path` variable in your preferences to any valid directory path. If you’re using an external CM system, `baseline_path` is typically set to the CM system’s workspace. For example:

  ```
  baseline_path=C:/CVSWorkspace/AmeosBaselines
  ```

  If the variable is set, the variable-specified baseline path will appear as the default path in the:

  - **Save Directory** field of the Baseline Current System dialog
  - **Baseline file** field of the Restore Selected Baseline System dialog

  If the `baseline_path` variable is not set, the defaults for both of these fields is `<projdir>/baseline`. If a baseline subdirectory does not exist in your `<projdir>`, Ameos will create it automatically the first time a baseline is created (see "Creating a Baseline" on page 65).

**Restoring a Baseline**

To restore a baseline for an entire system or a subsystem:

1. From the desktop’s System menu, select either of the following:
   - Baselines > Restore Selected Baseline System
   - Model Management > Restore Subsystem

   The Restore Selected Baseline System or Restore Subsystem dialog appears.

2. In the **Baseline file** field, enter or browse and choose the name of the baseline file, including the full directory path.

3. Enter additional information into the following fields (only if restoring the complete system):
   - **Restore to directory** —Name of the full path to the directory where the baseline file is to be restored, typically your Ameos project directory
   - **Restore System name**—Name of the system into which the restored baseline file is to be saved

4. Click OK.

   If the system to be created already exists, the existing system directory is overwritten with the restored files.
Otherwise, Ameos creates a new system with the specified name, to which the unzipped files are transferred, overwriting any empty default files. For restored Jet systems, the extracted .mdb file is renamed as the new system name. For restored Sybase systems, the extracted Sybase dump files are loaded into the new Sybase database that was created.

When Ameos completes the restore operation, the system is ready to use.

**Comparing Baselines**

You can compare two versions of the same Ameos system or even two different Ameos systems (although that may have limited utility). The systems to be compared can be in the form of an actual system or a system baseline. The result of baseline comparison is a report, `compare_system`, that lists the differences between the two systems. Examples of changes reported are new symbols and diagrams, global renames of symbols and diagrams, annotation/description changes, etc.

**Note:** The systems to be compared must have the same root. That is, they must have been created with the same version of Ameos (e.g., StP 8.3 or Ameos `<version>`). You cannot migrate and then compare.

As an example of usage, compare the current system to another system as follows.

1. Open a system in Ameos, then choose **Edit > Compare System.**

2. In the dialog that appears, specify the location of the second system in the **Other System** field.

3. Specify the name and path of the output file and format as desired. The default file for Windows platforms is `<projdir>/qr_files/compare_system.rtf`.

4. Click **View After Run**, then click **OK**.

You may view the file directly or by choosing **File > Open Report** from the Ameos desktop.

Note that you can compare two subsystems in a similar fashion. To do so, use **System > Baselines > Compare Baseline.**

You can do other comparisons by changing the `proj`, `system`, or `baseline` external values appropriately with the Ameos script manager.
Using Version Control

To use the Ameos version control facility, you access the Version Control dialog in either of the following ways:

- Automatically, from Ameos’s baselining utility (if the use_version_control ToolInfo variable is set to True)
- From the desktop, by choosing System > Version Control.

Figure 2 shows the Version Control dialog.

Figure 2: Version Control Dialog

For information about the commands available in the Version Control dialog, refer to the online help (click “About version control” on the Index page).

Using External CM Systems

You may want to keep track of your Ameos baselines with a CM system. Ameos currently provides two types of interfaces to CM systems. The first interface is for CVS, the dominant open-source network-transparent version control system. The other type of interface (e.g., for Continuus, ClearCase, and GNU RCS) is based on Tcl scripts. These scripts can be adapted to any CM system that provides a command line interface. Note that the Tcl interface may be replaced in future releases of Ameos by an interface similar to that of CVS.

CVS

Setting Up the CVS Integration

To set up your CVS integration, follow the steps below.

Note: The steps below are set up for using WinCVS. You can set up for using the Linux CVS tool, linCVS, instead if you wish. Download the tool from http://www.lincvs.org and change the directory names in the steps below appropriately for Linux.

1. Download CVS (http://www.cvshome.org/) and WinCVS (http://www.wincvs.org/).
2. Install CVS into c:/cvs (or some other path) and add that path to your PATH environment variable.
3. Create a folder, c:/cvsroot, and set the CVSROOT environment variable:
CVSROOT=:local:c:/cvsroot
This is your CVS repository (not the workspace).

4. From a command shell, run:
   
cvs init

5. Create a directory \textit{C:/CVSWorkspace/AmeosBaselines} (this will be your CVS workspace).

6. Install WinCVS, putting the installation into, e.g., \textit{WinCVS1.3}.

7. Launch WinCVS.

8. When prompted, set CVSROOT like this. (Note: The dialog may be different for different releases of WinCVS.)


10. Using the "Change location" button, change your browse location in WinCVS to \textit{C:/CVSWorkspace/AmeosBaselines} to view the Ameos baseline files you create.
11. Assuming you’ve installed in the locations noted above, set the following in your preferences:

   baseline_path=C:/CVSWorkspace/AmeosBaselines
   CVSROOT= $(env.CVSROOT)
   CVSGui=C:/WinCVS1.3/wincvs.exe

**Example: Baselining and Version Control with the CVS Integration**

This section presents a simple example of how to use the Ameos baselining and version control facilities with CVS. Note that, in addition to the what is shown in the example, CVS capabilities are also available for subsystems (System > Model Management commands) and for comparing systems (Edit > Compare System).

1: Create a Baseline and Import It into CVS

1. First, make sure you have installed CVS and set appropriate variables as described previously.
2. Start Ameos and open system `uml_email`.
3. Create a system baseline for `uml_email`.

   To do this, select System > Baselines > Baseline Current System. A dialog similar to the following appears.

   ![Baseline Current System dialog](image)

   **Note:** The Overwrite existing baseline file checkbox should always be activated.

Since `C:/CVSWorkspace/AmeosBaselines` is not yet under version control and there is no module in CVS yet, Ameos will ask if it should import the `AmeosBaselines` directory as a new module:

![Import module dialog](image)
4. Click **Yes**. Ameos will now run the following two CVS commands:

```bash
cvs import -kb -m"First commit" AmeosBaselines ameos init
cvs co AmeosBaselines
```

In other words, it will:
- Create a new module, **AmeosBaselines**, in the CVS repository.
- Import **uml_email.zip** (and any other files that might be present in **AmeosBaselines**).
- Check out **uml_email.zip** to set up the workspace.

The result is that:
- A baseline of **uml_email** is now stored in CVS.
- Your workspace has an unmodified copy of this baseline.

**Note:** Importing (and creating a new module) happens only if the baseline directory is not yet under CVS control (that is, no CVS directory is present). In this case, Ameos always creates a module with the same name as that of the baseline directory (e.g., **AmeosBaselines**). If, however, the baseline directory is already under version control but the baseline to be created is not, Ameos adds the zip file to the baseline module like this:

```bash
cvs add -kb Millennium_Clock_Java.zip
cvs commit -m" Millennium_Clock_Java" Millennium_Clock_Java.zip
```

### 2: Commit a New Baseline to the Repository

Ameos automatically creates a new version of a baseline in CVS if (1) a baseline that has been created already exists in the baseline directory and (2) the **Commit to CVS** button of the **Baseline Current System** dialog is activated. For the **uml_email** baseline we are illustrating in this example, the Ameos message log would read as follows:

```bash
cvs commit -m"Ameos commit for system uml_email" uml_email.zip
Checking in uml_email.zip;
e:/cvsroot/AmeosBaselines/uml_email.zip,v  <--  uml_email.zip
   new revision: 1.2; previous revision: 1.1
```

The result is that a new version of **uml_email** is created in the repository

### 3: Restore a Previous Version from CVS

Provided you already have a versioned copy of your baseline in the baseline directory, you can restore it. To do so:

1. Select **System > Baseline > Restore Selected Baseline System.**
2. Browse to select the **Baseline file**.
3. Activate the **Update from CVS** button.
4. Optionally provide the **CVS version** number you wish to retrieve.

In this case, Ameos runs the CVS command

```
cvs update -P -r 1.1 AmeosBaselines/uml_email.zip
```

before restoring the model.

### 4: Get a New Baseline from CVS

Ameos cannot assist you in browsing for available CVS modules and files, since the CVS API does not support it. Therefore, when you want to restore a baseline for a model that you have not checked out before, you must manually enter a `<module>/<baseline>.zip` string into the **Baseline file** field of the **Restore Selected Baseline System** dialog.

