Rational Software Corporation

Toolset Guide

RATIONAL ROSE® REALTIME

VERSION: 2002.05.00

PART NUMBER: 800-025116-000

WINDOWS/UNIX

Rational the software development company

support@rational.com
http://www.rational.com

Contents

Preface

Contents

This chapter is organized as follows:

This document captures important information you need to use the Rational Rose RealTime toolset after it has been installed. It is available as part of the comprehensive online help system and in hardcopy.

Online Help

Comprehensive online help is available in MS HTML Help format. As such, it includes four navigation tabs: Contents, Index, Search, and Favorites. There are numerous hyperlinks between related topics. As well, there are multimedia demos of how to use specific features of the toolset.

Activating the Online Help

To activate the online help, select one of the items from the **Help** menu.

What's This Help

Context sensitive help is available from the toolset. You can access this help three ways:

- by clicking the Help Contents button
- by selecting the Context Sensitive Help to open help on particular topic
- by pressing Shift + F1

Extended Help

Extended Help leads you to information in the Rational Unified Process and other sources about how to perform tasks using the current tool. The other sources can be organization-specific, project-specific standards and guidelines, or almost any other type of information that can be represented in a HTML document. The information accessed from Extended Help depends on the context from which it is invoked. The context varies from task to task.

Extended Help is driven by a set of databases that identify the context for each piece of information (called the target). One database is delivered with the Rational Unified Process and is registered upon installation. You may create and register any number of databases, each containing different information and pointers to different HTML pages. For example, you may have a database that contains pointers to pages that are appropriate for all projects in your organization. You may have another one that is specific to a project or project type. You might also decide to create a personal database with pointers to information that is important to you.

Tutorials

The online help provides tutorials to help you learn to use the main features of Rational Rose RealTime.

About the Help Viewer

The following topics describe most of the general features available in the Help Viewer:

- Getting More Out of Help
- To find a Help Topic
- To create a list of favorite help topics
- To copy a help topic
- To print the current help topic
- To get help in a dialog
- To find topics using the toolbar buttons
- To hide or show the Navigation pane
- Using accessibility shortcut keys in the Help Viewer
- Using the shortcut menu commands

Getting More Out of Help

Here are some tips on how to find more information when using the HTML Help Viewer:

- To link to another topic, a Web page, a list of other topics, or a program, click the colored, underlined words.
- To view topics that contain related information, click topic titles under the headings "Related Topics," "Related Tasks," and "See Also," which may appear at the end of a topic.
- To see if a word or phrase contained in a topic is in the index, select the word, and then press F1.
- If you are viewing content from the Web in the Topic pane, you can click Stop or Refresh on the toolbar to interrupt a download or refresh a Web page.
- If you use a particular help topic often, you can add it to your favorites list.
- Right-click the Contents tab or Topic pane for shortcut menu commands.

To find a Help Topic

In the Navigation pane, click one of the following tabs:

- To browse through a table of contents, click the Contents tab. The table of contents is an expandable list of important topics.
- To see a list of index entries, click the Index tab, and then type a word or scroll through the list. Topics are often indexed under more than one entry.
- To locate every occurrence of a word or phrase that may be contained in a help file, click the Search tab, and then type the word.

Notes

Click the contents entry, index entry, or search results entry to display the corresponding topic.

To create a list of favorite help topics

1. Locate the help topic you want to make a favorite topic.
2. Click the **Favorites** tab, and then click **Add**.

Notes

- To return to a favorite topic, click the **Favorites** tab, select the topic, and then click **Display**.
- If you want to rename a topic, select the topic, and then type a new name in the **Current topic** box.
- To remove a favorite topic, select the topic and then click **Remove**.

To copy a help topic

1. In the Topic pane, right-click the topic you want to copy, and then click **Select All**.
2. Right-click again, and then click **Copy**. This copies the topic to the Clipboard.
3. Open the document you want to copy the topic to.
4. Position your cursor where you want the information to appear.
5. On the **Edit** menu, click **Paste**.

Notes

If you want to copy only part of a topic, select the text you want to copy, right-click, and then click **Copy**.

To print the current help topic

- Right-click a topic, and then click **Print**.

Notes

If you print from the **Contents** tab (by right-clicking an entry, and then clicking **Print**) you will see options to print only the current topic, or the current topic and all subtopics.

To get help in a dialog

- Click the question mark in the upper-right corner of the dialog, and then click an item in the dialog.

Notes

- To close the pop-up window, click anywhere on the screen.
- If the dialog does not have the question mark, click **Help** or press F1.
- You can also get help on an item by right-clicking it.
- Not all dialogs include dialog help.

To find topics using the toolbar buttons

There are five navigational buttons that can be located on the toolbar in the Help Viewer. You can click these buttons to find help topics:

- **Back** displays the last topic you viewed.
- **Forward** displays the next topic in a previously displayed sequence of topics.
- **Home** displays the Home page topic for the help file you are viewing.
- **Refresh** updates Web content that is currently displayed in the Topic pane.
- **Stop** stops downloading file information. Click this button to stop a Web page from downloading.

Notes

The toolbar in your Help Viewer may not contain all of these navigational buttons.

To hide or show the Navigation pane

On the toolbar, click **Hide** or **Show** to close or display the Navigation pane, which contains the **Contents**, **Index**, **Search**, and **Favorites** tabs.

Notes

If you close the **Help Viewer** with the Navigation pane hidden, it will appear that way when you open the Help Viewer again.

Using accessibility shortcut keys in the Help Viewer

The following keyboard shortcuts can be used for navigation in the **HTML Help Viewer**. The help author who builds a compiled help (.chm) file can specify which buttons appear on the **Help Viewer** toolbar, so some of these options may not be available in your version of the viewer.

