OpenAmeos

Rose2Ameos Conversion Tool

Rose2Ameos Conversion Tool

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.

US Headquarters
5040 Shoreham Place, Suite 100
San Diego, CA 92122
Phone: (800) 97-AONIX
Fax: (858) 824-0212
E-mail: info@aonix.com

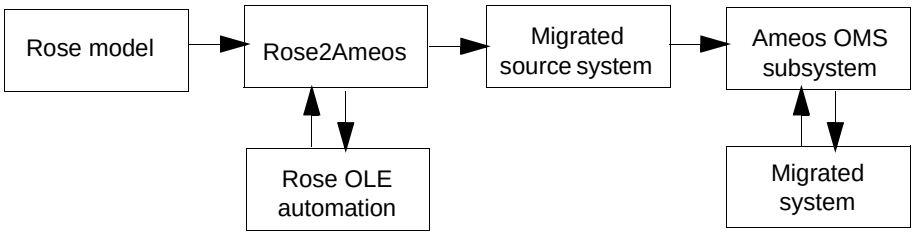
European Headquarters
Batiment B
66/68, Avenue Pierre Brossolette
92247 Malakoff cedex, France
Tel: +33 1 4148-1000
Fax: +33 1 4148-1020
Email: info@aonix.fr

Table of Contents

Introduction

Rose2Ameos is a tool for converting models created with Rose versions 98/98i/2000/2001 into Ameos UML models.

Rose2Ameos uses the Rose Extensibility Interface to retrieve model data from Rational Rose. The Ameos system source flat files are created according to Rose information. The Ameos OMS subsystem is then used to rebuild the Ameos system repository. The process is shown below.



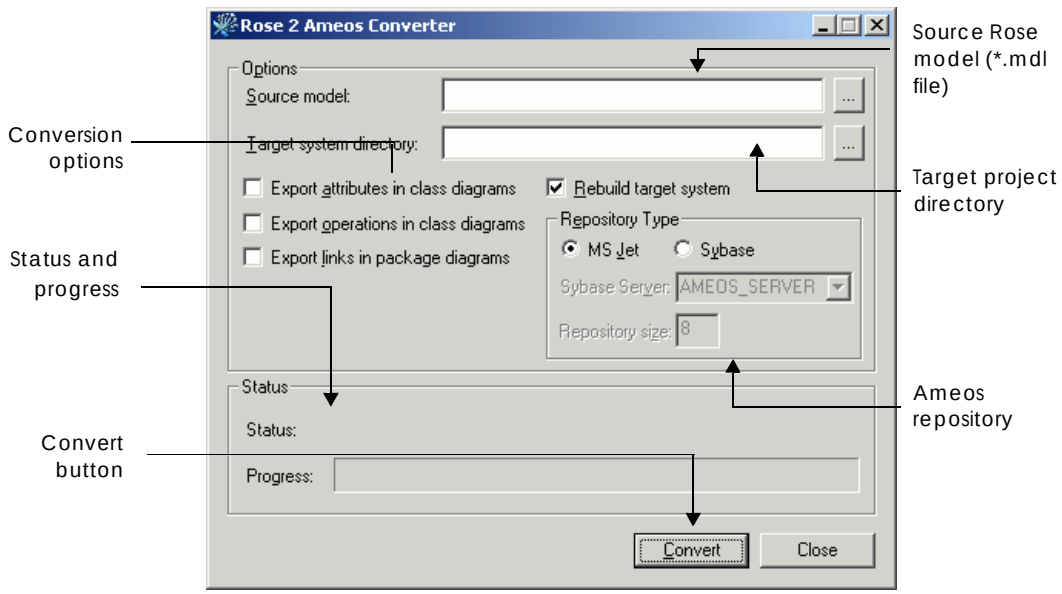
The Rose2Ameos converter exists in *rose2ameos<version>.exe*, in *<Ameos>/bin/<platform>.* It runs on Windows platforms on which Ameos and Rational Rose have been properly installed and are running. Officially supported versions of Rose are 98, 98i, 2000, and 2001.

Using Rose2Ameos

Main Window

When you invoke Rose2Ameos, the main converter window appears (see Figure 1). The callouts in the figure are described below the figure.

Figure 1: Main Converter Window



Source Rose model

Here you enter the full path name of the Rose model (**.mdl* file) you want to convert into an Ameos UML system. Alternatively you can use the browse button to locate the file.

Target project directory

Here you enter the path to the project directory you want to use (typically your Ameos project directory). The name of the Rose system will be used as the Ameos system name.

Conversion options

There are only two conversion options: **Export attributes in class diagrams** and **Export operations in class diagrams**. Attributes or operations of classes are exported in class diagrams only if the corresponding check box is turned on. We recommend that you turn the these two options off, because of the possibility that classes will overlap on migrated diagrams. After migration you can selectively add attributes/operations as necessary from the migrated information that is stored in the class tables.

Ameos repository options

Rose2Ameos dumps the source files of the system into a directory within the target project directory; the name of the directory matches the name of the converted Rose model. To open the system in Ameos, its repository must be built (or rebuilt). Rose2Ameos rebuilds the repository automatically if you check **Rebuild target system**. If the Ameos system does not exist, a new one is created according to the options specified (repository size and type).

Status and progress indication

During conversion various status messages are printed. A progress bar indicates the progress on the current task. There is no progress indication while the repository is being rebuilt.

Convert button

After the above fields are filled in, you press **Convert** to start conversion. At this point all fields become disabled and the **Convert** button becomes **Stop**. By clicking **Stop**, you can interrupt the conversion at any time. When you press **Stop**, the conversion stops at the next step. This could take a while, especially with large Rose models. When the conversion finishes and system rebuild starts, **Stop** will be disabled because the repository rebuild cannot be safely interrupted.

Log Files

During conversion process three log files are generated:

- *RoseConversion.log*. In this log file Rose2Ameos stores warnings about model elements that are not exactly mapped between Rose and Ameos. See "Known Problems" on page 30 for a description of these messages. The file exists in the same directory as the migrated source system flat files.
- *Rebuild.log*. This log is generated only if you checked **Rebuild target system**. If Rose2Ameos sends the message "There were errors while rebuilding the system, see rebuild.log file", the list of errors will be printed in *Rebuild.log*. The file exists in the same directory as the migrated source system flat files.
- *Error. log*. This log file is generated only if Rose2Ameos encounters internal problem (e.g., a protection fault). If you see the message "Internal converter error. See error.log for details", send the file to Aonix Customer Support. The file exists in the directory where Ameos is located (e.g., *c:/Ameos/bin/w32ntx86*).

Rose to Ameos UML Conversion

Since both Ameos and Rose are UML modeling tools, the models designed with them are similar. However, each tool has its peculiarities. Rose2Ameos deals with such peculiarities in a way that makes the target model as close to the original as possible.

Other differences exist as well. The sections below describe the most important differences and how Rose2Ameos resolves them.

Recent Improvements

Starting with StP 8.3, the Rose to Ameos conversion process has been improved. Two of the more significant improvements are:

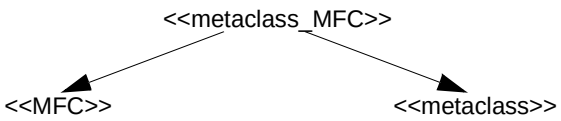
- *Handling of empty diagrams*. Rose2Ameos no longer creates empty files. If, in the Rose model, the properties of an object are not initialized, the annotation file for the Ameos equivalent will not be created. In addition, empty diagrams are eliminated during the conversion process.
- *Handling of classes with the same name in different packages.* Rose, unlike Ameos, allows classes in different packages to have the same name. To accommodate this difference, Rose2Ameos now adds a unique number to the names of converted classes in which the class exists in more than one package. This allows them to be distinguishable in Ameos.