For example, if you want to restore `Millennium_Clock_CPP.zip`, but have not yet checked it out, manually enter the full path name in the **Baseline file** field and press **OK**. Ameos will use "AmeosBaselines/Millennium_Clock_CPP.zip" as the module and file name and will attempt to check it out from CVS:

```
No local CVS information found for Millennium_Clock_CPP.zip.
Trying to check out AmeosBaselines/Millennium_Clock_CPP.zip from CVS ...
Updating AmeosBaselines/Millennium_Clock_CPP.zip from CVS ...
cvs checkout AmeosBaselines/Millennium_Clock_CPP.zip
```

If the checkout is successful, Ameos will continue to restore the model.

---

**Note on default module names and the baseline directory**

In the current implementation, for CVS imports and CVS checkouts when no CVS control files are present, Ameos always maps the last part of the baseline directory path name to the CVS module name. To avoid this, use WinCVS or a similar tool for the first operation. After that, commits and updates will use the module names found in the CVS control files instead of the default mapping.

---

**Other Integrations**

This section addresses how to set up for the supported Tcl-based CM integrations, namely Continuus, ClearCase, and GNU RCS.
When a Tcl-based version control system is integrated with Ameos, it is detected with the
use_version_control variable defined in the preferences. This tells Ameos to display a special dia-
log after creating the baseline, which allows you to check the baseline directly into the external CM sys-
tem. The specific version control system is specified by the vcc_cmds variable.

Note: The Ameos Tcl-based baselining system does not include built-in “differences” or “merge” func-
tions, which are typically associated with CM systems. To detect differences between two systems or
subsystems, you can perform a differences operation on the ASCII flat files or open two instances of
Ameos and Alt+Tab between each current diagram and the baseline. The Restore Subsystem com-
mand (System > Model Management) effectively merges a baselined subsystem with the entire system.

Note: Recall that the Tcl interface may be replaced in future releases of Ameos by an interface similar to
that of CVS.

**Continuus**

Ameos supports an interface to Continuus 4.5. Before using the Continuus integration, you must:

1. Set the following in your preferences:
   ```
   vcc_cmds=ct/vcc/continuus/Vcc_continuus_commands
   vcc_server=<server>
   vcc_datadir=<directory>
   vcc_dbname=<database>
   ```

2. Set the CCM_PASSWORD environment variable to your Continuus password.

**ClearCase**

Ameos supports an interface to ClearCase CM software. Before using the ClearCase integration, you
must set the following in your preferences:

```
vcc_cmds=ct/vcc/clearcase/Vcc_clearcase_commands
vcc_vobtag=/<vob_name>
vcc_viewtag=<view_name>
```

Note: To determine the appropriate values for vcc_vobtag and vcc_viewtag, run the ClearCase
commands `lsvob` and `lsview`. The vcc_viewtag variable may be set to a specific directory within the
view where baselines are to be created.

Note: In order to get version control to work with ClearCase, you may have to adapt the variable
vcc_create_command located in line 7 of `<Ameos>/ct/vcc/clearcase/Vcc_clearcase_commands`. To
do so, set the file type specified in parameter "-t" to a value matching the ClearCase element type for zip
files in the VOB you are planning to use. This is probably "zip", but could be "compressed_file" or some-
thing else depending on how your ClearCase administrator has set up the VOB.

**GNU RCS**

Before using the RCS integration, you must set the vcc_cmds ToolInfo variable preference to the
Vcc_rcs_commands file:

```
vcc_cmds=ct/vcc/rcs/Vcc_rcs_commands
```
Ameos model management supports distributed development of large, complex projects through subsystem partitioning and management. Ameos model management allows independent development of separate subsystems while protecting the integrity of the entire model. Partitioning into subsystems can be done at any time during the development process. Developers are not forced to prematurely partition a system at the beginning of a project.

This chapter includes the following topics:

- An overview of model management (below)
- Using the model management tree browser (page 75)
- Defining a subsystem (page 76)
- Managing subsystems (page 81)

Using Ameos Model Management—An Overview

Ameos model management allows an Ameos/UML model to be decomposed into subsystems, which can then be developed and managed independently of one another while maintaining the overall integrity and consistency of the entire model. Users can check out a portion of a model and work “off line” or in a private workspace in a flexible manner, and check their work back in again, with seamless integration of results into a shared repository.

With Ameos model management, you can:

- Partition an Ameos/UML model into logical subsystems.
- Create private and shared views of a model.
- Track changes to and dependencies between subsystems.
- Lock and unlock subsystems, as needed.
- Manage subsystems, using version control, with a supported external configuration management (CM) system.

About Subsystems

Ameos model management uses the subsystem construct introduced in OMG UML 1.3 to provide “containers” for the model partitions. Subsystems are represented graphically by using the package symbol with a «subsystem» stereotype. Subsystems are the basis for configuration control and for private workspaces. Ameos model management provides several commands for managing subsystems. Each command is performed on a single subsystem or a tree of subsystems (if the subsystem contains other subsystems).

Subsystem Contents

A subsystem can contain any UML model elements. It includes those model elements that exist within the subsystem and the interfaces that the subsystem exports. Subsystems can contain any logical grouping of model elements, including other subsystems.
Local and External Elements

Each element contained in a subsystem is called a “local model element” (local class, local use case, and so forth). A model element is considered external to a subsystem (“external model element”) if it is referenced in a diagram belonging to that subsystem, but the model element itself is assigned to another subsystem (or to no subsystem). External model elements are normally used to describe the interface between subsystems.

The Model Management Process

To use model management, you generally perform the following steps:

1. Partition a model into logical subsystems (see “Defining a Subsystem” on page 76).
2. Export the subsystem into a private workspace or configuration management system (see “Creating a Private Workspace or Baselining a Subsystem” on page 83).
   Exporting a subsystem collects together all of the entities contained within a subsystem and any related elements in the wider model.
3. Lock the subsystem in the main model to prevent unwanted changes (see “Locking and Unlocking a Subsystem” on page 84).
4. Reassemble the subsystems by restoring them to the main model (“Restoring a Subsystem” on page 84).
   Restoring a subsystem restores all of the model elements of a subsystem into the main model, releasing the locks and resolving all of the changes that have taken place since the subsystem was exported.
5. Update external elements in your current private workspace model with relevant changes made to the main model (see “Reconfiguring External Model Elements” on page 85).

Using the Model Management Tree Browser

All subsystems defined in an Ameos model appear in the ModelManagement View category in the Ameos desktop’s model pane. When a subsystem is selected in the ModelManagement View, all of its assigned diagrams appear in the objects (right) pane. Click the plus sign (+) in front of a subsystem to see all the other assigned model elements, such as use cases and classes. An example of the ModelManagement View appears in Figure 1.
Figure 1: ModelManagement Tree Browser

The Unassigned Diagrams folder in the ModelManagement View shows all diagrams that are not yet assigned to any of the defined subsystems.

All actions that can be performed on subsystems are available on the desktop, from commands available by choosing one of the **System > Model Management** commands (or the shortcut (right-click) menu. To use the shortcut menu, select a subsystem in the model pane and click the right mouse button to display a shortcut menu of available commands.

**Note:** To ensure reliable results and identify potential problems before performing model management operations on subsystems, first ensure that your global model is semantically correct by performing semantic checking on the entire model.

**Defining a Subsystem**

Subsystems can be used in various ways in different Ameos/UML diagrams. For example, you can define the architecture of the overall system and show relationships between subsystems in a class diagram. Other diagrams, such as use case diagrams, may define the subsystem containment of model elements.

To define a subsystem in an Ameos/UML model:

1. On a class or use case diagram, insert and label a package symbol with the subsystem name.
2. Call the properties window for the package with **Edit > Properties** or the corresponding shortcut menu.
3. In the properties window, change to the **Extensibility** tab.
4. From the pick list of predefined stereotypes that appears, select **subsystem**, as shown in Figure 2, and click **OK** on the pick list dialog.
5. Click **Apply** on the package symbol’s properties window.

The «subsystem» stereotype appears within the package symbol, in the diagram, identifying it as a subsystem.

Figure 3 shows a simplified class diagram for an example system in which subsystems have been drawn.