For the Help Viewer:

To	Press
Close the Help Viewer.	ALT+F4
Switch between the Help Viewer and other open windows.	ALT+TAB
Display the Options menu.	ALT+O
Change Microsoft Internet Explorer settings. The **Internet Options** dialog box contains accessibility settings. To change these settings click the **General** tab, and then click **Accessibility**.	ALT+O, and then press I
Hide or show the Navigation pane.	ALT+O, and then press T
Print a topic.	ALT+O, and then press P, or right-click in the
Move back to the previous topic.	ALT+LEFT ARROW, or ALT+O, and then press B
Move forward to the next topic (provided you have viewed it just previously).	ALT+RIGHT ARROW, or ALT+O, and then press F
Turn on or off search highlighting.	ALT+O, and then press O
Refresh the topic that appears in the Topic pane (this is useful if you have linked to a Web page).	F5, or ALT+O, and then press R
Return to the home page (help authors can specify a home page for a help system).	ALT+O, and then press H
Stop the viewer from opening a page (this is also useful if you are linking to the Web and want to stop a page from downloading).	ALT+O, and then press S
Jump to a predetermined topic or Web page. The help author who builds a compiled help (.chm) file can add two links, on the **Options** menu, to important topics or Web pages. When you select a **Jump** command you go to one of those topics or Web pages.	ALT+O, and then press 1 or 2

Switch between the Navigation pane and the Topic pane.	F6
Scroll through a topic.	UP ARROW and DOWN ARROW, or PAGE UP and PAGE DOWN
Scroll through all the links in a topic or through all the options on a Navigation pane tab.	TAB

For the Contents tab:

To	**Press**
Display the **Contents** tab.	ALT+C
Open and close a book or folder.	PLUS SIGN and MINUS SIGN, or LEFT ARROW and RIGHT ARROW
Select a topic.	DOWN ARROW and UP ARROW
Display the selected topic.	ENTER

For the Index tab:

To	**Press**
Display the **Index** tab.	ALT+N
Type a keyword to search for.	ALT+W, and then type the word
Select a keyword in the list.	UP ARROW and DOWN ARROW
Display the associated topic.	ALT+D

For the Search tab:

To	**Press**
Display the **Search** tab.	ALT+S
Type a keyword to search for.	ALT+W, and then type the word
Start a search.	ALT+L

Select a topic in the results list.	ALT+T, and then UP ARROW and DOWN ARROW
Display the selected topic.	ALT+D
The following options are only available if full-text search is enabled.	
Search for a keyword in the result list of a prior search.	ALT+U
Search for words similar to the keyword. For example, to find words like "running" and "runs" for the keyword "run."	ALT+M
Only search through topic titles.	ALT+R

For the Favorites tab:

To	**Press**
Display the **Favorites** tab.	ALT+I
Add the currently displayed topic to the Favorites list.	ALT+A
Select a topic in the Favorites list.	ALT+P, and then UP ARROW and DOWN ARROW
Display the selected topic.	ALT+D
Remove the selected topic from the list.	ALT+R

Notes

- There are also shortcut menu commands that can be accessed through the keyboard.
- Shortcut keys also work in secondary and pop-up windows.
- Every time you use a shortcut key in the Navigation pane, you lose focus in the Topic pane. To return to the Topic pane, press F6.
- The Match similar words check box, on the Search tab, will be selected if you used it for your last search.

Using the shortcut menu commands

There are several commands on the shortcut menu that you can use to display and customize information.

Command	**Description**
Right-click in the table of contents, and then click **Open All**.	Opens all books or folders in the table of contents. This command only works if the **Contents** tab is displayed.
Right-click in the table of contents, and then click **Close All**.	Closes all books or folders. This command only works if the **Contents** tab is displayed.
Right-click, and then click **Print**.	Prints the topic.
Right-click in the table of contents, and then click **Customize**	Opens the Customize Information Wizard, which allows you to customize the documentation. If the help file was built with information types, you can use this wizard to select a subset of topics to view. For example, you could choose to see only overview topics.

Notes

- These commands can be accessed through the keyboard. You can click SHIFT+F10 to display the shortcut menu, and then click the appropriate shortcut keys. Or, you can enable Mousekeys. Use a Mousekey combination to display the shortcut menu, and then click the appropriate shortcut keys.

About the Search tab

The **Search** tab that allows you to search through every word in a help file to find a match. For example, if you do a full-text search on the word "index," every topic that contains the word "index" will be listed.

To use full-text search

1 Click the **Search** tab, and then type the word or phrase you want to find.

2 Click **List Topics**, select the topic you want, and then click **Display**.

To highlight words in searched topics

When searching for words in help topics, you can have each occurrence of the word or phrase highlighted in the topics that are found.

To highlight all instances of a search word or phrase, click **Options** on the toolbar, and then click **Search Highlight On**.

Notes

- To turn off this option, click **Options** on the toolbar, and then click **Search Highlight Off**.
- If you are viewing a long topic, only the first 500 instances of a search word or phrase will be highlighted.

Searching for help topics

A basic search consists of the word or phrase you want to find. You can use wildcard expressions, nested expressions, boolean operators, similar word matches, a previous results list, or topic titles to further define your search.

The basic rules for formulating queries are as follows:

- Searches are not case-sensitive, so you can type your search in uppercase or lowercase characters.
- You may search for any combination of letters (a-z) and numbers (0-9).
- Punctuation marks such as the period, colon, semicolon, comma, and hyphen are ignored during a search.
- Group the elements of your search using double quotes or parentheses to set apart each element. You cannot search for quotation marks.