Added Stereotypes

Rose supports some class types that are not available in Ameos. Rose2Ameos converts these class types to the closest class type available in Ameos and sets additional stereotypes so that the semantics are preserved.

If a class already has a stereotype set in the source model, a new stereotype is constructed. The name of the new stereotype is the name of the original stereotype plus the name of the added stereotype with underscore between the two. An artificial stereotype diagram is generated in the Ameos model to describe the new stereotype.

Example: Consider a metaclass in Rose that has a stereotype <<MFC>>. It is converted as a class having stereotype <<metaclass_MFC>>; a stereotype diagram will be added to the model: as shown below.



Artificial Package Diagrams

In a Rose model it is legal for classes to exist without appearing in any class (use case, actors) diagram. Classes (use cases, actors) that do not appear in any diagram might have be related to other classes (use cases, actors), etc. To preserve this information, Rose2Ameos generates a new diagram for each package in the Rose model. These diagrams serve two purposes:

1. They indicate the membership of classes (use cases, actors) in packages.
2. They contain classes (use cases, actors) that do not appear in any of the class (use case) diagrams in the Rose model.

An artificial package diagram has the same name as the parent package had the Rose model. Name conflicts are resolved as stated under “Diagram File Names” in “Known Problems” on page 30.

Unnamed Elements and Diagrams

In a Rose model it is legal for some model elements like classes, use cases, and actors to have void names (to be unnamed). Ameos does not allow multiple unnamed model elements if they cannot be uniquely identified. Rose2Ameos resolves this by generating names for such elements in the form “NonameXXX” where XXX is a unique number. Although unnamed links between two nodes (classes, use cases, etc.) are generally considered unique, there are some exceptions:

- Class diagrams. Associations in class diagrams are distinguished by their names and their roles - if in Rose you have two or more associations with the same name and roles connecting the same pair of classes, a label in the form NonameXXX is generated.

- Use case diagrams. Associations in class diagrams are treated as described for associations in class diagrams.

Bad Character Set for Names of Converted Elements

Rational Rose allows any character to be used when naming model elements and their properties, but Ameos puts some restrictions on the characters that can be used in these cases. In order for Rose2Ameos to produce a correct Ameos UML system, all characters in the elements' names of the migrated model are checked against a list of bad (not allowed) characters. If such characters are found, they are replaced by an underscore ('_').

There are two types of bad character sets – built-in and user defined:

- **Built-in**

  The default bad character set for all model elements is defined by the characters ' ' (space), plus:

  ```
  ! @ $ # " ^ & % ? ’ * . , ; \ / < > | - +
  ```

  There are also some model elements that must have empty bad character sets to preserve their semantics. They are:

UmlComponentSource	UmlBalkingMessage
UmlComponentBinary	UmlTimeoutMessage
UmlComponentExecutable	UmlAsynchronousMessage
UmlComponentInterface	UmlStereotype
UmlState	UmlAssociation
UmlHistoryState	UmlDependency
UmlActionState	UmlGeneralization
UmlSimpleMessage	UmlDeploymentNode
UmlSynchronousMessage	

- **User defined**

  After a Rose model is first converted, Rose2Ameos creates a file *rose2ameos.cfg* in the directory where the Rose2Ameos executable resides. This file can be used to specify bad character sets that override the built-in sets for particular model element or the whole model. This is how the *rose2ameos.cfg* file looks by default:

  ```
  # Default Rose2Ameos configuration file

  Default : " !@$#"^&%?’*.,;\/<>|-+"
  UmlComponentSource : ""
  UmlComponentBinary : ""
  UmlComponentExecutable : ""
  UmlComponentInterface : ""
  UmlState : ""
  UmlHistoryState : ""
  UmlActionState : ""
  UmlSimpleMessage : ""
  UmlSynchronousMessage : ""
  UmlBalkingMessage : ""
  UmlTimeoutMessage : ""
  UmlAsynchronousMessage : ""
  UmlStereotype : ""
  UmlAssociation : ""
  UmlDependency : ""
  UmlGeneralization : ""
  UmlDeploymentNode : ""
  ```

  The file can contain multiple entries in the form <*element type*> : "<*list of bad characters*>", each on new line. Changing this default set overrides the bad character set for all model elements to which the default built-in set applies. Note that if you want to change the bad character set for a model ele-

ment that has a set different from the default one, you must do so explicitly. Here is a list of the valid values for *<element type>*:

.

Class	UmlFinalState	UmlRoleNavigability
Interface	UmlHistoryState	UmlRoleQualifier
Package	UmlActionState	UmlStereotype
ParameterizedClass	UmlMergeControl	UmlTaggedValue
InstantiatedClass	UmlSplitControl	UmlPackage
Attribute	UmlSimpleMessage	UmlActor
Operation	UmlSynchronousMessage	UmlClass
UmlClassDefinition	UmlBalkingMessage	UmlOperation
UmlClassIsAbstract	UmlTimeoutMessage	UmlAttribute
UmlClassType	UmlAsynchronousMessage	UmlAssociation
UmlClassVisibility	UmlMessageDefinition	UmlDependency
UmlClassParameters	UmlSequenceExpression	UmlRefines
UseCase	UmlAggregationType	UmlGeneralization
Actor	UmlAttributeDDLDeclarations	UmlRole
UmlUseCaseDefinition	UmlAttributeIsPrimaryKey	UmlDeploymentNode
UmlUseCaseException	UmlConstraintItem	UmlTransition
UmlUseCasePostcondition	UmlCxxInheritanceDefinition	UmlUseCaseExtends
UmlUseCasePrecondition	UmlCxxInheritanceIsVirtual	UmlUseCaseUses
UmlComponentSource	UmlCxxInheritanceVisibility	UmlUseCaseInteraction
UmlComponentBinary	UmlExtensibilityDefinition	UmlUseCase
UmlComponentExecutable	UmlMultiplicity	UmlObjectClassScope
UmlComponentDependency	UmlOperationCxxCode	UmlObjectInState
UmlComponentInterface	UmlRefineArgs	UmlObjectInstance
UmlStateMachine	UmlRefinesDefinition	State
UmlState	UmlRoleCxxDefinition	Swimlane
UmlInitialState	UmlRoleCxxVisibility	deploymentComponent
	UmlRoleDefinition	

Maps: Rose Elements to Ameos Elements

General

Ameos annotation notes and items get their values from the Rose specification of the migrated object. Class tables are constructed from class specification information. State tables are constructed from state specification information.