**Figure 3: Subsystems Drawn in a Class Diagram**

The title bar displays the name of the diagram, followed by the system name and the name of the current editor. If the diagram has been modified but not saved, the diagram/table and system names appear within parentheses:

(<diagram_name>: system_name)

**Note:** The word <unassigned> may appear in the title bar if the current diagram has not been assigned to a subsystem, but at least one other diagram has been assigned.

**Representing the Subsystem**

**Assigning Model Elements to a Subsystem**

Subsystems are containers for other model elements. All model elements that are part of the UML notation are assigned to subsystems when they are drawn within the subsystem package symbol. In
Figure 4, the classes drawn inside the clockwork subsystem’s package symbol are automatically assigned to subsystem clockwork.

**Figure 4: Clockwork Subsystem with Classes**

Propagating Subsystem Membership

Logical membership in a subsystem is propagated wherever reasonable. For example, if package MailMessenger is part of subsystem MailLibrary, all classes in package MailMessenger automatically belong to subsystem MailLibrary. All directly associated elements, such as a superstate for a class or a scenario for a use case, also belong to the subsystem to which the parent (“scope element”) itself belongs.

Table 1 describes propagation of subsystem membership, where the scoped element (child) belongs to the subsystem of the scope element (parent).

<table>
<thead>
<tr>
<th>Scope Element</th>
<th>Scoped Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>Contained elements (packages, classes, use cases)</td>
</tr>
<tr>
<td>Class</td>
<td>Operations, attributes, and inner classes</td>
</tr>
</tbody>
</table>

Assigning Diagrams to Subsystems

Diagrams can be assigned to subsystems as well. This does not redefine any existing association between elements and subsystems; rather, it defines the graphical representation of those elements the user wants to see in a subsystem.

You can assign diagrams to subsystems from an Ameos editor or from the Ameos desktop. You cannot assign or reassign diagrams to or from a subsystem that is locked.

**Note:** Class, state, and activity diagrams are always implicitly assigned to a subsystem and cannot be manually assigned or reassigned. For details on implicit assigned diagrams see Implicit Assigned Diagrams to Subsystems.

**Assigning diagrams to subsystems in an Ameos editor**

To assign the current diagram to a subsystem from within an Ameos editor:

- From the editor, choose **Edit > Assign Diagram to Subsystem**.
- In the **Object Selector** dialog that appears, select the subsystem and click **OK**.
Assigning diagrams to subsystems from the Ameos desktop

To assign a diagram to a subsystem from the Ameos desktop:

- In the ModelManagement View, open Unassigned Diagrams and select one of the diagram subcategories.
- In the Unassigned Files list (right pane), select a file.
- From the context menu, choose Assign File to Subsystem.
- In the dialog that appears (Figure 5), select a subsystem name and click OK.

![Assign File to Subsystem Dialog](image)

Assigning/removing multiple files to/from a subsystem

To remove a group of diagrams from a subsystem or to assign them to a subsystem from the Ameos desktop:

1. In the ModelManagement View, select the appropriate subsystem.
2. From the context menu, choose Assign/Remove Files to Subsystem.
3. The dialog that appears (Figure 6), you see the following elements.
   - At the top: The selected subsystem
   - On the left (upper list): Diagrams where the subsystem is referenced (drawn)
   - On the left (lower list): Diagrams in which this subsystem is not visible
   - On the right: Diagrams that are already assigned to the subsystem
Figure 6: Assign/Remove Files To Subsystem Dialog

All diagrams are shown by name, followed by the name of the subsystem to which they are assigned.

- To add a diagram to the subsystem, select it from one of the lists on the left, then press the corresponding > button.
- To remove a diagram from the subsystem, select it in the right list and press the < button.
- When you are finished press OK.

You can always cancel your reassignment by leaving that dialog by pressing Cancel.

To indicate that a diagram has been assigned to a subsystem, the title bar in the diagram displays the subsystem to which it belongs, within angle brackets, between the diagram name and the system name, as in Figure 7.

Figure 7: Subsystem Name in Diagram's Title Bar

After at least one diagram has been assigned to any subsystem in your model, any unassigned diagrams thereafter display the word <unassigned> in the title bar, and also appear in the Unassigned Diagrams category in the ModelManagement View on the desktop. If no diagrams have been assigned to a subsystem, then the word <unassigned> does not appear in any diagram’s title bar.
Implicitly Assigned Diagrams

Wherever possible, diagrams that include elements of a decomposition or navigation are implicitly assigned to the subsystem to which the element’s parent belongs.

For example, if a class belongs to a subsystem and the state machine of the class is described in a state diagram, this diagram is implicitly assigned to the subsystem.

Table 2 shows how diagrams that include elements of a decomposition/navigation (Scoped Element) are automatically assigned to the subsystem of the parent (Scope Element).

Table 2: Implicit Assignment of Diagrams to Subsystems

<table>
<thead>
<tr>
<th>Scope Element</th>
<th>Scoped Element</th>
<th>Diagram/Table Implicitly Assigned to Scope Element’s Subsystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case</td>
<td>Scenario instance</td>
<td>Sequence diagram (*)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Collaboration diagram (*)</td>
</tr>
<tr>
<td></td>
<td>State machine</td>
<td>State diagram</td>
</tr>
<tr>
<td>Class</td>
<td>State machine</td>
<td>State diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Activity diagram (**)</td>
</tr>
<tr>
<td>State</td>
<td>State definition</td>
<td>State table</td>
</tr>
</tbody>
</table>

Note: When it is initially created, a scenario defined in a sequence or collaboration diagram is assigned to the same subsystem as its parent use case. However, this initial assignment can be changed by the user. Because sequence and collaboration diagrams can contain objects from classes belonging to different subsystems, it is intentionally left to the user to manually assign these diagrams to the desired subsystem.

Note: An activity or a state diagram that refines a state symbol of another activity or state diagram is implicitly assigned to the same subsystem as the parent activity state.

Managing Subsystems

Subsystems form the basis for private workspaces and provide the interface to version control systems. When one or more subsystems have been defined, you can perform the tasks described in the following sections:

- Checking semantics for subsystems (immediately below)
- Creating a private workspace or baselining a subsystem (page 83)
- Locking and unlocking a subsystem (page 84)
- Restoring a subsystem from either a private workspace or a CM tool into a “global” model (page 84)
- Reconfiguring model elements — updating external model elements in a private workspace with the current version of elements from the main model (page 85)

Commands for performing these operations are available in the ModelManagement View of the desktop tree browser and can be accessed through the appropriate model management command on the Model menu (in the Check, Locks, and Model Management submenus). To access a model management command, select a subsystem in the ModelMangement View, then navigate to the command.

The following sections describe model management features. Each section (e.g., “Checking Semantics for Subsystems”) corresponds to a model management command (e.g., Check Semantics for Subsystems).
Checking Semantics for Subsystems

The purpose of checking semantics in subsystems is to guarantee completeness and consistency within a subsystem, even if the subsystem is restored in a private workspace. A link in a diagram belonging to a particular subsystem is semantically correct if it affects local model elements only (those belonging to the same subsystem). Semantic errors are reported for any links that potentially affect the definition or behavior of model elements from external subsystems. For a definition of local and external model elements, see "About Subsystems" on page 74.

Note: Model management’s subsystem semantic checking does not take the place of global completeness and consistency checks for the entire model. To perform global semantic checking on your model, use System > Check > Check Semantics for Whole Model from the desktop.

You can check the semantics of subsystems from the desktop or within the Ameos editors:

- To check semantics for a selected subsystem only, select the subsystem in the ModelMangement View on the desktop, and choose System > Model Management > Check Semantics for Subsystem or select it directly from the shortcut (right-click) menu.
- To check semantics in an Ameos editor for the subsystem to which the current diagram belongs, choose Check Diagram’s Subsystem Semantics.

Checking semantics within an editor reports only errors caused by the current diagram, and allows you to navigate to the symbol that is the source of the error in that diagram.

In addition, when you work with subsystems in private workspaces and then reassemble the subsystems into your global model, it is possible for some elements, such as state machines, to become “orphaned” if the parent class has been deleted from the model. If an element is orphaned in this way, it is not recognized by model management as belonging to the system, and will not appear in the ModelManagement View. To detect these potential problems, use System > Model Management > Check Semantics for Subsystems from the desktop each time you reassemble subsystems into the global system. Ameos issues warnings about apparently orphaned elements, as described in “Orphaned element check,” below.

Orphaned element check. Ameos issues a warning message if it detects either of the following:

- A state or activity diagram contains no state machine.
- The state machine in these diagrams has no corresponding class or use case in a diagram.