Notes

If you are searching for a file name with an extension, you should group the entire string in double quotes, ("filename.ext"). Otherwise, the period will break the file name into two separate terms. The default operation between terms is AND, so you will create the logical equivalent to "filename AND ext."

To find information with advanced full-text search

1. Click the **Search** tab, and then type the word or phrase you want to find.
2. Click to add boolean operators to your search.
3. Click **List Topic**s, select the topic you want, and then click **Display**.
4. To sort the topic list, click the Title, Location, or Rank column heading.

Notes

- You can precisely define a search by using wildcard expressions, nested expressions, and boolean operators.
- You can request similar word matches, search only the topic titles, or search the results of a previous search.
- You can set the Help Viewer to highlight all instances of search terms that are found in topic files. Click the Options button, and then click Search Highlight On. This feature only works with Internet Explorer 4.0 or later.

Searching for words or phrases

You can search for words or phrases and use wildcard expressions. Wildcard expressions allow you to search for one or more characters using a question mark or asterisk. The table below describes the results of these different kinds of searches.

Search for	Example	Results
A single word	select	Topics that contain the word “select.” (You will also find its grammatical variations, such as “selector” and “selection.”)
A phrase	“new operator” or new operator	Topics that contain the literal phrase “new operator” and all its grammatical variations. Without the quotation marks, the query is equivalent to specifying “new AND operator,” which will find topics containing both of the individual words, instead of the phrase.
Wildcard expressions	esc* or 80?86	Topics that contain the terms “ESC,” “escape,” “escalation,” and so on. The asterisk cannot be the only character in the term. Topics that contain the terms “80186,” “80286,” “80386,” and so on. The question mark cannot be the only character in the term.

Notes

- Select the Match similar words check box to include minor grammatical variations for the phrase you search.

Defining search terms

The AND, OR, NOT, and NEAR operators enable you to precisely define your search by creating a relationship between search terms. The following table shows how you can use each of these operators. If no operator is specified, AND is used. For example, the query “spacing border printing” is equivalent to “spacing AND border AND printing.”

Search for	Example	Results
Both terms in the same topic.	dib AND palette	Topics containing both the words “dib” and “palette.”
Either term in a topic.	raster OR vector	Topics containing either the word “raster” or the word “vector” or both.
The first term without the second term.	ole NOT dde	Topics containing the word “OLE,” but not the word “DDE.”
Both terms in the same topic, close together.	user NEAR kernel	Topics containing the word “user” within eight words of the word “kernel.”

Notes

- The |, &, and ! characters don't work as boolean operators (you must use OR, AND, and NOT).

Using nested expressions when searching

Nested expressions allow you to create complex searches for information. For example, “control AND ((active OR dde) NEAR window)” finds topics containing the word “control” along with the words “active” and “window” close together, or containing “control” along with the words “dde” and “window” close together.

The basic rules for searching help topics using nested expressions are as follows:

- You can use parentheses to nest expressions within a query. The expressions in parentheses are evaluated before the rest of the query.
- If a query does not contain a nested expression, it is evaluated from left to right. For example: “Control NOT active OR dde” finds topics containing the word “control” without the word “active,” or topics containing the word “dde.” On the other hand, “control NOT (active OR dde)” finds topics containing the word “control” without either of the words “active” or “dde.”
- You cannot nest expressions more than five levels deep.

To search for words in the titles of HTML files

1 Click the **Search** tab, type the word or phrase you want to find, and then select the **Search titles only** check box.

2 Click **List Topics**, select the topic you want, and then click **Display**.

Notes

- If you use this option, all HTML topic files will be searched, including any that are not listed in the table of contents.

To find words similar to your search term

This feature enables you to include minor grammatical variations for the phrase you search. For example, a search on the word “add” will find “add,” “adds,” and “added.”

1 Click the **Search** tab, type the word or phrase you want to find, and then select the **Match similar words** check box.

2 Click **List Topics**, select the topic you want, and then click **Display**.

Notes

- This feature only locates variations of the word with common suffixes. For example, a search on the word “add” will find “added,” but it will not find “additive.”

To search only the last group of topics you searched

This feature enables you to narrow a search that results in too many topics found. You can search through your results list from previous search by using this option.

1. On the **Search** tab, select the **Search previous results** check box.
2. Click **List Topics**, select the topic you want, and then click **Display**.

Notes

- If you want to search through all of the files in a help system, this check box must be cleared.
- The **Search** tab will open with this check box selected if you previously used this feature.

Changing the Help Viewer

Users can make a variety of changes to the Help Viewer. These related topics describe settings the user can specify.

To customize the Help Viewer

There are a few ways to easily change the size and position of the Help Viewer and the panes in the viewer:

- To resize the Navigation or Topic pane, point to the divider between the two panes. When the pointer changes to a double-headed arrow, drag the divider right or left.
- To proportionately shrink or enlarge the whole Help Viewer, point to any corner of the Help Viewer. When the pointer changes to a double-headed arrow, drag the corner.
- To change the height or width of the Help Viewer, point to the top, bottom, left, or right edge of the Help Viewer. When the pointer changes to a double-headed arrow, drag the edge.
- To reposition the Help Viewer on your screen, click the title bar and drag the Viewer to a new position.

Notes

- The Help Viewer will appear with the last size and position settings you specified when it is opened again.

To change formatting or styles for accessibility

1. On the **Options** menu, click **Internet Options**, and then click **Accessibility**.
2. In the **Accessibility** dialog box, select the options you want, and then click **OK**.

Notes

- These changes do not apply to the Navigation pane or toolbar of the Help Viewer.
- This will also change your accessibility settings for Internet Explorer 4.0.

To view help topics grouped by information type

You can customize your help system so that it includes only those help topics that are relevant to you.