The subsections in this section address Rose elements and how they are converted to Ameos UML model elements. For details, refer to:

- “Use Case Diagram” (immediately below)
- “Class Diagram” on page 12
- “Collaboration Diagram” on page 16
- “Sequence Diagram” on page 18
- “State Diagram (Statechart)” on page 19
- “Activity Diagram” on page 20
- “Component Diagram” on page 22
- “Deployment Diagram (Deployment View)” on page 23

Use Case Diagram

Table 1: Use Case Diagram Mappings

Rose element	Ameos element
Package	
Package	Package
Property Sheet for Package:	
General > Name	Annotation of the Package > Object > Name
General > Stereotype	Annotation of the Package > Extensibility Definition > Stereotype
General > Documentation	Annotation of the Package > Object: Description
Detail, Files, C++	Not migrated
Actor	
Actor	Actor
Property sheet for Actor:	
General > Stereotype	Not migrated If the stereotype differs from “Actor”, the element is migrated as class in a class diagram with the corresponding stereotype.
General > Export control	Not migrated If the stereotype differs from “Actor”, the element is migrated as class in a class diagram. Check the Class category in “Class Diagram Mappings” on page 12.
General > Documentation	Annotation of the Actor > Object: Description
Detail > Cardinality Detail > Size Detail > Persistence Detail > Concurrency Detail > Abstract Detail > Formal Arguments Operation Attribute Components Relations Nested Files	Not migrated An actor in Rose is converted into its corresponding use case diagram in Ameos. If the object has attributes, annotations, or other class-specific properties, they are converted as *class* in another use case diagram and data in its respective class table and annotation. See the class category in “Class Diagram Mappings” on page 12
Use Case	
Use Case	Use Case
Property sheet for Use Case:	
General > Name	Annotation of the Use Case > Object > Name
General > Stereotype	Annotation of the Use Case > Extensibility Definition > Stereotype
General > Rank General > Abstract	Not migrated
General > Documentation	Annotation of the Use Case > Object: Description
Diagrams, Relations, Files	Not migrated

Table 1: Use Case Diagram Mappings (Continued)

Dependency	
Dependency	Actor Inheritance
Property sheet for Use Case:	
General > Name	Not migrated
General > Stereotype	Annotation of the Dependency > Extensibility Definition > Stereotype
General > Friendship required General > Export Control General > Cardinality from/to	Not migrated
General > Documentation	Annotation of the Dependency > Object: Description
Association	
Association	Between Actor and Use Case - Communicates Between Actor and Actor - Actor Inheritance Between Use Case and Use Case - Include
Property sheet for Association	
General > Name	Between Actor and Use Case - Communicates > Name Between Actor and Actor - Not migrated Between Use Case and Use Case - Not migrated
General > Stereotype	Between Actor and Use Case - Communicates > Stereotype Between Actor and Actor - Not migrated Between Use Case and Use Case - Include > Stereotype
General > Role A/B	Between Actor and Use Case - Not migrated Between Actor and Actor - Not migrated Between Use Case and Use Case - Not migrated
General > Documentation	Between Actor and Use Case - Communicates > Description Between Actor and Actor - Actor Inheritance > Description Between Use Case and Use Case - Include > Description
Detail Role A/B General Role A/B Detail	Not migrated
Generalization	
Generalization between Actor and Actor	Actor Inheritance
Generalization between Use Case and Use Case	Extends
Property sheet for Generalization:	
General > Name	Not migrated
General > Stereotype	Annotation of the Dependency > Extensibility Definition > Stereotype
General > Export Control General > Cardinality from/to	Not migrated
General > Documentation	Annotation of the Dependency > Object: Description
General > Friendship required General > Virtual inheritance	Not migrated

Class Diagram

Table 2: Class Diagram Mappings

Rose element	Ameos element
Package	
Package	Package
Property sheet for Package:	
General > Name	Annotation of the Package > Object > Name
General > Stereotype	Annotation of the Package > Extensibility Definition > Stereotype
General > Documentation	Annotation of the Package > Object: Description
Detail, Files, C++	Not migrated
Class	
Class	Class
ParameterizedClass	Parameterized Class
InstantiatedClass	Instantiated Class
ClassUtility	Normal Class with stereotype <<Utility>> If the ClassUtility in Rose has a stereotype <<rose_stereotype>>, Rose2Ameos migrates the class as a normal class with stereotype <<rose_stereotype_utility>> and a stereotype diagram, showing <<rose_stereotype_utility>> inheriting the <<utility>> and <<rose_stereotype>> stereotypes.
ParameterizedClassUtility	Parameterized Class with stereotype <<Utility>> If the ParameterizedClassUtility in Rose has a stereotype <<rose_stereotype>>, Rose2Ameos migrates the class as a parameterized class with stereotype <<rose_stereotype_utility>> and a stereotype diagram, showing <<rose_stereotype_utility>> inheriting the <<utility>> and <<rose_stereotype>> stereotypes.
InstantiatedClassUtility	Instantiated Class with stereotype <<Utility>> If the InstantiatedClassUtility in Rose has a stereotype <<rose_stereotype>>, Rose2Ameos migrates the class as an instantiated class with stereotype <<rose_stereotype_utility>> and a stereotype diagram, showing <<rose_stereotype_utility>> inheriting the <<utility>> and <<rose_stereotype>> stereotypes.
Interface	UmlInterface + UmlClass with stereotype <<interface>> Additional class diagram is generated for each Rose interface, containing the interface (as a interface bubble), the classes which implement it and implements links between them and the interface.
MetaClass	Normal Class with stereotype <<metaclass>> If the MetaClass in Rose has a stereotype <<rose_stereotype>>, Rose2Ameos migrates the class as a normal class with stereotype <<rose_stereotype_metaclass>> and a stereotype diagram, showing <<rose_stereotype_metaclass>> inheriting the <<metaclass>> and <<rose_stereotype>> stereotypes.

Table 2: Class Diagram Mappings (Continued)

Property sheet for Class:	
General > Export control	Annotation of the class > Class Definition > Class Visibility If the value is set to “Implementation” in Rose, Rose2Ameos migrates it as “private” class visibility.
General > Stereotype	Annotation of the Class > Extensibility Definition > Stereotype
General > Documentation	Annotation of the Class > Object: Description
Detail > Cardinality	Annotation of the class > Class Definition > Multiplicity The symbol “n” in Rose is replaced by “*” in Ameos; for example, “1..n” becomes “1..*”
Detail > Size Detail > Persistence Detail > Concurrency	Not migrated
Detail > Abstract	Annotation of the class > Class Definition > Abstract Class
Detail > Formal Arguments	Annotation of the class > Class Definition > Parameters
Operation	Operation
Operation > Stereotype	Annotation of the operation > Extensibility Definition > Stereotype
Operations > return type	Class Table > Operation’s return type
Attribute	Attribute
Attribute > Stereotype	Annotation of the attribute > Extensibility Definition > Stereotype
Attribute > Type	Class Table > Attribute’s type
Attribute > Initial	Class Table > Attribute’s default value
Nested, File, C++, MSVC	Not migrated
Property sheet for Operation:	
General > Name	Class Table > Operation’s Name
General > Return Class	Class Table > Operation’s return type
General > Export control	Class Table > Operation’s Analysis Items > Visibility. If the value is set to “Implementation” in Rose, Rose2Ameos migrates it as “private”.
General > Stereotype	Annotation of the Operation > Extensibility Definition > Stereotype
General > Documentation	Annotation of the Operation > Object: Description
Detail > Arguments	Class Table > Operation’s Arguments
Detail > Protocol Detail > Qualification	Not migrated
Detail > Exceptions	Class Table > Operation’s Analysis Items > Throws
Detail > Size Detail > Time Detail > Concurrency Precondition Semantics Postcondition Files	Not migrated

Table 2: Class Diagram Mappings (Continued)