General checks. At least one file containing the subsystem package has to be assigned to the subsystem. A model element cannot be assigned to more than one package.

Specific checks. The restrictions noted below are intended to confine changes to model elements inside the subsystem (and thus to prevent changes to external elements).

(1) Use case diagram checks. The following use-case restrictions apply to diagrams assigned to a subsystem:

- A use case cannot extend a use case belonging to a different subsystem.
- A use case cannot be included by a use case belonging to a different subsystem.
- A use case cannot be the generalization of a use case belonging to a different subsystem.

These restrictions are necessary to prevent changing elements outside the subsystem. For example, suppose that MyUseCase is assigned to subsystem MySubsystem, and you then decide to make MyUseCase a generalization of an external use case OtherUseCase. If you do this, you change the inner structure of OtherUseCase. This is because the external element OtherUseCase is now defined as a specialization of MyUseCase. If you require MyUseCase to be a generalization of OtherUseCase, draw the generalization link in a diagram that is either assigned to a to a subsystem that contains OtherUseCase or to no subsystem at all.
(2) **Class diagram checks.** The following constructs are not allowed in diagrams assigned to a subsystem:

- A generalization from external to local class
- A dependency from external to local class
- A dependency with friend stereotype from and to a local class
- An undirected or bidirectional association between an external and local class
- A directed association from an external to a local class

In addition:

- A local class cannot be the aggregation/composition of an external class.
- An association class cannot belong to a different subsystem than the class where the navigation starts.
- For an n-ary association, all partner classes must be in the same subsystem. According to UML, there is no way to define a navigability for n-ary associations, therefore Ameos considers the navigation to be in all directions).
- An inner class cannot be assigned to a different subsystem than the outer class.

As with use cases, class diagram checks are necessary to prevent changing elements outside the subsystem.

(3) **Sequence diagram checks.** The following constructs are not allowed in diagrams assigned to a subsystem:

- Any message sent from an external to a local class except for return messages
- Any message to an external class with no corresponding operation

(4) **Collaboration diagram checks.** These checks are the same as for a sequence diagram.

(5) **Activity diagram checks.** No checks are needed.

(6) **State diagram checks.** No checks are needed.

(7) **Component diagram checks.** Any symbol referenced in a diagram assigned to a subsystem cannot be referenced in another diagram assigned to a different subsystem.

(8) **Deployment diagram checks.** These checks are the same as for a component diagram.

### Creating a Private Workspace or Baselining a Subsystem

Use Ameos's subsystem baselining capabilities to:

- Create a private workspace for a user.
- Export a subsystem (check in a baseline) to a configuration management (CM) version control system.

To baseline a subsystem:

1. Select the subsystem on the desktop and choose **System > Model Management > Baseline Subsystem.**
2. Complete or modify the entries in the dialog that appears.
The **Baseline Subsystem** dialog allows you to select the subsystem to be baselined from a list of all not currently locked subsystems in the model.

In creating a baseline for a subsystem, Ameos creates a file named `<subsystem-name>.zip` (in this example, `MailMessenger.zip`) in the specified directory. If the baselined subsystem has nested subsystems, Ameos generates a separate zip file for each subsystem.

Creating a baseline for a subsystem performs checks that are specific to model management. This ensures the integrity of the subsystem so that it can be safely re-imported at a later time. *Errors reported by those checks need to be corrected before a subsystem can be baselined.* (Note: Error checking can be performed separately via **Model > Model Management > Check Semantics for Subsystem**.)

### Locking and Unlocking a Subsystem

A defined subsystem can be manually locked and unlocked, using **Lock Subsystem** and **Unlock Subsystem** on the **Model > Model Management** menu or shortcut (right-click) menu. You can also lock a subsystem by selecting the **Lock subsystem after baselining** option when creating the baseline.

A locked subsystem cannot be changed and no baseline can be created from it. All the locks can be removed with **Unlock Subsystem**.

**Note:** Ameos generates an error message if you attempt to lock a subsystem that is already locked, or attempt to unlock a subsystem that is already unlocked.

### Restoring a Subsystem

Baselines of subsystems that were created as described in “Creating a Private Workspace or Baselining a Subsystem” on page 83 can be restored to create a private workspace or to load a different version. All information from a selected zip file is restored into the current Ameos system.
To restore a baseline for a subsystem:

1. Select the subsystem you want to restore in the ModelManagement View on the desktop.
2. From the Model menu, choose Model Management > Restore Subsystem.
3. In the Restore Subsystem dialog, enter or browse and choose the baseline file from which to restore the subsystem.

The behavior and the actions that are performed during the restore depend on the existing information in the system, as described in the following sections.

**Restoring to an Empty System**

This is the easiest case; no checks are necessary. All information in the zip file is restored and all restored external model elements are locked to prevent them from changing.

**Restoring to a System with Other Subsystems**

In this case, the Ameos system contains legal model elements. Therefore, Ameos has to check for name conflicts before restoring the subsystem. A name conflict occurs if existing model elements in the Ameos system have the same name as model elements in the baseline, but belong to different subsystems. Name conflicts need to be resolved by the user before the restore operation can succeed.

External model elements that already exist in the current system are not restored, to avoid overwriting the current version.

All external model elements are locked to prevent them from changing.

**Restoring to a System Where this Subsystem Exists**

This occurs if a subsystem from a previous version or a subsystem from a private workspace is restored.

Ameos first checks for name conflicts, which must be resolved as described in “Restoring to a System with Other Subsystems” (above). Next, Ameos checks to see whether the same model elements in the baseline also exist in the current system, but have different names. If so, the model elements are renamed in the current system to ensure that a consistent model exists after the restore.

All external model elements are locked to prevent them from changing.

**Reconfiguring External Model Elements**

The Reconfigure External Model Elements command allows you to update all external model elements in your current private workspace model with the current version of those elements from the main model. For example, suppose you are working on MailMessenger in your private workspace and you learn that MailServer (and many others that you do not “own”) have changed. You need to re-import the external model elements into your private model.

This command looks for the external subsystem zip files in a path defined by the baseline_path variable of your preferences. The default path is `<projdir>/baseline`. You can select a different path in the command’s dialog. It then updates all external model elements, such as class tables and annotations of external elements, provided a zip file containing the parent subsystem of that element is present in the selected directory. This means that in order to update all external elements of a system, all subsystem baselines (zip files) need to be put into the selected baseline directory (for example, by checking all those files from a CM tool).

To invoke this command, select a subsystem in the ModelManagement View on the desktop and do either of the following:

- From the Model menu, choose Model Management > Reconfigure External Model Elements.
Known Issues and Limitations

To see recent changes or additions to subsystems, it may be necessary to refresh the current model management window or dialog, using the dialog’s Reset button or the Refresh tree command on the desktop ModelManagement View shortcut (right-click) menu. For example, if a new subsystem is added, it is not immediately visible in the Baseline Subsystem dialog until you click the dialog’s Reset button. Likewise, if you move a file into a subsystem as described in “Managing Subsystems” on page 81, it will not appear in the ModelManagement View on the desktop until you refresh the desktop, using the Refresh Tree command.

Baselining a subsystem that has nested subsystems generates a separate zip file for each subsystem. When you restore this subsystem, you need to restore each of the zip files separately.
This chapter discusses the third-party integrations available with Ameos – specifically DOORS (below) and the SNIFF+ and Eclipse programming environments (page 90).

**DOORS**

DOORS, a registered trademark of Telelogic, is designed to capture, manage, link, trace, generate, and analyze textual and graphical information to ensure a product’s compliance with system requirements and specifications.

The Ameos/DOORS integration allows you to:

- Export single or multiple objects from Ameos to DOORS.
- Navigate from Ameos-exported objects to DOORS.
- Allocate requirements to those exported objects using DOORS.
- Navigate from exported objects in DOORS back to Ameos.

The integration of Ameos and DOORS ensures that both requirements and application development models systematically reflect changes made throughout the development cycle.

The Ameos/DOORS integration is available from all Ameos editors and products. All objects that support allocated requirements can be exported to DOORS.

For detailed information on DOORS, refer to the DOORS product documentation.

**How DOORS Stores Ameos Objects**

The Ameos/DOORS integration allows Ameos objects to be sent to DOORS and allows DOORS to manage the details of linking these objects to requirements. The only restrictions placed upon the storage of Ameos objects in DOORS are related to how the objects are grouped into modules.