Suppose for example, that you have a help system for an educational software program that includes topics aimed at administrators, teachers, and students. You can customize your help so that it includes only the topics that are important to teachers and students.

On the toolbar, click **Options**, and then click **Customize**.

To change the font size of a topic

On the **Options** menu, click **Internet Options**, and then click **Fonts**.

Notes

- These changes do not apply to the Navigation pane or toolbar of the Help Viewer.
- This will also change your font settings for Internet Explorer.

To change colors in the Topic pane of the Help Viewer

1. In Microsoft Internet Explorer, on the **View** menu, click **Internet Options**.
2. On the **General** tab, click **Colors**.
3. In the **Colors** dialog box, select the options you want, and then click **OK**.
4. To apply the new color settings, in the **Internet Options** dialog box, click **OK**.

Notes

- These changes do not apply to the Navigation pane or toolbar of the Help Viewer.
- This will also change your color settings for Internet Explorer 4.0.

Related Documentation

The following documents are related to the *Rational Rose RealTime Toolset Guide*:

- *Installation Guide*
- *Modeling Language Guide*
- *Guide to Team Development Guide*

Overview of Rational Rose RealTime 1

Contents

This chapter is organized as follows:

Overview

Rose RealTime is a software development environment tailored to the demands of real-time software. Developers use Rose RealTime to create models of the software system based on the Unified Modeling Language constructs, to generate the implementation code, compile, then run and debug the application.

Rose RealTime can be used through all phases of the software development lifecycle, from initial requirements analysis through design, implementation, test and final deployment. It provides a single interface for model-based development that integrates with other tools required during the different phases of development. For example, developers work directly through Rose RealTime to generate and compile the code that implements the model. The actual compilation is performed behind the scenes by a compiler/linker outside of the toolset.

Using Rose RealTime, developers work at a higher level of abstraction specifying behavior in state diagrams and communication relationships in collaboration diagrams. This is a natural and logical evolution in computer languages. Just as third generation language tools provided greater productivity than assembly language coding, visual development tools provide significant productivity gains over current third generation languages.

Rose RealTime includes features for:

- creating UML models using the elements and diagrams defined in the UML
- generating complete code implementations (applications) for those models
- executing, testing and debugging models at the modeling language level using visual observation tools
- using Change Management systems for team development

The first three of these topics are covered in this guide; the last topic is covered in the *Guide to Team Development.*

Languages and Code Generation

Code generation of models is provided by specialized language add-ins. The content of the generated code, regardless of the language, is based on

- the specification of each model element
- the values of the model properties that are attached to model elements

Language add-ins provide custom properties that store language-specific information for each model element. In addition to Rose, Rose RealTime add-ins also provide a Services Library that provides support for specialized real-time services such as concurrency, message passing, and timing services. The Services Library is shipped as a library file and is linked into every executable created with Rose RealTime.

By providing both code generation and the specialized Services Libraries Rose RealTime allows you to not only generate but also compile and run models.

Compilation

The compilation of models generated from Rose RealTime is done using commercial compilers and linkers. Rose RealTime generates the code then calls the compiler and linker to compile and link the generated source code with the pre-compiled Services Library. Rose RealTime does NOT ship with a compiler/linker, these must be installed before you can build and run a model.

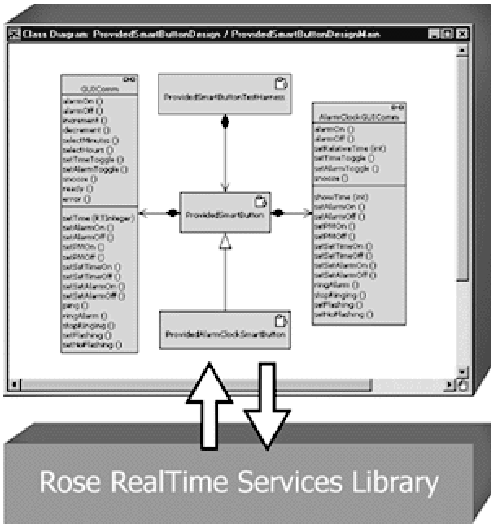
Services Library

In order to construct a functioning Rose RealTime model two major parts are required:

- the structure and behavior of the model
- the Rose RealTime Services Library

The relationship between these two parts is shown in Figure 1.

Figure 1 The services library is a framework for real-time systems



The services library is essentially a framework for real-time systems. It includes functionality for controlling concurrent execution of finite state machines, for delivering messages, and for providing timing and logging services. A framework is like a library of classes and operations used by an application, but with an inversion of control, meaning that the main control lies in the framework, and the framework invokes functions in the application to pass control to application objects as required. Application classes are subclassed from framework classes so that they inherit certain operations.

There is no **main()** function in a Rose RealTime model. The **main()** function is contained in the Services Library and takes care of creating the capsules in your model and kicking off the execution of their state machines. All you need to do is describe the capsules and define state machines for them and they will be automatically created and executed by the services library. The capsule state machines can, in turn, invoke operations on other classes (data classes), and send messages to other capsules. The Services Library is responsible for managing the creation and destruction of capsules, and the delivery of messages between capsules (even across threads).

The addition of these real-time notations to the UML concepts allows the toolset to generate complete code for the model which is tied in to the services library. When you generate code for and compile a model in Rose RealTime, the tool will link it with a Services Library compiled for the language and particular platform you are running on.

Capsules, Protocols, Ports, Capsule State and Structure Diagrams

In addition to supporting the core UML constructs, Rose RealTime uses the extensibility features of the UML to define some new constructs that are specialized for real-time system development. These new constructs allow code generation of elements that can use the services provided in the Services Library, such as concurrent state machines, concurrency, message passing, and timing services.