C++ > OperationKind	- Virtual: Class Table > Operation's C++ Items > Virtual? - Static: Class Table > Operation's C++ Items > Const? - Common: Not migrated - Abstract: Not migrated - Friend: Not migrated
C++ > OperationIsConstant C++ > OperationIsExplicit	Not migrated
C++ > Inline	Class Table > Operation's C++ Items >Inline?
MSVC	Not migrated
Property sheet for Attribute:	
General > Name	Class Table > Attribute's name
General > Type	Class Table > Attribute's type
General > Stereotype	Annotation of the Attribute> Extensibility Definition > Stereotype
General > Initial Value	Class Table > Attribute's default value
General > Export control	Class Table > Attribute's analysis items > Visibility. If the value is set to "Implementation" in Rose, Rose2Ameos migrates it as "private".
General > Documentation	Annotation of the Attribute > Object: Description
Detail > Containment	Not migrated
Detail > Static	Class Table > Attribute's Analysis Items > Class Attr?
Detail > Derived	Class Table > Attribute's Analysis Items > Derived?
DDL	Not migrated
C++ > DataMemberVisibility	Class Table > Attribute's C++ Items > Visibility. "Implementation" is migrated as "private". "AtAttributeVisibility" sets it identical to Class Table > Attribute's Analysis Items > Visibility
C++ > DataMemberMutability	- Const: Class Table > Attribute's C++ Items > Const? - Unrestricted: Not migrated - Mutable: Not migrated
C++ > DataMemberIsViolate	Class Table > Attribute's C++ Items > Violate?
MSVC	Not migrated
Generalization link	
Generalization	Generalization
Property sheet for Generalization:	
General > Name	Annotation of the Generalization > Object > Name
General > Stereotype	Annotation of the Generalization > Extensibility Definition > Stereotype

Table 2: Class Diagram Mappings (Continued)

General > Export control	Annotation of the Generalization > C++ Inheritance Definition > Inheritance Visibility If the value is set to “Implementation” in Rose, Rose2Ameos migrates it as “private”.
General > Documentation	Annotation of the Generalization > Object: Description
General > Friendship Required	Not migrated
General > Virtual inheritance	Annotation of the Generalization > C++ Inheritance Definition > C++ Inheritance Is Virtual
C++	Not migrated
Association/Aggregation link	
Association	Association
Property sheet for Association/Aggregation:	
General > Name	Annotation of the Association > Object > Name
General > Stereotype	Annotation of the Association > Extensibility Definition > Stereotype
General > Role A Element A:XXX	A role to the diagram’s element XXX
General > Role B Element A:YYY	A role to the diagram’s element YYY
General > Documentation	Annotation of the Association > Object: Description
Detail > Derived Detail > Link Element	Not migrated
Detail > Name direction	Sets “Role Definition > Role Navigability” to False for the role, which is in the opposite direction to the Name’s direction
Detail > Constraints	Annotation of the Association > Extensibility Definition > Constraint
Role A/B General > Role	Annotation of the role A/B > Object > Name
Role A/B General > Export control	Annotation of the role A/B > role’s C++ Definition > Visibility If the value is set to “Implementation” in Rose, Rose2Ameos migrates it as “private”.
Role A/B General > Documentation	Annotation of the role A/B > Object: Description
Role A/B Detail > Constraints	Annotation of the role A/B > Extensibility Definition > Constraint
Role A/B Detail > Cardinality	Annotation of the role A/B > Role Definition > Multiplicity The symbol “n” in Rose is replaced by “*” in Ameos; for example, “1..n” becomes “1..*”.
Role A/B Detail > Navigable	Annotation of the role A/B > Role Definition > Role Navigability
Role A/B Detail > Aggregate	Annotation of the role A/B > Role Definition > Aggregation Type:Aggregation
Role A/B Detail > Static Role A/B Detail > Friend Role A/B Detail > Containment of A/B	Not migrated
Role A/B Detail > Keys/Qualifiers	Annotation of the role A/B > Role Definition > Qualifiers

Table 2: Class Diagram Mappings (Continued)

C++ C++ A/B MSVC MSVC A/B	Not migrated
Dependency link	
Dependency	Dependency
Property sheet for Dependency:	
General > Name	Annotation of the Dependency> Object > Name
General > Stereotype	Annotation of the Generalization > Extensibility Definition > Stereotype
General > Friendship Required	Sets the Dependency's stereotype to <<friend>> If the Dependency in Rose has a stereotype <<rose_stereotype>>, Rose2Ameos changes the stereotype to <<friend_the_rose_stereotype>> and creates a stereotype diagram, showing <<friend_the_rose_stereotype>> inheriting the <<friend>> and <<rose_stereotype>> stereotypes.
General > Export control General > Cardinality From: General > Cardinality To:	Not migrated
General > Documentation	Annotation of the Generalization > Object: Description
C++	Not migrated
nstantiates link	
Instantiates	Binds (UmlRefines link with stereotype <<binds>>) The annotation of the Binds link is set according to the information in the Rose property sheet of the instantiated class (IC) from which the Instantiates link begins.
IC Detail > Actual Arguments	Annotation of the Binds > Refinement Definition > Refinement Arguments Argument with name <AAA>, type <BBB> and value <CCC> is migrated as "<AAA>:<BBB>=<CCC>"
Association Class link	
Association Class link	UmlAssociationClassLink
Realize link	
Realize	Implements

Collaboration Diagram

Table 3: Collaboration Diagram Mappings

Rose element	Ameos element
Object/MultiObject	
Object	Object or MultiObject In Ameos the symbol Object is mapped to three repository objects: UmlObjectInstance, UmlObjectInState and UmlObjectClassScope.
Property sheet for Object/MultiObject:	

Table 3: Collaboration Diagram Mappings (Continued)

General > Name	Annotation of the UmlObjectInstance > Object > Name
General > Class	Annotation of the UmlObjectClassScope > Object > Name
General > Documentation	Annotation of the UmlObjectInstance > Object: Description
General > Persistence	Not migrated
General > Multiple Instances	If checked, migrates the Rose collaboration diagram object as an Ameos MultiObject
Class Instance	
Class Instance	Object In Ameos the symbol Object is mapped to three repository objects: UmlObjectInstance, UmlObjectInState, and UmlObjectClassScope.
Property sheet for Class Instance:	
General > Name	Annotation of the UmlObjectInstance > Object > Name
General > Class	Annotation of the UmlObjectClassScope > Object > Name
General > Documentation	Annotation of the UmlObjectInstance > Object: Description
Object Link	
Object link	Object link
Property sheet for Object link:	
General Messages	Not migrated
Link Message/Reverse Link Message	
Link Message / Reverse Link Message	Message In Ameos separate repository types represent the different types of messages. See property sheet below.
Property sheet for Link Message/Reverse Link Message:	
General > Name	Annotation of the Message > Object > Name
General > Documentation	Annotation of the Message > Object: Description
Detail > Synchronization	- Simple: UmlSimpleMessage - Synchronous: UmlSynchronousMessage - Balking: UmlBalkingMessage - Timeout: UmlTimeoutMessage - Asynchronous: UmlAsynchronousMessage
Detail > Frequency	Not migrated
Data Flow/Reverse Data Flow	
Data Flow/ Reverse Data Flow	Not migrated

Sequence Diagram

Table 4: Sequence Diagram Mappings

Rose element	Ameos element
Object	
Object	Passive object
Property sheet for Object:	
General > Name	Annotation of the UmlObjectInstance > Object > Name
General > Class	Annotation of the UmlObjectClassScope > Object > Name
General > Documentation	Annotation of the UmlObjectInstance > Object: Description
General > Persistence General > Multiple Instances	Not migrated
Object Message	
Object Message	Message In Ameos separate repository types represent the different types of messages. See the property sheet below.
Property sheet for Object Message:	
General > Name	Annotation of the Message > Object > Name
General > Documentation	Annotation of the Message > Object: Description
Detail > Synchronization	- Simple: UmlSimpleMessage - Synchronous: UmlSynchronousMessage - Balking: UmlBalkingMessage - Timeout: UmlTimeoutMessage - Asynchronous: UmlAsynchronousMessage
Detail > Frequency	Not migrated
Message to Self	
Message to Self	Message In Ameos separate repository types represent the different types of messages. See the property sheet below.
Property sheet for Message to Self:	
General > Name	Annotation of the Message > Object > Name
General > Documentation	Annotation of the Message > Object: Description
Detail > Synchronization	- Simple: UmlSimpleMessage - Synchronous: UmlSynchronousMessage - Balking: UmlBalkingMessage - Timeout: UmlTimeoutMessage - Asynchronous: UmlAsynchronousMessage
Detail > Frequency	Not migrated