DOORS stores Ameos objects in formal modules. All Ameos objects from the same Ameos system are grouped together into their own module. Therefore, you will have one module for every Ameos system you are using with DOORS. DOORS manages all links from the formal module to requirements.

When you export an Ameos object to DOORS, several things happen:

1. The integration checks to make sure you have a project open. If no project is open, a dialog appears telling you to open a project and to press OK to continue.
2. The integration searches the project for formal modules belonging to the Ameos system that owns the Ameos object.
   - If such a module is found, it is opened and the object is added to it or is updated if it already exists.
   - If no module is found, you will be prompted for the name of the new module. The default name provided is the name of the Ameos system. If you choose to abort, the module is not created.
3. DOORS stores the Ameos object as a surrogate object, which represents the actual Ameos object in DOORS.

The process is illustrated in Figure 1.
After you have exported an Ameos object to DOORS, you can perform any valid DOORS operation involving the exported object, such as linking the object with requirements, editing the object’s attributes, and so on. You can write reports in DOORS that show the status of these Ameos objects and their linked requirements. It is also possible to write a QRL report in Ameos that will extract the DOORS requirements linked to Ameos objects, and print them in an Ameos report, such as a Software Requirements Specification (SRS).

### Enabling the Ameos/DOORS Integration

Both Ameos and DOORS must be installed before you can enable the Ameos/DOORS integration. Follow the steps below to enable the integration.

1. Work within a single Ameos project directory and use that name as your `projdir` preference variable.
2. Set up your preferences file for DOORS integration.
   - Set `use_doors_integration`. Set it to “True” for each Ameos/DOORS user. “True” enables DOORS; “False” disables DOORS.
3. Move the following from the `Ameos\templates\ct\doors\addins` directory to the DOORS installation directory:
   - `Ameos\`: The `Ameos` directory contains the Ameos customization. Copy the entire directory to the `%DOORSHOME%lib\dxl\addins` directory.
   - `startup.dxl`: Add the contents of this file to:
     - `%DOORSHOME%lib\dxl\startup.dxl`
     
     If the file does not exist, create it.
   - `addins.hlp`: Add the contents of this file to:
     - `%DOORSHOME%lib\dxl\addins\addins.hlp`
     
     If the file does not exist, create it.
   - `addins.idx`: Add the contents of this file to:
     - `%DOORSHOME%lib\dxl\addins\addins.idx`
     
     If the file does not exist, create it.
4. Add, to your `$PATH` environment variable, the name of the directory where the Ameos executables are stored. For example, if you have installed Ameos on your `c:` drive, use:
   ```
c:\Ameos\bin\W32NTX86
   ```
5. (For Windows implementations only) Create a new environment variable `IDE_LICENSE` and set it to either `ameos_developer`, `ameos_modeler`, or `ameos_analyst` depending on your installed license. This is required so that DOORS can properly start Ameos. (If you start Ameos before running DOORS, this is not necessary.)

**Note:** If you change the location of your project after the surrogate module has been generated, you need to specify the new location by setting `Ameos_ProjectDir` within DOORS. To set `Ameos_ProjectDir:`
Open DOORS, then open the surrogate module.

Choose File > Module Properties, then select Ameos_ProjectDir.

Press the edit button, modify the attribute, then press OK.

Using the Ameos/DOORS Integration

When you have enabled the Ameos/DOORS integration, you can perform the following additional actions within the Ameos and DOORS programs:

1. Export objects from Ameos to DOORS.
2. Navigate from Ameos to DOORS.
3. Navigate from DOORS to Ameos.
4. Update renamed objects exported to DOORS.

These actions are described in the sections that follow.

Exporting Objects from Ameos to DOORS

Exporting objects is the means for creating a link between an object in the Ameos model and its requirements in the corresponding DOORS module. When Ameos objects are exported to DOORS, identifying information about the Ameos objects is sent to DOORS, which then adds the object to the Ameos surrogate module. You can then navigate between the object in the Ameos model and its representative object in the DOORS module.

Objects can be exported from Ameos to DOORS in two ways:

1. From the desktop. Select one or more files, right click to bring up the context menu, and select Export to DOORS. This action exports all exportable objects in the selected diagrams into DOORS.
2. In the editors as a result of the navigation commands. As a convenience, navigation from Ameos to DOORS automatically exports the selected object to DOORS if it does not already exist in the DOORS surrogate module. To export objects in this manner, select a single object (class, use case, process, and so on) and choose the DOORS menu item from the GoTo menu. This action exports the currently selected exportable object to DOORS.

Note that not all types of Ameos objects support allocated requirements. If Export to DOORS and/or the GoTo > DOORS menu items do not appear, it means that the item selected does not support allocated requirements, and thus cannot be exported to DOORS. (Multiple items cannot be selected for export to DOORS.)

Navigating from Ameos to DOORS

You navigate from a selected object in an Ameos editor to DOORS by choosing DOORS from the editor’s GoTo menu.

If the DOORS command does not appear, and Ameos/DOORS is enabled, the selected object does not support allocated requirements and cannot be exported to DOORS.

To navigate from an Ameos object to DOORS:

1. Make sure DOORS is running.
2. In DOORS, open the project to which you want to export the Ameos object.
3. In Ameos, select an object (class, use case, process, and so on).
   You cannot export a group of objects to DOORS. Export only one object at a time; wait until the object has completely exported before exporting the next object.
4. From the GoTo menu, choose DOORS.
If this is the first time you are exporting an Ameos object from this system, DOORS will prompt you to enter a name for the module.

5. Enter a name (DOORS will offer the name of the current Ameos system as the default) and click Create.

After you have exported an Ameos object to DOORS, you can perform any valid DOORS operation on the surrogate object, such as linking the object with requirements, editing the object's attributes, and so on.

**Navigating from DOORS to Ameos**

After you export an Ameos object to DOORS, DOORS lets you allocate requirements to the Ameos surrogate object. DOORS groups all Ameos objects from the same Ameos system into their own formal module. In other words, there is one formal module for every Ameos system being used with DOORS.

To navigate from DOORS to Ameos:

1. In DOORS, select an object (class, use case, process, and so on).
2. Choose Navigate to Ameos from the Ameos menu.
   - If Ameos has more than one object with the name of the selected object, an object selector dialog appears, listing the object locations.
3. Select the object location you need and click OK or Apply.

After you navigate from DOORS to Ameos, the Find window of the Ameos desktop appears, which enables the next navigation. An Ameos/DOORS navigation requires an Ameos editor because the navigation uses QRL statements. The Find window appears because it accepts QRL statements and is not Ameos product specific.

**Updating Renamed Objects Exported to DOORS**

Whenever you rename an object by using the editor's Edit > Rename <object> Systemwide command, you need to update the object's references in DOORS. To do so, select System from the desktop and choose Synchronize DOORS with Ameos.

For more information on the Rename Systemwide command, see “Renaming” on page 93.

**DOORS Limitations**

Depending on the version of DOORS installed, the integration may only work properly if DOORS has been first invoked from Ameos. If you find that the navigations from one tool to the other are not working, restart DOORS from Ameos. This is accomplished by running any DOORS navigation from Ameos without the DOORS tool previously started.

**Integrating SNiFF+ or Eclipse with Ameos/UML**

Ameos offers navigational support between Ameos/UML and these programming environments:

- Bidirectional navigation between Ameos and SNiFF+ (Wind River Systems, Inc.)
- Bidirectional navigation between Ameos and Eclipse (2.1 and 3.0 M6)
Setting Up Integrations

In order to navigate successfully between Ameos/UML and the supported programming environments, you must:

- Set the appropriate environment and Ameos preferences, as described in the README files provided on the CD.
- Use source code in which the object names are identical to the object names in the Ameos/UML model (usually accomplished by either generating or reverse engineering the code in Ameos).

For directions on setting up the integrations, refer to the following README files on your Ameos installation CD: [ECR# 5278]

- templates\ct\sniff\README-sniff.txt
- templates\ct\eclipse\README.txt

Note: Ameos does not support concurrent integrations of both SNiFF+ and Ameos.