In fact, many real-time projects must implement most of the above services. Using the added modeling elements in Rose RealTime allows you to concentrate on implementing the functionality of the system right away without having to hand-code the common real-time services and concurrency support.

Capsules

- A capsule is a stereotype of a class.

  Has much of the same properties as regular classes with added semantics for modeling of communication relationships between capsules and modeling of its event based behavior using a state diagram.

- Provides built-in support for light weight concurrent objects.

  Because of the message based nature and high encapsulation of capsules, they can be easily distributed to different physical threads of control without any change to the capsule.

- Highly encapsulated objects using message based communication to other capsules via its port objects.

  The advantage of the message-based interfaces is that a capsule has no knowledge of its environment outside of these interfaces, making it a much more **distributable**, **reusable**, and **robust** than regular objects.

- Capsule can aggregate other capsules.

  Like classes, a capsules structure is defined by its attributes (encapsulation of objects of other types of classes). But it can also be defined by attributes that are other capsules, which we call **capsule roles**.

Protocols

- Defines the set of messages exchanged between a set of capsules.
- Messages are defined from the perspective of both the receiver and the sender.

  There are therefore different perspectives of a protocol, which we call protocol roles. Protocol roles represent the communication from the perspective of one participant in the communication scenario.

- Messages that are sent between capsules contain a required signal name (which identifies the message), an optional priority (relative importance of this message compared to other unprocessed messages on the same thread), and optional application data.

Ports

- Ports are objects whose purpose is to send and receive messages to and from capsule instances.
- They are owned by the capsule instance in the sense that they are created along with their capsule and destroyed when the capsule is destroyed.
- To specify which messages can be sent to and from a port, a port realizes a protocol role. The protocol role is the specification of a set of the messages that can be received (in) and sent (out) from the port. The protocol role essentially defines the port type.

State Diagrams

- Uses the same notation as defined in the UML.
- Are generated to source code and make up the behavior of capsules.
- All trigger events are defined by a port and signal pair. A capsule's behavior is therefore based on the receipt of messages.
- Final states are not allowed on capsule state diagrams.
- Junction points do not support the continuation kind attribute; that is, if a transition is not continued, it defaults to history (except for internal transitions).

Capsule Structure Diagrams

- A new diagram has been introduced the specify the capsule's interface (ports) and its internal composition (capsule roles). The diagram is called a capsule structure diagram, and it is based on the UML 1.3 specification collaboration diagram.
- This is a specification type of diagram, and not an interaction diagram (object) as collaboration diagrams in other versions of Rose are.
- Allow you to specify the communication relationships between capsules.

Executable Models

The addition of the capsule and the formal semantics surrounding the capsule structure allows Rose RealTime to generate, compile and run a complete implementation based on a model containing capsules.

The ability to execute models has a revolutionary impact on the software development process. The results are higher quality software, and shorter and more predictable delivery cycles. Executing models is the surest way to find problems and issues that whiteboarding and document reviews do not find. Even high-level architectural models can be executed.

Use model execution to better understand the problem, to detect errors and problems in requirements and architecture specifications, to explore alternative designs quickly, and to test design models continuously during the development process.

Process note: To make the best use of Rose RealTime, you should aim to get your model running as often as possible. Making small, incremental changes and running your model each day will bring much better results than making widespread changes and working for weeks to get the model running again.

Constructing Models in Rational Rose RealTime

This section describes:

- Modeling Elements
- Diagrams
- Development Process
- Essential Workflows

Modeling Elements

There are many different modeling elements supported in Rose RealTime, and it is not easy for new users to know which elements to use to accomplish their goals. In practice, there are only a few elements that are required to construct a running model. The other elements provide greater flexibility and control over the expression of your design model.

As you have already seen from the overview description of the Rose RealTime services library, the services library is essentially a framework for executing capsules. Capsules are the main initiators and controllers of activity in a Rose RealTime model. You must define at least one capsule, but typically many more, in order to generate and compile code for the model. The capsules in the model should contain other classes, usually referred to as passive or data classes, because they must be invoked from a capsule behavior before they can perform any action. Also, they are primarily used by capsules to contain detail data, which is operated on within the capsule.

Required Elements

First of all, there are two elements that most developers will make use of at the start of a project to understand the problem domain and begin the process of turning the vague problem descriptions into detailed designs and implementations. These two elements are use cases and actors. They are used together (along with use case diagrams) for use case modeling, which helps architects, designers, testers, and others involved in the project understand the original system requirements and relate those requirements to elements in the design model. Use cases and actors are not strictly required to create an executable model, but use case modeling is highly recommended as an effective method in the overall analysis and design process.

Your model then must consist of capsule classes, and protocols, which specify the messages that capsules use to communicate. Almost all models also contain some data classes for capsules to use to store and operate on detailed data. Larger models contain many hundreds or thousands of capsule, protocol and data classes. These models should contain packages to organize the classes into related units.

A capsule must have a state machine defined for it in order to perform any useful behavior. The state machine defines the set of valid inputs that can be processed by the capsule. A complete code implementation is generated for capsule state machines, and any user-defined code to be performed as actions on state transitions is embedded in the generated code.

In any non-trivial model, some capsules will have structure defined for them that describes the interfaces which capsules use to communicate with each other. These interfaces are called ports. The structure also describes how capsules are contained by other capsules to construct composite systems. When one capsule is contained by another capsule, it is referred to as a capsule role.

Before you can compile a model you must create a component that describes which classes should be compiled as a unit, and the various settings that should be used to control the code generation, compilation and link processes.

Finally, in order to run your model, you must specify the deployment by adding a processor to the deployment diagram. The processor specifies the processing node (workstation or embedded target) on which your model will be executed. The compiled component must be mapped as a component instance on the processor so that the tool can load the compiled component onto the specified processor for execution and observation.