State Diagram (Statechart)

Table 5: State Diagram Mapping

Rose element	Ameos element
State	
State	State
Property sheet for State:	
General > Name	Annotation of the State > Object > Name
General > Stereotype	Annotation of the State > Extensibility Definition > Stereotype
General > Documentation	Annotation of the State> Object: Description
General > State/Activity History	Migrated as separate Shallow History
General > Sub State/Activity History	Migrated as separate Deep History
Actions > Type:Entry	State table > Entry Action
Actions > Type:Exit	State table > Exit Action
Actions > Type:Do	State table > Activity
Actions > Type:On	State table > Internal Event and Internal Action
Transitions Swimlanes	Not migrated
Start State	
Start State	Initial State
Property sheet for Start State:	
General > Name	Annotation of the State > Object > Name
General > Documentation	Annotation of the of the State> Object: Description
Actions, Transitions, Swimlanes	Not migrated
End State	
End State	Final State
Property sheet for End State:	
General > Name	Annotation of the State > Object > Name
General > Documentation	Annotation of the State> Object: Description
Actions, Transitions. Swimlanes	Not migrated
State Transition/Transition to self	
State Transition or Transition to self	State Transition
Property sheet for State Transition:	
General > Trigger Event	Annotation of the State Transition > Object > Part of the name Accessible through the ‘Event’ section in state transition’s property sheet
General > Arguments	

Table 5: State Diagram Mapping (Continued)

General > Stereotype	Annotation of the State Transition > Extensibility Definition > Stereotype
General > Documentation	Annotation of the State Transition> Object: Description
Detail > Guard condition	Annotation of the State Transition > Object > Part of the name Accessible through the 'Guard' section in state transition's property sheet
Detail > Action	
Detail > Send Event	
Detail > Send arguments	
Detail > Send target	Not migrated
Decision	
Decision	Converted to a junction point in the corresponding Ameos state diagram (Ameos 8.3 and beyond only; otherwise not migrated

Activity Diagram

Table 6: Activity Diagram Mappings

Rose element	Ameos element
Activity	
Activity	Action State
Property sheet for State:	
General > Name	Annotation of the Action State > Object > Name
General > Stereotype	Annotation of the Action State > Extensibility Definition > Stereotype
General > Documentation	Annotation of the of the Action State> Object: Description
General > State/Activity History General > Sub State/Activity History Actions > Type:Entry Actions > Type:Exit Actions > Type:Do Actions > Type:On Transitions Swimlanes	Not migrated
State	
State	Action State
Property sheet for State:	
General > Name	Annotation of the Action State > Object > Name
General > Stereotype	Annotation of the Action State > Extensibility Definition > Stereotype
General > Documentation	Annotation of the of the Action State> Object: Description

Table 6: Activity Diagram Mappings (Continued)

General > State/Activity History General > Sub State/Activity History Actions > Type:Entry Actions > Type:Exit Actions > Type:Do Actions > Type:On Transitions Swimlanes	Not migrated
Decision	
Decision	Decision
Object Instance	
ObjectInstance (Rose 2000 and beyond)	Object
ObjectFlowLink (Rose 2000 and beyond)	Input Link
State Transition/Transition to self	
State Transition/Transition to self	State Transition
Property sheet for State Transition:	
General > Trigger Event General > Arguments	Not migrated
General > Stereotype	Annotation of the State Transition > Extensibility Definition > Stereotype
General > Documentation	Annotation of the State Transition> Object: Description
Detail > Guard condition	Annotation of the State Transition > Object > Part of the name Accessible through the ‘Guard’ section in state transition’s property sheet
Detail > Action	Annotation of the State > Object > Part of the name Accessible through the ‘Action list’ section in state transition’s property sheet
Detail > Send Event Detail > Send arguments Detail > Send target	Not migrated
Horizontal synchronization	
Horizontal synchronization	Split/Synchronize Control
Vertical synchronization	
Vertical synchronization	Split/Synchronize Control
Swimlane	
Swimlane	Swimlane

Component Diagram

Table 7: Component Diagram Mappings

Rose element	Ameos element
Package	
Package	Package
Property sheet for Package:	
General > Name	Annotation of the Package > Object > Name
General > Stereotype	Annotation of the Package > Extensibility Definition > Stereotype
General > Documentation	Annotation of the Package > Object: Description
Detail, Files, C++	Not migrated
Component	
Component	Source/Binary/Executable Component
Property sheet for Component:	
General > Name	Annotation of the Component > Object > Name
General > Stereotype	Stereotype: <None> - Binary Component Stereotype: ActiveX - Binary Component with stereotype ‘activex’ Stereotype: Applet - Binary Component with stereotype ‘applet’ Stereotype: Application - Executable Component with stereotype ‘ application’ Stereotype: DLL - Binary Component with stereotype ‘dll’ Stereotype: EXE - Binary Component with stereotype ‘exe’ Stereotype: Generic Package - Source Component with stereotype ‘generic_package’ Stereotype: Generic Subprogram - Source Component with stereotype ‘generic_subprogram’ Stereotype: Main Program - Source Component with stereotype ‘main_program’ Stereotype: Package Body - Source Component with stereotype ‘package_body’ Stereotype: Package Specification - Source Component with stereotype ‘package_specification’ Stereotype: Subprogram Body - Source Component with stereotype ‘subprogram_body’ Stereotype: Subprogram Specification - Source Component with stereotype ‘subprogram_specification’ Stereotype: Task Body - Source Component with stereotype ‘task_body’ Stereotype: Task Specification - Source Component with stereotype ‘task_specification’
General > Documentation	Annotation of the Component > Object: Description
General > Language Detail Realizes Files	Not migrated

Table 7: Component Diagram Mappings (Continued)

Dependency	
Dependency	Dependency

Deployment Diagram (Deployment View)

Table 8: Deployment Diagram Mappings

Rose Element	Ameos Element
Processor	
Processor	Deployment Node with stereotype <<processor>>
Property Sheet for Processor:	
General > Name	Annotation of the Processor > Object > Name
General > Stereotype	If the Processor in Rose has a stereotype <<rose_stereotype>>, Rose2Ameos migrates the Deployment Node with stereotype <<processor_the_rose_stereotype>> and a stereotype diagram, showing <<rose_stereotype_metaclass>> inheriting the <<processor>> and <<rose_stereotype>> stereotypes.
General > Documentation	Annotation of the Processor > Object: Description
Detail	Not migrated
Device	
Device	Deployment Node with stereotype <<device>>
Property sheet for Device:	
General > Name	Annotation of the Processor > Object > Name
General > Stereotype	If the Device in Rose has a stereotype <<rose_stereotype>>, Rose2Ameos migrates the Deployment node with stereotype <<device_rose_stereotype>> and a stereotype diagram, showing <<rose_stereotype_metaclass>> inheriting the <<device>> and <<rose_stereotype>> stereotypes.
General > Documentation	Annotation of the Processor > Object: Description
Detail	Not migrated
Connection	
Connection (Rose 2000 and beyond)	‘Communicates’ link

Migrating Customized Property Sheet Information

The standard Rose2Ameos conversion process, described up to this point, is not capable of migrating all of the customized information available in the property sheets of Rose model elements due to the differences between the modeling concepts of Ameos and Rational Rose. In the case of converting large, complex systems (which usually contain customized information about model elements in order to overcome the limitations of standard UML), problems with lost semantics can arise in converted Ameos systems.