Navigating to SNiFF+

To navigate from Ameos/UML to the SNiFF+ programming environment, after setting up the integration, as described in the README file:

1. Start a SNiFF+ session and load the appropriate project into the programming language tool.
2. In Ameos, select an object that supports this navigation in a class diagram or table (see Table 3 on page 91).
3. From the GoTo menu or shortcut (right-click) GoTo submenu, choose one of the provided context-sensitive commands for navigation to SNiFF+, as follows:
   - For classes:
     - SNiFF Source Editor
     - SNiFF Class Browser
     - SNiFF Class Hierarchy Browser
     - SNiFF Class Hierarchy Browser (only relatives)
     - SNiFF Retriever
   - For operations:
     - SNiFF Operation Definition
     - SNiFF Operation Implementation
     - SNiFF Retriever

<table>
<thead>
<tr>
<th>Programming Environment</th>
<th>Ameos Supported Languages</th>
<th>Navigation From/To</th>
<th>Ameos/UML Editor</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNiFF+ (Wind River Systems, Inc.) ECR 4179</td>
<td>C++, Java</td>
<td>Ameos to SNiFF+</td>
<td>Class</td>
<td>Class, Attribute, Operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SNiFF+ to Ameos</td>
<td>Class, State</td>
<td>Class</td>
</tr>
<tr>
<td>Eclipse</td>
<td>Java</td>
<td>Ameos to Eclipse</td>
<td>Class</td>
<td>Class</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse to Ameos</td>
<td>Class</td>
<td>Class</td>
</tr>
</tbody>
</table>

Table 3: Programming Environment Integrations
Navigating to Eclipse

To navigate from Ameos/UML to the Eclipse programming environment, after setting up the integration, as described in the README file:

1. In Ameos, select an appropriate object in a diagram or table that supports this navigation (see Table 3 on page 91).
2. From the GoTo menu or shortcut (right-click) GoTo submenu, choose Programming Environment.

Navigating to Ameos/UML

To ensure correct navigation, you must first:

- Perform the setup steps in the README file, as described in “Setting Up Integrations” on page 91.
- Set the system variable in the preferences to the name of the system to which you want to navigate,
- Set the projdir variable in the preferences to the project directory.

Navigating from SNiFF+

To navigate from SNiFF+ to Ameos/UML, choose one of the following commands from SNiFF+’s Ameos/UML menu:

- Class Diagram for Class <class name>
- State Diagram for Class <class name>

Navigating from Eclipse

To navigate from Eclipse to Ameos/UML, choose Ameos > Navigate to Ameos, which is available in the context menu of classes and interfaces.
This chapter discusses renaming (below) and lock utilities (page 97).

Renaming

Ameos provides several options for renaming a diagram symbol or table cell:

- **Global Rename**: Changes the name of a specific repository object. This ensures that the name is changed in diagrams in which it occurs and is the default behavior. See “Renaming an Object Systemwide” on page 95 for details.

- **Local Rename**: Changes the name of a symbol, but only in the current diagram. See “Local Rename” on page 97 for details.

- **Multiple Global Rename**: Changes the name of every object in the repository whose name corresponds to a particular string. See “Multiple Global Rename” on page 97 for details.

Objects and Object References

This section briefly reviews the way Ameos stores information. This concept is necessary in order to understand the difference between relabeling a symbol and renaming it using Rename `<apptype>` Systemwide or Multiple Global Rename. These commands affect objects patterned after the persistent data model (PDM). For detailed information about the PDM, see Chapter 2 of Object Management System.

Each diagram symbol and table cell in a UML system is a symbolic reference to an object in the repository. A single repository object can be represented in any number of diagrams or table cells. Each symbolic reference is a particular view of the object and contributes defining information to the object in the form of annotations and relationships to other objects. Any defining information that a particular diagram symbol or table cell contributes to its corresponding repository object automatically applies to all other symbols and table cells that represent the object. Figure 1 shows a repository object (Employee) that has references in two diagrams.

---

1. `<apptype>` is the application type, e.g., UmlClass for a class, UmlAssociation for an association link.
The repository object and its symbolic references are associated by their common name. When you rename a reference in a diagram or table, you can choose either to break or to preserve the association to the underlying repository object. Renaming the reference by retyping its label breaks the association, since the name of the reference and the name of the object no longer correspond. Renaming the reference using **Rename Systemwide** (or **Multiple Global Rename**) actually renames the repository object and all of its symbolic references, so the association is preserved.

### Objects That Can Be Renamed

Any element that can be labeled in a diagram or table can be renamed. For special circumstances, see "Renaming Instances in Ameos/UML" on page 94.

Any element that derives its name from the names of related elements cannot be renamed directly. However, Ameos automatically adjusts these elements to reflect name changes in the objects on which they depend. For more information, see "Effects on Dependent Objects" on page 96.

### Renaming Instances in Ameos/UML

Renaming an instance of an element (such as an object or a message) in Ameos/UML updates the repository in the following ways:

- For objects, all instances with the same name are updated.
- For messages, all instances with the same signature and connected between the same objects are updated. (Anonymous objects with the same name are considered different unless they are created by generating a sequence or collaboration diagram.)
- The reference to the source element (such as the class or operation) is also updated to reflect the new source element name.

The original source element (such as the class or operation) is not renamed in the repository.

For example, if you rename an object from `instance:ClassA` to `instance:ClassB`, all objects currently labeled with `:ClassA` are updated to `:ClassB`. However, no changes are applied to `ClassA` in the repository.
To update ClassA (the source element) systemwide, you must rename the class in a class symbol or class table.

**Renaming an Object Systemwide**

This is the default rename behavior; the repository objects and all its dependent objects will be renamed. The name change is reflected in every diagram and table in which the object is represented.

To rename an object systemwide:

1. Select the symbol.
2. Press F2 or choose **Edit > Rename <apptype> Systemwide**
3. Change the label to the new name.
4. Press <Enter>.

or

1. Double-click on a table cell.
2. Change the label to the new name.
3. Press <Enter>.

The object and all of its references are renamed. The names and/or signatures of dependent objects and files are adjusted to reflect the change.

The rename operation is confirmed in the message log. In addition, a broadcast message is sent to any user who is editing a diagram or table containing references to the renamed object.

The rename operation is final, even if the diagram or table is not saved. To change back to the object’s original name, you must repeat the global rename procedure.

**Previewing Your Rename Selection**

You may want to preview your rename selection to check:

- How many objects (diagrams, tables) will be affected
- Whether your rename operation will succeed

You can accomplish both goals by choosing **Check Impact of Systemwide Rename**. Type in the new name in the dialog and press **Check Impact**. This creates a report displayed in the Ameos message log that shows what will happen if you decide to rename. The report shows which objects will be renamed and which diagrams (references) will have changes.

For a discussion of what to expect to be renamed for each object, see “**Effects on Dependent Objects**” on page 96.

**Global Renaming Errors**

There are two instances in which systemwide renaming does not work:

- When the new name is identical to the name of an existing object that has annotations or references
- When two users simultaneously rename either the same object or different objects with the same dependent object

If the new name already exists in the repository, the message log displays an error message and the rename operation fails. If you cannot find the object in any diagram or table, it is probably an “unreferenced object.”
Note: You should delete unreferenced objects regularly. To do this, use System > Repository > Empty Repository Trash from the Ameos desktop. For more information, see Emptying Repository Trash in the "Ameos Users" chapter of the Ameos Administration Guide.

When two users attempt to rename the same object (or a dependent object), the first rename operation to get to the object is applied. Subsequent rename operations can no longer find the target object and therefore fail. In this case, an error message is displayed.

**Effects on Dependent Objects**

Some types of objects have names or signatures that are derived from the names of related objects. For example, the name of a class table or state table corresponds to the class it describes. When an object is renamed, Ameos automatically updates the names of any dependent objects. Table 4 lists these automatic updates.

All potential changes to an object’s dependencies appear in the Check Impact report, where each object is referred to by its application type (e.g., UmlClass).