Further Reading

These are all the model elements that are required in order to create an executable model in Rose RealTime. Each of the possible elements in the model is described further in the *Modeling Language Guide*.

Diagrams

Diagrams are an essential part of the model. Diagrams describe how the different elements are combined together to make up the system. They also specify other forms of relationships among the model elements.

There are eight diagrams supported in the Rose RealTime tool, not all of which are required to create an executable model. Although not all diagrams are required, they exist for a purpose: the combination of these diagrams provides an excellent description of the total composition and behavior of the model.

The supported diagrams are:

- use case diagrams
- class diagrams
- state diagrams
- collaboration diagrams
- capsule structure diagrams
- sequence diagrams
- component diagrams
- deployment diagrams

Of these eight diagrams, only class diagrams, state diagrams, and capsule structure diagrams are essential in the development of an executable model.

The capsule structure diagrams describe the composition and connectivity of the capsules in the model. This is essential in the creation of anything more than the most trivial model. The generated code for the capsules reflects the information in the capsule structure diagram.

State diagrams must be created for each capsule that has any significant functionality. The tool will generate the implementation code for capsule state diagrams (no code is generated for state diagrams for other classes), and the capsule state diagrams provide the starting point for all behavior in the model. Class diagrams are used to define inheritance relationships between classes (capsule, protocol and data classes can all be subclassed). Class diagrams can also be used to show other associations among classes. The class diagrams may result in code being generated to implement the relationships defined in the diagrams, depending on the detailed settings for those relationships.

Component diagrams are used to specify the parameters for compilation of the model. In more complex systems, a hierarchy of components is compiled to make an executable.

Deployment diagrams must also be used to get a model running. The deployment diagram specifies how the model will be deployed on the destination hardware.

Further Reading

Each of these diagrams is explained in more detail in the *Modeling Language Guide*. The instructions for creating the diagrams are contained in the individual chapters for the diagrams in this guide.

Development Process

The Rose RealTime toolset is oriented to the use of an iterative, object-oriented development process. However, detailed description of the development process is beyond the scope of this document. We strongly recommend that you look at the Rational Unified Process (RUP) to gain a better understanding of the iterative object-oriented development process. See **http://www.rational.com**. At a more detailed level, the process of creating an executable model in Rose RealTime can be summarized as follows:

Use the use case modeling elements and use case diagram to develop a detailed, semi-formal understanding of the problem. The use case elements can be associated with design elements as the design model evolves to maintain traceability.

Create capsules, protocols, classes, use class diagrams, capsule structure diagrams, and capsule state diagrams to develop the structure and behavior of the model. Add detailed implementation code to the capsule state diagrams and to class operations.

In addition, use collaboration diagrams and sequence diagrams to capture the intended behavior of the system for various use cases. Use Rose RealTime's execution and debugging tools to validate the model behavior at run-time. Use collaboration and sequence diagrams to help you in the design process by making the communication patterns in the design evident. They will also help others understand your design.

Once the design has stabilized, use state diagrams for classes and protocols to capture the abstract design so that others can understand. This is particularly important for protocols, where the state machine specifies how a capsule using that protocol must behave.

Use the component diagram to specify the configuration of the model for compilation purposes.

Use the deployment diagram to indicate how the components should be executed. Also, the deployment diagram can be used to document the physical structure of the target system.

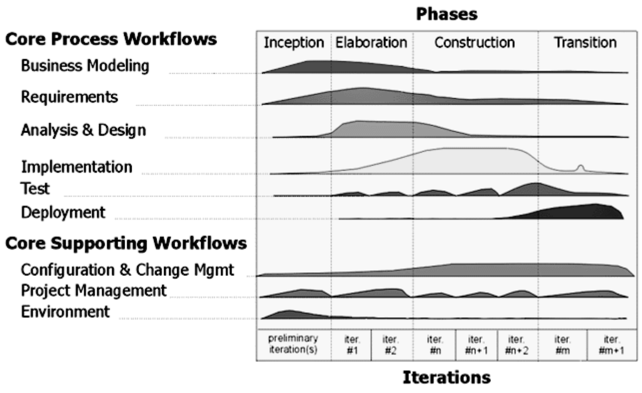
Further Reading

An overview of the Rational Unified Process is available in the Online Help book: Development process reference. The complete Rational Unified Process description is available at http://www.rational.com.

Essential Workflows

The following chart describes the workflows and project phases defined in the Rational Unified Process.

Figure 2 Workflows in the Rational Unified Process



Rose RealTime is not applicable to all of these workflows. The following workflows are important in the context of Rose RealTime:

Requirements - are typically captured in text documents or databases outside of the Rose RealTime toolset. Some analysis of the requirements is performed to develop a more abstract model of the problem. The abstract model is called a use case model. The Rose RealTime use case modeling tools are used in this worklow. In addition, design model elements can be traced back to requirements in two ways: through associations between use case model elements and design model elements captured in class diagrams, and through linking external files (requirements specifications, for example) to model elements.

Analysis & Design - is the primary workflow supported in the Rose RealTime toolset. All of the Rose RealTime class and capsule modeling tools are used in the analysis and design. There is no clear distinction between analysis and design in the Rose RealTime toolset. They are part of the same process, which is the process of turning vague problem descriptions into specifications of software-based solutions. The end goal of this process is a design model, which in Rose RealTime is complete enough to be executable. Execution of the design model is used as the basis for verifying whether the design meets the requirements. Intermediate artifacts such as design documents can be produced from the model.