Rose2Ameos is able to solve this problem by giving you an option for generating user customizations for the annotations of Ameos model elements based on the information contained in the Rose model.

The Conversion Process

The actual process consists of four main stages:

1. “Migrating User Customizations” immediately below
2. “Editing the Migrated Customizations” on page 28
3. “Configuring Ameos to Use the Migrated Customizations” on page 29
4. “Actual Migration of the Rose System” on page 29

Migrating User Customizations

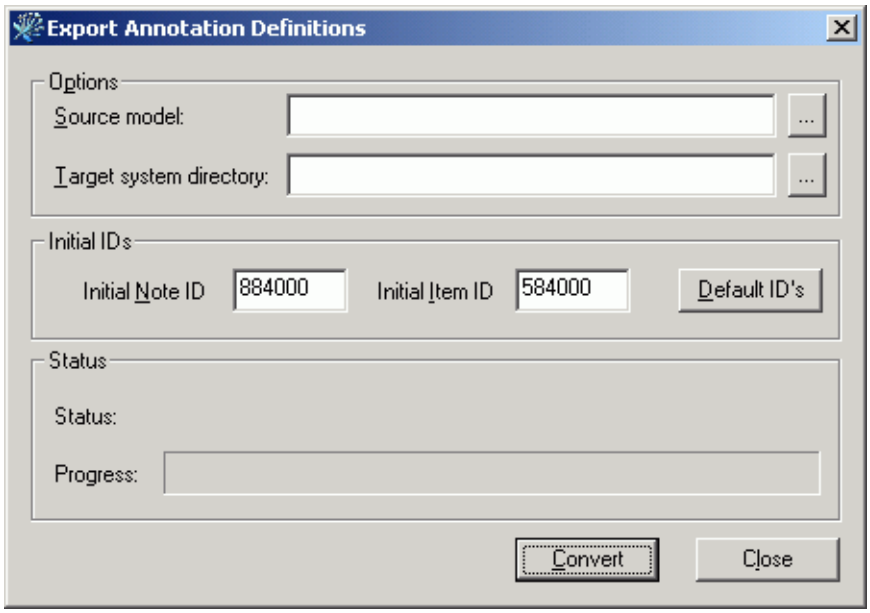
This stage uses the Rose capability to add additional information about model elements by defining so-called *properties*. These properties are grouped by so-called *tools*. Rose adds some predefined sets of tools and properties in various files with the extension *.pty*; an example is the file *.../c++/rosecpp.pty*, which holds more than 400 properties. Here is an excerpt of the additional information for an operation:

```
(object Attribute
                tool            "cg"
                name            "default__Operation"
                value           (list Attribute_Set
                    (object Attribute
                        tool            "cg"
                        name            "CodeName"
                        value           "")
                    (object Attribute
                        tool            "cg"
                        name            "OperationKind"
                        value           ("OperationKindSet" 200))
                    (object Attribute
                        tool            "cg"
                        name            "OperationKindSet"
                        value           (list Attribute_Set
                            (object Attribute
                                tool            "cg"
                                name            "Common"
                                value           200)
                            (object Attribute
                                tool            "cg"
                                name            "Virtual"
                                value           201)
...
                            (object Attribute
                                tool            "cg"
                                name            "Friend"
                                value           204)))
...
                    (object Attribute
                        tool            "cg"
                        name            "Inline"
                        value           FALSE)
                    (object Attribute
                        tool            "cg"
                        name            "EntryCode"
                        value           (value Text ""))
...
                )
                )
```

The excerpt above shows some properties of the tool ‘cg’ defined for an operation. The prefix “default_” means that this is the default set. Rose can define different sets of properties depending on the stereotypes attached to an element; the converter is only able to migrate the default set.

To begin the migration of user customizations, you must invoke Rose2Ameos with the command line parameter “-genannot” or by setting the global environment variable `IDE_ROSE2AMEOS_GENANNOT` to 1. The main window (below) is displayed.

.



Pressing the **Convert** button starts the process. When conversion is over, there will be several files in the flat file directory that Rose2Ameos created in the *Target* directory. Below is a short description of each file:

- *RoseTranslation.txt*

  This file contains the mapping definition defining the translation of the Rose properties to Ameos notes and items. Here is an example file:

```
//------------------------------------------------------
###
### Rose Tool-Properties for IRoseCategory
###

#
# Tool cg PropertySet: default PropertyClass: Category
IRoseCategory:UmlPackage:cg:IsNamespace:cgNote:IsNamespaceItem
IRoseCategory:UmlPackage:cg:Indent:cgNote:IndentItem
IRoseCategory:UmlPackage:cg:CodeName:cgNote:CodeNameItem
IRoseCategory:UmlPackage:cg:GenerateEmptyRegions:cgNote:GenerateEmptyRegionsItem



###
### Rose Tool-Properties for IRoseClass
###

#
# Tool IDL PropertySet: default PropertyClass: Class
IRoseClass:UmlClass:IDL:ImplementationType:IDLNote:ImplementationTypeItem
IRoseClass:UmlClass:IDL:ConstValue:IDLNote:ConstValueItem
IRoseClass:UmlClass:IDL:GenerateDefaultSpecifier:IDLNote:GenerateDefaultSpecifierItem
IRoseClass:UmlClass:IDL:DefaultSpecifier:IDLNote:DefaultSpecifierItem
IRoseClass:UmlClass:IDL:IDLElement:IDLNote:IDLElementItem
IRoseClass:UmlClass:IDL:IDLSpecificationType:IDLNote:IDLSpecificationTypeItem
. . .
//------------------------------------------------------
```

  In general a definition line of the file looks like this:

<Rose-type>:*<Ameos-type>*:*<Rose-tool >*:*<Rose-property>*:*<Ameos-note>*[:*<optional-Ameos-item>*]

Be aware that some Rose types map to multiple Ameos types; for example, an InheritRelation in Rose is transformed to Ameos UmlGeneralization or UmlRefines, depending on the context. Therefore the Rose properties for IRoseInheritRelation might be found more than once in different Ameos types. It is up to the user to decide weather to migrate a specific Rose property to both UmlGeneralization and UmlRefines, or to migrate it only to one of these two Ameos types.