<table>
<thead>
<tr>
<th>Source Object</th>
<th>Dependent Object</th>
<th>What Happens to Dependent Objects upon Rename</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Object instance attribute value</td>
<td>(Class editor) Renames the “Value of” instance attributes for objects instantiated from the class containing the attribute in the repository and in the diagrams.</td>
</tr>
<tr>
<td>Class (in class editor)</td>
<td>Class table page</td>
<td>(Class editor, table page) Renames the file in the repository and in the file system.</td>
</tr>
<tr>
<td>Class (in class editor)</td>
<td>State diagram</td>
<td>(State editor) Renames the class portion of the name in the repository and renames the file in the file system. Renames the state machine with the new class name.</td>
</tr>
<tr>
<td>Class (in class editor)</td>
<td>State table page</td>
<td></td>
</tr>
<tr>
<td>Association</td>
<td></td>
<td>(Class editor) Renames the association in the repository, but no change is visible in the diagrams.</td>
</tr>
<tr>
<td>Operation</td>
<td></td>
<td>(Class editors) Renames the class’s constructor and destructor functions in the repository and in the diagrams and tables.</td>
</tr>
<tr>
<td>Operation’s return type</td>
<td></td>
<td>(Class editor, table page) Renames the return type to match the new class name in the repository and in the table.</td>
</tr>
<tr>
<td>Class (in an object symbol)</td>
<td>Object class’s scope</td>
<td>(Class, sequence, collaboration editors) Renames UmlObject-ClassScope to reflect the reference to the new class. Does not change the original class name in the repository.</td>
</tr>
<tr>
<td>Operation</td>
<td>Overloaded operation (differs from source operation only by signature)</td>
<td>(Class editor) If Rename Overloaded Operations in Class is on, renames all matching operation names in the repository and references; the signatures are not affected. Renames all destructor operation names if the source operation happens to be a constructor.</td>
</tr>
<tr>
<td>Use Case</td>
<td>Use case parent</td>
<td>(Use case editor) Renames the parent in the repository and the diagram.</td>
</tr>
<tr>
<td>Use Case</td>
<td>Use case diagram</td>
<td>(Use case editor) Renames the use case portion of the diagram name in the repository and renames the file in the file system.</td>
</tr>
<tr>
<td>Extension point</td>
<td></td>
<td>(Sequence and collaboration editors) Renames the extension point to match the use case name.</td>
</tr>
</tbody>
</table>
Local Rename

When you locally rename the label of a symbol, you map the symbol to a different repository object corresponding to the new name. Any relationships defined in the current diagram or table are transferred to the new object. Because the scope of retyping a label is limited to the current diagram, relationships defined in other diagrams are severed along with the mapping to the original object.

If the new label is not already in use within the system, a new repository object is created, and all of the annotations associated with the old object are copied to the new object. If the new label corresponds to an existing repository object, that object does not inherit annotations unless it has none of its own.

Changing a Label

To make a local name change of a symbol or table cell:
1. Select the symbol or table cell.
2. Choose Choose <apptype> Symbol Name from the Edit or context menu.
3. Select a name from the list or type in a new one in the Selection text field.
4. Click OK.

Multiple Global Rename

The desktop command System > Multiple Global Rename allows you to globally rename all packages, classes, operations, attributes, use cases, or actors in the repository whose names match a user-specified search pattern in the form:

\[ [*] <string> [*] \]

All matching objects are renamed with a user-specified replacement pattern.

UNIX users can use regular expressions in the search and replacement patterns.

To use this command:
1. From the Ameos desktop, choose System > Multiple Global Rename.
2. In the dialog that appears, do the following:
   - In the Type field, display the options list and select one of the object types to search for.
   - In the Search Pattern field, specify a search pattern in the form:
     \[ [*] <string> [*] \]
   - In the Rename Pattern field, specify the replacement name pattern in the form:
     \[ [*] <string> [*] \]
   - To preview the potential name changes without actually changing them, select Check impact only. This displays a list of existing names and the proposed name change for each.
3. Click OK.

Lock Utilities

Developers generally need access to one another’s models, but it is important that they do not interfere with one another’s work. If two or more users try to edit the same file simultaneously, the Ameos locking utility detects the situation and prevents users from accidentally overwriting each other’s work. Ameos also provides the ability for all users to read a file at any time, assuming that they have the necessary file permissions to do so.
How Locking Works in Ameos

There are two ways to use locking commands in Ameos:

- Via **Admin > Ameos Utility**. This command provides general locking commands for a *system*. See the “Using Ameos Utility” chapter of *Ameos Administration* for details on Ameos utility.

- Via **System > Locks > <command>**. This command allows you to operate on a specific *diagram*. This is discussed below.

You can also lock and unlock *subsystems* you create from your model. Subsystems are managed independently through Ameos model management. See Chapter 8, “Model Management” on page 74 for details.

Some locks are automatically set by the editors, while other locks must be set manually.

**Automatic Locks**

Ameos sets an automatic lock when you load a file for editing. Ameos discards the automatic lock when you load another file or quit the editor. An automatic lock can be removed by a lock administrator, who is a user with the authority to override all locks.

This is how automatic locking works:

- When a user starts editing a file, Ameos locks the file.
- When a user terminates editing, Ameos removes the lock.
- When a user tries to edit a file already locked by another user, Ameos displays a reset lock confirmation box.

**Permanent Locks**

Permanent locks are user-defined locks that control file access by a single user or a group of users. Ameos checks a list of machine/user names associated with every permanent lock to verify access rights. When you set a lock, Ameos makes you the lock owner. Lock owners can specify who is able to edit the file (is on the access list) and who is able to remove the lock (is on the delete list).

**Specifying Users in Access Lists**

Ameos maintains the following lists to determine which users have permission to override all locks, access a file, delete a lock on a file, and own a file:

- Lock administrators list (for this system)
- Access list (for each file)
- Delete list (for each file)
- Owner list (for each file)

To specify these users in locking dialogs, use comma-separated lists of users and hostnames, with no blank spaces before or after the commas.

Each entry in the list can be in either of the following form(s):

```
<user>@<host>
<host>:<user>
```

The name can be a pattern, which is in turn a regular expression. The pattern matching feature is especially useful with the ‘*’ pattern, which indicates a universal match. If part of the user specification is empty, ‘*’ is assumed. If neither delimiter (@ or :) is used, the user@host form is assumed and ‘*’ is assumed for the host.

For example:
• aonix@* means user "aonix" on any machine.
• aonix@aonix* means user “aonix” on any machine starting with “aonix,” such as aonix@aonix-demo or aonix@aonixhome.

Using Locking Commands

To use the locking commands:

1. From the Ameos desktop, choose System > Locks.
2. From the Locks submenu, choose one of the following locking commands:
   • Set Lock Administrators—To specify which users have authority to override all locks in the current system
   • List Locks—To display lock information for all or specified file(s) in the entire system
   • Manage Locks—To implement and manage locking on selected files
3. Select options and enter appropriate information in the dialogs that appear.

Setting Lock Administrators

By default, the initial lock administrator for an Ameos system is the creator of that system. You can add more users as lock administrators as follows:

1. From the desktop, choose System > Locks > Set Lock Administrators.
2. In the Set Lock Administrators dialog, type entries for all lock administrator(s) for this system, including yourself, using the following format, then click OK. (For details, see “Specifying Users in Access Lists” on page 98.)
   <user>@<host> or <host>:<user>

Listing Locks

To list locks for the current system:

1. From the Ameos desktop, choose System > Locks > List Locks.
2. In the List Locks dialog (below), type the name(s) of the files, minus file extensions, whose lock information you want to display. Leave this field blank to list locks for the entire system. If desired, use Ameos’s pattern matching capabilities to specify the file names.
3. Set other options to further constrain the search to specified lock types or locks set with certain user permissions. Then click **OK**.

The locking information for the specified diagram or table appears in a separate window.

**Using Locking on Selected Files**

You use the **Manage Locks** command to implement and manage locking on specified files.

To use locking on one or more selected files:

1. In the desktop’s model pane, open a category and select a subcategory of diagram types or table types.
2. In the objects pane, select one or more diagrams or tables.
3. Choose **System** > **Locks** > **Manage Locks**.
4. In the **Manage Locks** dialog (below), display and select an available command in the **Command** field.

5. Select options desired. Then click **OK**. Results appear in a separate window.

**Opening a Locked File**

If you try to edit a locked file, Ameos displays a confirmation box, asking if you want to:

- Reset the lock and open the file.
- Open the file in read-only mode.