Implementation - in a traditional development process without Rose RealTime, there is typically a gulf between analysis & design and implementation. In implementation, developers take the design specifications and produce code to implement those specifications. The mapping is not always straightforward, and the design specifications may be vague, incomplete or erroneous, leading to confusion, lost productivity and time delays. With Rose RealTime, the implementation is automatically produced directly from the model. Implementation details are still necessary, but they are added directly within the model framework. Going back-and-forth is not required to keep the model in sync with the implementation. It is always in sync.

Test - testing in Rose RealTime involves compiling a model and running it. A number of tools are provided in the run-time interface to assist with testing.

Configuration and Change Management - this workflow is ongoing, and is essential for orderly development in a large team environment. Configuration and change management involves putting the model and model elements under source control. See the *Guide to Team Development*.

Further Reading

Each of these workflows and the impact of the tools in Rose RealTime on those workflows is described in the Online Help.

User Interface Overview 2

Contents

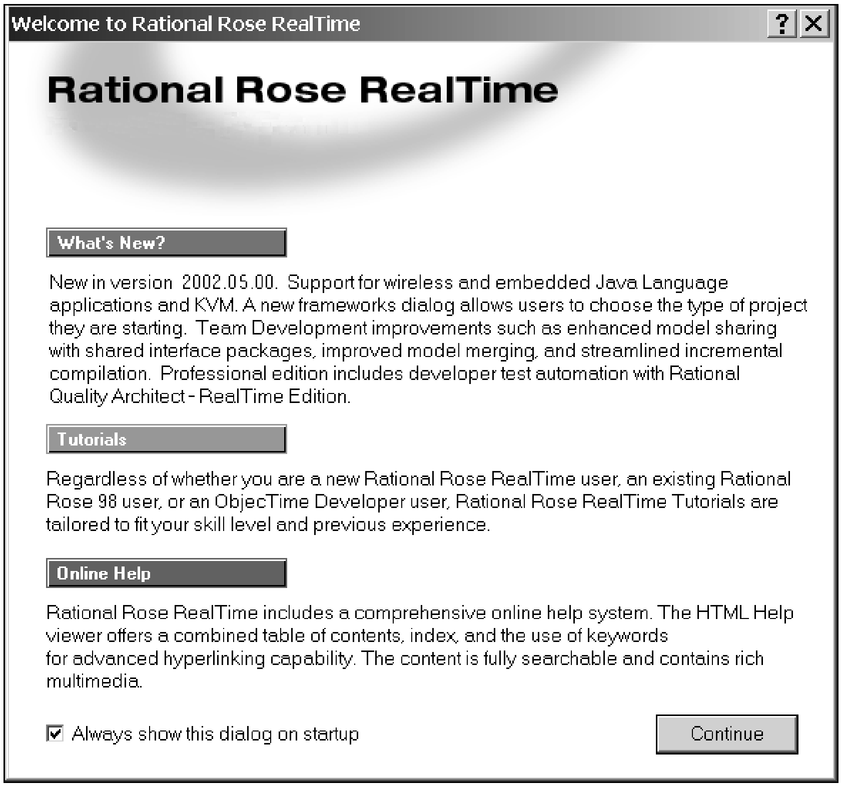
This chapter is organized as follows:

Startup Screen

The Startup screen (see *Startup screen* on page 30) has direct links to

- What's New? - to review out the new features of this release of Rose RealTime
- Tutorials - to review various tutorials tailored to your skill level and backgrounds
- Online Help - to access the Rational Rose RealTime help system

Figure 3 Startup screen


Welcome to Rational Rose RealTime

Rational Rose RealTime

New in version 2002.05.00. Support for wireless and embedded Java Language applications and KVM. A new frameworks dialog allows users to choose the type of project they are starting. Team Development improvements such as enhanced model sharing with shared interface packages, improved model merging, and streamlined incremental compilation. Professional edition includes developer test automation with Rational Quality Architect - RealTime Edition.

Regardless of whether you are a new Rational Rose RealTime user, an existing Rational Rose 98 user, or an ObjecTime Developer user, Rational Rose RealTime Tutorials are tailored to fit your skill level and previous experience.

Rational Rose RealTime includes a comprehensive online help system. The HTML Help viewer offers a combined table of contents, index, and the use of keywords for advanced hyperlinking capability. The content is fully searchable and contains rich multimedia.

☑ Always show this dialog on startup

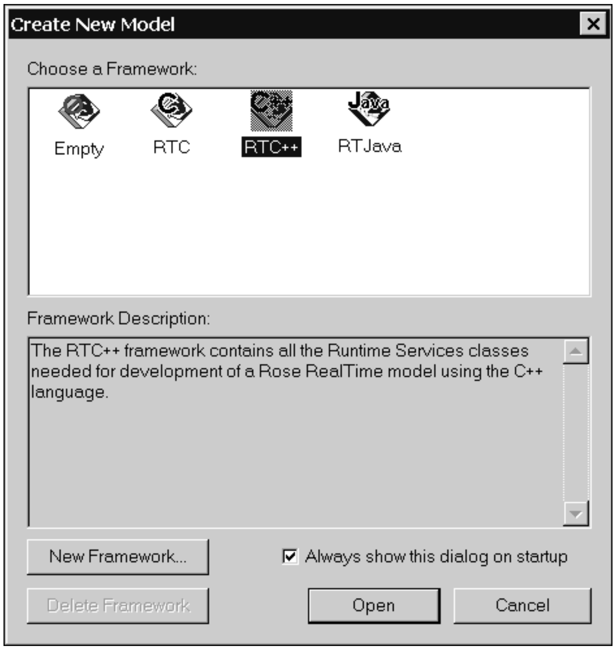
You have the option of always showing this screen on startup or bypassing it.