- *user_oms_data_model*

  This file contains the data type definitions needed for choice values in Ameos. Here is an example file:

```
//--------------------------------------------------
//
// Item enumeration definitions of user added extensions
//
//
// Append the content to templates\user\ct\oms\user_oms_data_model
//

ItemValType GenerateEmptyRegionSet
{
    { ChoiceType
        { Choice "None" }
        { Choice "Preserved" }
        { Choice "Unpreserved" }
        { Choice "All" }
    }
}
ItemValType IDLSpecSet
{
    { ChoiceType
        { Choice "Interface" }
        { Choice "Typedef" }
        { Choice "Enumeration" }
        { Choice "Const" }
        { Choice "Exception" }
        { Choice "Struct" }
        { Choice "Union" }
    }
}
ItemValType GenerateSet
{
    { ChoiceType
        { Choice "DeclareAndDefine" }
        { Choice "DeclareOnly" }
        { Choice "DoNotDeclare" }
    }
}
ItemValType VisibilitySet
{
    { ChoiceType
        { Choice "Public" }
        { Choice "Protected" }
        { Choice "Private" }
        { Choice "Implementation" }
    }
}
ItemValType GenerateSet
{
    { ChoiceType
        { Choice "DeclareAndDefine" }
        { Choice "DeclareOnly" }
        { Choice "DoNotDeclare" }
```

```
    }
}
ItemValType VisibilitySet
{
    { ChoiceType
        { Choice "Public" }
        { Choice "Protected" }
. . .
. . .
. . .
//------------------------------------------------------
```

- *user_app.types*

  This file contains the OMS type definitions for Ameos's notes and items. Here is an example file:

```
//------------------------------------------------------
// Item and Note definitions of user added extensions
//
//
// Append the content to templates\user\ct\oms\user_app.types
//

NoteObject 884000 cgNote {}
ItemObject 584000 IsNamespaceItem {{ DataType "Boolean" }}
ItemObject 584001 IndentItem {{ DataType "Int" }}
ItemObject 584002 CodeNameItem {{ DataType "String" }}
ItemObject 584003 GenerateEmptyRegionsItem {{ DataType "GenerateEmptyRegionSet" }}
NoteObject 884001 IDLNote {}
ItemObject 584004 ImplementationTypeItem {{ DataType "String" }}
ItemObject 584005 ConstValueItem {{ DataType "String" }}
ItemObject 584006 GenerateDefaultSpecifierItem {{ DataType "Boolean" }}
ItemObject 584007 DefaultSpecifierItem {{ DataType "String" }}
ItemObject 584008 IDLElementItem {{ DataType "Boolean" }}
ItemObject 584009 IDLSpecificationTypeItem {{ DataType "IDLSpecSet" }}
ItemObject 584010 ClassKeyItem {{ DataType "String" }}
ItemObject 584011 PutBodiesInSpecItem {{ DataType "Boolean" }}
ItemObject 584012 GenerateDefaultConstructorItem {{ DataType "GenerateSet" }}
ItemObject 584013 DefaultConstructorVisibilityItem {{ DataType "VisibilitySet" }}
ItemObject 584014 InlineDefaultConstructorItem {{ DataType "Boolean" }}
ItemObject 584015 ExplicitDefaultConstructorItem {{ DataType "Boolean" }}
ItemObject 584016 GenerateCopyConstructorItem {{ DataType "GenerateSet" }}
ItemObject 584017 CopyConstructorVisibilityItem {{ DataType "VisibilitySet" }}
. . .

//------------------------------------------------------
```

- *user_<OMS type>*

  This set of files contains definitions for OAE to annotate various Ameos elements. Here are twoexamples of these files:

  Example 1: user_UmlClass:

```
//------------------------------------------------------
//
// Rose Tool-Properties for IRoseClass
//
// Add the following part to the annotation definition of UmlClass
// this is usual in templates\user\ct\annotation\user_UmlClass
//


Note 884001
{
    { AppType IDLNote }
    { HelpText "Rose tool IDL" }
    { PrintName "IDL" }
```

```
    { Bounds { Min 0 } { Max 1 } }
    { Item
        { AppType ImplementationTypeItem }
        { HelpText "Rose String Property ImplementationType" }
        { PrintName "ImplementationType" }
        { InitValue "" }
        { Bounds { Min 0 } { Max 1 } }
    }
    { Item
        { AppType ConstValueItem }
        { HelpText "Rose String Property ConstValue" }
        { PrintName "ConstValue" }
        { InitValue "" }
        { Bounds { Min 0 } { Max 1 } }
    }
    { Item
        { AppType GenerateDefaultSpecifierItem }
        { HelpText "Rose Boolean Property GenerateDefaultSpecifier" }
        { PrintName "GenerateDefaultSpecifier" }
        { InitValue "" }
        { Bounds { Min 0 } { Max 1 } }
    }
    { Item
        { AppType DefaultSpecifierItem }
        { HelpText "Rose String Property DefaultSpecifier" }
        { PrintName "DefaultSpecifier" }
. . .
//------------------------------------------------------
```

Example 2: user_UmlAssociation

```
//------------------------------------------------------
//
// Rose Tool-Properties for IRoseAssociation
//
// Add the following part to the annotation definition of UmlAssociation
// this is usual in templates\user\ct\annotation\user_UmlAssociation
Note 884000
{
    { AppType cgNote }
    { HelpText "Rose tool cg" }
    { PrintName "cg" }
    { Bounds { Min 0 } { Max 1 } }
    { Item
        { AppType NameIfUnlabeledItem }
        { HelpText "Rose String Property NameIfUnlabeled" }
        { PrintName "NameIfUnlabeled" }
        { InitValue "" }
        { Bounds { Min 0 } { Max 1 } }
    }
}
//------------------------------------------------------
```

Editing the Migrated Customizations

Note: You can edit all files above.

Deleting/commenting out lines from *RoseTranslation.txt* causes the deleted/commented out properties to be ignored and not migrated to the Ameos system. The only things that should not be edited are the names of the Rose tools and properties: it seems that unknown names cause Rose to crash, preventing successful subsequent model conversion. If name changes of notes and/or items are made, they need to be made in every file. This can be useful in case of Rose type, tools, or property names containing char-

acters that are illegal in Ameos.

As shown above (page 26), *RoseTranslation.txt* has an optional sixth column [:<*optional-Ameos-item*>]. This column is not generated when the type of the Rose property is set to “Text”, which in Rose is a multi-line text container. It is up to you to choose to migrate this “Text” property into an Ameos item. This can be done by simply adding this column. Again, this additional item must be added to the corresponding other files too. On the other hand, it is also possible to remove the Item column when the content of the Rose property should go into the description field of an Ameos note.

Configuring Ameos to Use the Migrated Customizations

To make the migrated user customizations recognizable to Ameos, the generated files, described in “Migrating User Customizations” on page 24, should be copied to the locations listed below:

- *RoseTranslation.txt* (page 25). Keep this file in the flat-files directory.
- *user_oms_data_model* (page 26). Copy or append this file to *templates/user/ct/oms/user_oms_data_model.*
- *user_app.types* (page 27). Copy or append this file to *templates/user/ct/oms/user_app.types*.
- *user_<OMS type>* (page 27). Copy or append these files to *templates/user/ct/annotation*.

Actual Migration of the Rose System

Finally Rose2Ameos needs to be started without the “-genannot” command line parameter. In this case, migration of the source Rose system is performed as described in “Using Rose2Ameos” on page 5.

Converting Multiple Rose Models with Different Customizations

Rose2Ameos provides a mechanism that can be used for merging different customizations from multiple Rose models. Each time Rose2Ameos is started with “-genannot”, it reconstructs the OMS IDs for notes and items. By doing so, every migrated custom property has a unique OMS ID, as long as you do not reset the ID to its default (884000 for notes and 584000 for items). When you want to add new customizations to the set that is already migrated, you must compare the entries from *user_oms_data_model*, *user_app.types*, *user_<OMS type>* in your target flat files folder with the ones, found in *templates/user/ct/oms/* and *templates/user/ct/annotation*, extract the ones that have a unique OMS ID and name and are missing in the current valid set, and then append those entries to the corresponding files in *templates/user/ct/oms/* and *templates/user/ct/annotation*.

Some Background Information

Note: Numbers in the range of 880000 - 899999 and item numbers within 584000 - 599999 are reserved for user added customizations; an offset of 4000 is used to avoid conflicts with already existing customizations.

When migrating without a *RoseTranslation.txt* file, the tool behaves as before and none of the additional user-defined properties are migrated to Ameos.