If a machine crash or other error caused the locked file, you may reset the lock to open the file. Otherwise, you can open a locked file for viewing in read-only mode.
Index

A
Ada 95 code
  parsing (RE) 44
aggregation diagrams
  generated by reverse engineering 52
  multiplicity in reverse engineering diagrams 53
roles in reverse engineering diagrams 53
Ameos
  applications 8
  components of 8
desktop 8
  files, system 9
repository 9
annotations
  generated by reverse engineering 57
  applications, Ameos 8
Assign/Remove Files to Subsystem command 79
association diagrams
  generated by reverse engineering 51
  multiplicity in reverse engineering diagrams 53
roles in reverse engineering diagrams 53
attributes
  effects of rename on dependent objects 96

B
Baseline Current System command 66
Baseline Subsystem command 83
baselines
  comparing 64, 67
  creating 64, 65
  restoring 64, 66
baselining 64
  subsystems 83

C
C code
  embedded code, avoiding parsing (RE) 46
C++ code
  parsing (RE) 38
case sensitivity for object names 9
Check RE Semantic Model Consistency command 48
Check RE Semantic Model Locks command 48
Check Semantics for Subsystem command 82
Check Semantics for Whole Model command 14
  82
class diagrams
  generated by reverse engineering 36, 50
  generated by XMI import 61
class editor
  Attributes and Operations commands 58
  Construct from Reverse Engineering commands 58
Generalization Hierarchy commands 58
classes
  effects of rename on dependent objects 96
ClearCase integration 73
CM
  using external systems for version control 68
code generation
  and the script manager 17
comment extraction (RE)
  described 41
  evaluating and improving 43
  position setting 42
  saving parameters 42
  swear word detection 43
uses of 43
using 42
compiler directives
  parsing source code (RE) 41
Continuous integration 73
Copy Script command 20
Create Script command 20
Create Subclass Hierarchy from Reverse Engineering command 58
Create Superclass Hierarchy from Reverse Engineering command 58
Create Whole Hierarchy from Reverse Engineering command 58

d
data sharing (UNIX and Windows) 9
Delete Script command 20
desktop commands
  Assign/Remove Files to Subsystem 79
  Baseline Subsystem 83
  Check RE Semantic Model Consistency 48
  Check RE Semantic Model Locks 48
  Check Semantics for Subsystem 82
  Check Semantics for Whole Model 82
  Copy Script 20
  Create Script 20
  Delete Script 20
  Edit Script 20
  Edit Value in External Editor 20
  Empty Repository Trash 96
  Import Profiles 13
  Locks commands 99
  Model Management 76
  Open HTML Report 28
  Open Report 27
  Preferences 14
  Properties 76
  Rescan 20
  Reverse Engineer Ada95 Code 44
  Reverse Engineer C++ Code 38
  Reverse Engineer Java Code 37
  Run Script 20, 27
  Set to Default Value 20
  Show Path 20
  Terminate Script 20
desktop commands
  Check Semantics for Whole Model 14
diagrams
  aggregation
    generated by reverse engineering 52
  association
    generated by reverse engineering 51
    generated by XMI import 61
  component
    generated by XMI import 62
dependency
  generated by reverse engineering 52
  generated by XMI import 62
generalization
  generated by reverse engineering 50
  generated by XMI import 61
generated by reverse engineering 48, 49, 50
implementation
  generated by reverse engineering 53
  generated by XMI import 62
nested
  generated by reverse engineering 53
package overview
  generated by XMI import 62
sequence
generated by XMI import 62
state
generated by XMI import 62
DOORS integration 87
enabling 88
navigating from Ameos 89
navigating to Ameos 90
updating renamed objects 90
duagran=ms
aggregation
generated by XMI import 52

E

Eclipse integration 90
navigating to Ameos 92
Edit Script command 20
Edit Value in External Editor command 20
ditor commands
  Rename Systemwide 93
Empty Repository Trash command 96
external variables
  accessible from script manager 20
  providing values for in script sanager 21

F

files
  system 9
folders
  for preprocessed source code (RE) 41
functions
  comment extraction for (RE) 42

G

Generate Model from Parsed Source Files
  command 48

H

hostnames in file lock lists 98
HTML reports 27

I

Import Profiles command 13
#include files
  parsing source code (RE) 41, 47
inheritance diagrams
generated by reverse engineering 50
integrations
  Continuos 73
  DOORS 87
  Eclipse 90
  SNIFF+ 90

J

Java
  environment variables 36
Java code
  parsing (RE) 36

L

locking 97
  access lists 98
  automatic locks 98
  commands 99
  described 98
  lock administrators 99
  lock types 98
  managing locks 100
  permanent locks 98
  viewing a locked file 100
locks
  on semantic model (RE) 48
Locks commands 99

M

makefile reader
  using 40
makefiles
  parsing source code (RE) 41
messages
  renaming 94
Microsoft extensions
  in reverse engineering 39
model elements
  reconfiguring external model elements 85
  updating 85
model management 74
  steps in the process 75
  tree browser 75
Model Management commands 76
model management, see also subsystems
  multiplicity
  in reverse engineering diagrams 53

N

nested diagrams
generated by reverse engineering 53

O

objects
  renaming 94
objects, exporting to DOORS 87
Open HTML Report command 28
Open Report command 27
operations
  effects of rename on dependent objects 96

P

parsing source code (RE)
  Ada 95 44
  C and C++ 39
  compiler directives in Defines file 41
  embedded languages 46
  file specification 49
  include file specification 41
  increasing speed of parser 43
  incremental mode 45
  Java 36
  makefile reader 40
  Microsoft extensions and 39
  preprocessing source files 41
  preprocessor options 40
  troubleshooting 47

Advanced Modeling Concepts
Page 102
version control systems 45
permanent file locks 98
predefined stereotypes 14
preferences 14
Preferences command 14
preferences See also ToolInfo variables
profile editor 11
node symbols 12
node symbols in 12
predefined stereotypes 14
profiles 11
modelling profiles in Ameos 13
reusing profile definitions 14
UML standard profile 12
using predefined profiles 13
projects described 9
Properties command 76
properties window 76

Q
Query and Reporting System 8

R
Rename Systemwide command 93
effects on dependent objects 96
renaming instances 94
renamed objects, with Ameos/DOORS 90
renaming
  global rename 95
  how information is stored 93
  local rename 97
  multiple global rename 97
report generation 26
Report Maker Editor, see RME 28
reports
  HTML reports 27
  RME reports 28
  UML reports 26
repositories
description 9
  sharing data between UNIX and Windows 9
requirements analysis see DOORS integration
Rescan command 20
Restore Selected Baseline System command 66
Restore Subsystem command 73
Reverse Engineer Ada95 Code command 44
Reverse Engineer C++ Code command 38
Reverse Engineer Java Code command 37
reverse engineering
  aggregation diagrams 52
  association diagrams 51
  class tables 56
  comment extraction 41
  generalization diagrams 50
  inheritance diagrams 50
  makefile reader 40
  model generation 48
  nested diagrams 53
  parsing problems 47
  semantic model locks 48
RME
  generating reports with 28
  objects and annotations 32
  roles
    in reverse engineering diagrams 53
Run Script command 20, 27
S
script manager
  adding a site scripts location 24
  Copy Script command 20
  copying a script 21
  Create Script command 20
  creating a script 22
  Delete Script command 20
  deleting a script 23
  deleting an external variable’s value 21
  Edit Script command 20
  Edit Value in External Editor command 20
  editing a script 23
  generating reports 26
  HTML reports 27
  providing values for external variables 21
  Rescan command 20
  Run Script command 20
  running scripts from 20
  script template 23
  Set to Default Value command 20
  Show Path command 20
  Terminate Script command 20
  terminating a script 21
  ToolInfo variables for 24
  UML reports 27
scripts
  copying in script manager 21
  creating in script manager 22
  deleting in script manager 23
  editing in script manager 23
  semantic model
    checking (RE) 48
    lock status of (RE) 48
    locks (RE) 48
  Set to Default Value command 20
  Show Path command 20
  simple_script 23
  SNIFF+ integration 90
  navigating from Ameos 91
  navigating to Ameos 92
stereotypes
defining 13
predefined 14
subsystems
  assigning diagrams to 78, 79
  baselining 83
  checking semantics for 82
  defining 76
  general information 74
  managing 81
systems
described 9
using 8
T
tagged values
  and profile support 11
Terminate Script command 20
third-party integrations see integrations
ToolInfo variables
  for script manager 24
  re_token_file 46, 47
  scriptman_ascii_view 24
  scriptman_product_externals_location 24
  scriptman_run_xterm 25
  scriptman_script_editor 24
  scriptman_script_process 25
  scriptman_system_externals_location 25
  use_doors_integration 98
troubleshooting
code parser (RE) 47

U
UML reports 26
   chapters of 26
   files involved in generating 28
UML standard profile 12
UNIX, sharing data with Windows 9
use cases
   effects of rename on dependent objects 96

V
variables
   ToolInfo, See ToolInfo variables
version control 64
   ClearCase integration 73
   Continuus integration 73
   using external systems 68

W
Windows, sharing data with UNIX 9

X
XMI
diagrams generated during import 61
exporting XMI streams 60
importing XMI streams 60
Parse XMI dialog 61