The generated files contain only Rose properties that are actually used in the model. If, for example, the Rose model does not contain any use cases, the tool will not generate any prototype for Rose properties attachable to a use case. We therefore recommend you use Rose2Ameos with “-genannot” - with the largest system to be migrated, not with some small test system.

The tool migrates to Ameos only those Rose properties that have values that differ from the defaults. In other words, only properties that are explicitly set in Rose are migrated.

Dealing with Large Systems

When converting very large systems it is possible for the rebuild of the repository to fail with a logged message in the *rebuild.log* file:

```
Arc [or Node] allocation failed, store the diagram, reload and try again
```

This could happen if there are many arcs and/or nodes in some diagrams of the target Rose model. To fix the problem, open your ToolInfo file and add the following two lines:

```
init_max_nodes=1000
init_max_arcs=1000
```

The default values are 300 for `init_max_nodes` and 200 for `init_max_arcs`. If you use 1000 and the rebuild still fails with the same error, try 2000, 4000, etc.

Known Problems

Diagram file names: Diagram file names in Ameos are the same as the names in the Rose model. However, Rose has a hierarchical structure, which allows diagrams with identical names to exist on different levels. Rose2Ameos deals with this by adding numbers at the end of file names on lower level diagrams to preserve their uniqueness.

For example, a Rose model with

- Logical view (package)
- Main (class diagram)
- Package 1 (package)
- Main (class diagram)

will produce in Ameos two class diagrams – Main and Main1.

Notes: Notes in activity diagrams in Rose 98i are not imported.

Links: In Rose, links can be contained in a model but not shown in any diagram. For all associations not contained in a diagram, a warning message is generated in *RoseConversion.log*:

```
Warning:  Association <name> between classes <class1> and <class2> does not exist in any diagram
and it is not converted
```

Note links: Rose Extensibility Interface does not provide information on links from/to note symbols. That is why all not all links in the diagrams are converted.

Note symbols: In Rose, note symbols can contain <tab> characters; in Ameos they cannot. In StP 8.3 and beyond, the symbols are converted as spaces. (Previously they were converted as empty rectangles.)

Nested classes: Ameos does not support nested classes.

Constraints: Rose Extensibility Interface provides constraints information only for association roles and for associations in Rose98i. That is why only constraints for association roles and associations are converted to Ameos. All other objects are converted without constraints.

Overlapping links: If there are multiple links between two model elements in the migrated Rose model, the migrated links will overlap in the resulting Ameos diagram. If the number of links is an even number, they will not be visible. To fix this problem in Ameos, add vertexes to the links until all of them are visible.

Component diagram elements overlap: If there are multiple symbols with a common name in component diagram of the migrated Rose model, the symbols in the target Ameos system overlap and appear as invisible if they are an even number.

Links in deployment diagrams: Connections in deployment diagrams are migrated as “communicates” links in Rose version 2000 and later. In Rose 98 and 98i, there is no way to retrieve a ConnectionRelation name; the name is converted as ProcessorOrDevice1_ProcessorOrDevice2.

Packages in component diagrams: Rose supports packages in component diagrams while Ameos does not. That is the reason why packages in component diagrams are missing from the converted system. A warning is logged in *RoseConverion.log* for such packages.

Dependency between packages in use case diagrams: Rational Rose supports dependency links between packages in use case diagrams. Ameos does not. A warning is logged in *RoseConversion.log* for those links.

Different instances of actors in interaction diagrams: Rose supports different instances of actors in interaction diagrams. Ameos supports only one instance of an actor. So in the converted diagram there is only one actor. A warning is logged in *RoseConversion.log* for these actor instances.

Different views of the same symbol: Different views of one symbol in Rose diagrams are converted as one symbol. All links to/from the symbol are connected to this single view. The semantics are correct, but the Ameos diagram looks different from the Rose diagram.

Activity state: There is no activity diagram in Rose98. Rose98i, however, supports activity diagrams with activity states. Ameos does not have activity states. The activity states from Rose are represented by action states in Ameos’s activity diagrams. A warning is logged in *RoseConversion.lo*g.

Links between two actors in interaction diagrams: Models can contain messages between actor and actor in sequence/collaboration diagrams. Rose2Ameos migrates systems containing such interactions, but during the rebuild error messages are generated:

- For collaboration diagrams: “Invalid Object Link connection between Actor symbol and Actor symbol”
- For sequence diagrams: “Invalid Simple Message connection between Actor symbol and Actor symbol”

Inconsistent/corrupt Rose files: Warning messages are produced when you attempt to build an Ameos system from inconsistent or corrupt Rose files.

The Rose model file (**.mdl*) contains properties information. These properties can have UserDefined types that are defined in two parts: name and type. Each entry has a number. This number is used by Rose to store the assigned value in the model file. As long as the number stored for some object matches one of the numbers stored in the definition of the enumeration type, Rose can translate this number into the appropriate string.

Sometimes property numbers get illegal values for unknown reasons. When the Rose interface finds an illegal value, it encloses it in parentheses (e.g., 203) before passing it to the application.

Since the Rose2Ameos converter cannot figure out if this is a valid value, it creates an annotation file containing an illegal value. Later, when the Ameos repository is created from the files, the program update produces a message saying that the annotation file contains an illegal value for an item.

Designing “Bridge Ready” Models in Rose

Rose2Ameos will convert any Rose model into Ameos. However, some models produce better results. The converted systems are more readable and easier to continue working with in Ameos.

Below is a list of recommendations for design of Rose98 and Rose98i systems so you get more accurate Ameos systems.

Leave more distance between symbols in diagrams. When drawing diagrams, set the nodes far from each other and/or give them short names to avoid overlapping in Ameos. This is highly recommended for classes with many operations and attributes. If this is not an option, turn off “Export attributes/operations in class diagrams” checkboxes before migrating.

Use straight links. Links consisting of more than one line (polylines) in Rose are converted as straight lines. This can make the Ameos diagrams hard to read. Try to design Rose diagrams where symbols are connected only with straight lines to get a better visual image in Ameos.

Avoid links in deployment diagrams. In Rose 98 and 98i, links in deployment diagrams cannot be converted. They are replaced by explanatory text boxes between the nodes. This could make the converted diagram hard to read. In Rose 2000 and beyond, these links are converted as "communicates" link, so avoiding links is unnecessary.

Avoid blanks in symbol names. Ameos does not allow blanks in names of some symbols (e.g., class, use case). To fix this, Rose2Ameos converts blanks to underscores. See "Bad Character Set for Names of Converted Elements" on page 8.

Avoid unnamed symbols. Ameos does not allow unnamed model elements for certain model element types (classes, use cases, etc.). Rose2Ameos sets a name in the form `NonameXX` to these unnamed model elements. This preserves the semantics of the source model, but the target model is hard to read and some links could be misplaced. See "Unnamed Elements and Diagrams" on page 7.

Use unique Attributes and Operation names within a class. Rational Rose allows two different attributes/operations to have the same name. Ameos does not allow this. So, if possible try to keep attributes unique by name.

Avoiding Note links. Links from notes to diagram elements and vice versa are not exported by Rose Extensibility Interface. Try not to rely on them for explanations of diagram elements. The best thing to do is to place the notes near the symbol they are related to.

Reflexive association roles must have different names. If the roles of a reflexive association have the same name, Ameos cannot distinguish the roles. Use different names for those roles to preserve the semantics of the model.

Dependency links with multiplicity. Dependency links with multiplicity in Rational Rose that are meant to be associations cannot be converted to dependencies with multiplicity. We recommend that you replace the dependency links with associations in Rose to get a more accurate Ameos system.