Create New Model Dialog

When you start Rose RealTime, the **Create New Model** dialog appears, with the following frameworks:

- Empty - enables you to open a model without any shared package, which is useful for pure modeling and /or use case development
- RTC - enables you to create a model in the C language
- RTC++ - enables you to create a model in the C++ language
- RTJava - enables you to create a model in the Java language

Additionally, any framework model that you create to be used as a template appears in the dialog. For information on creating a framework, see *Creating a Custom Framework for Rose RealTime Models* on page 377.



The RTC++ framework contains all the Runtime Services classes needed for development of a Rose RealTime model using the C++ language.

[✓] Always show this dialog on startup

Note: The **Empty** framework is useful for creating use case designs but should not be used for developing real-time applications.

If you want to open Rose RealTime without the **Create New Model** dialog box automatically appearing, select **File - New - Create New Model**, then deselect **Always show this dialog on startup**.

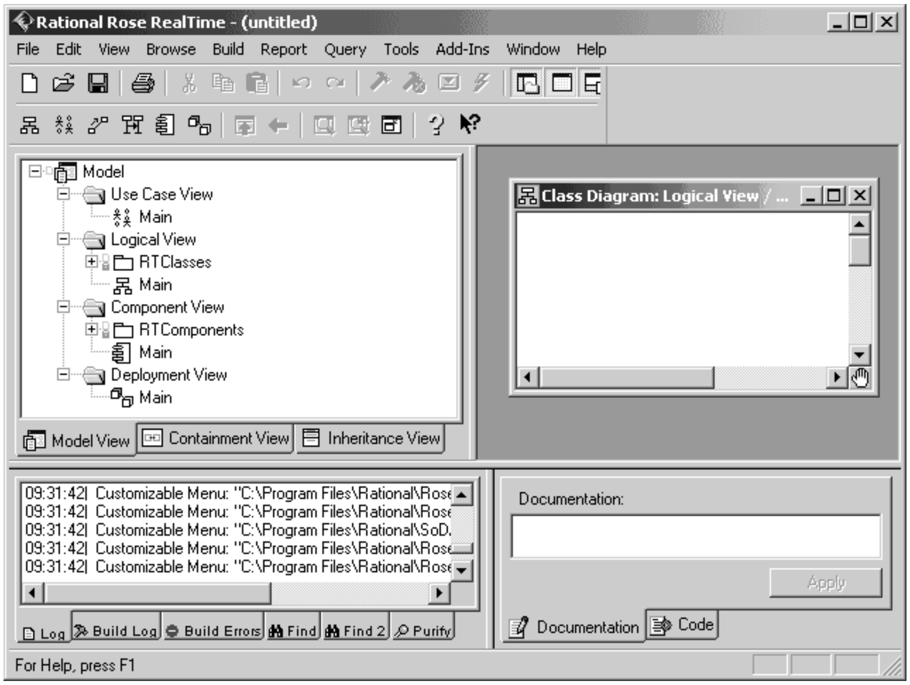
Application Window

The main elements of the Rose RealTime user interface are:

- The toolbar
- Menus
- Browsers
- Diagram Editors
- Specification Dialogs

Figure 4 shows an RTC++ example of the application window as it appears when the application is first started (with no model loaded).

Figure 4 Application window



```
09:31:42|  Customizable Menu:  "C:\Program Files\Rational\Rose
09:31:42|  Customizable Menu:  "C:\Program Files\Rational\Rose
09:31:42|  Customizable Menu:  "C:\Program Files\Rational\SoD.
09:31:42|  Customizable Menu:  "C:\Program Files\Rational\Rose
09:31:42|  Customizable Menu:  "C:\Program Files\Rational\Rose
```

Browsers

Model elements are created and viewed through browsers. The primary browser is the Model browser, that provides access to all elements of the current model. Browsers list the model elements - usually in a hierarchical way - allowing elements to be expanded to show additional information. Also, most browser lists can be filtered in various ways.

The containment view shows the containment hierarchy of the capsule classes in the model.

The inheritance view shows the inheritance hierarchy of the capsule classes, data classes, and protocols in the model.

Toolbar

There is a standard toolbar (see *The toolbar* on page 34) in the main application window that contains icons for a standard set of tools, which can be invoked at any time. This toolbar can be undocked and moved as a separate window.

Diagrams

Much of the model information is captured graphically in Diagram Editors. Additional non-graphical information, such as detailed program code, can be entered through Specification Dialogs. In many cases, information can be entered through a diagram or specification dialog, or Code window.

Each diagram or specification that you open is displayed in a window within the application window. These diagram and specification windows can be iconified. If the active window displays a diagram, that diagram is referred to as the current diagram. If the active window displays a specification, that specification is referred to as the current specification.

Toolboxes

Every diagram has an associated toolbox (see *Toolboxes* on page 77), that contains icons of tools that can be applied to that diagram. If the current diagram is write-protected, the diagram toolbox is not displayed. The diagram toolbox is dimmed when a diagram is displayed.

Menu bar

The menu bar lists commands available for operations in any diagram or specification window. Depending on the kind of diagram or specification displayed in the active window, some menu commands may not apply. These commands are dimmed. If the current diagram is write-protected, additional commands are rendered inaccessible.

About Rose RealTime Dialog

The About Rose RealTime dialog shows product version information, support contacts, and lists the add-ins.

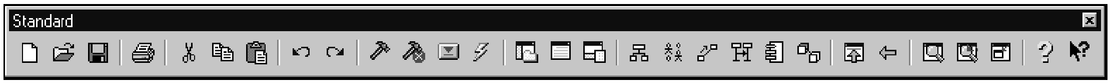
To open the **About** dialog select:

Help > About...

The toolbar

The standard toolbar is displayed directly below the menu bar along the top of the application window. The visibility of the toolbar is set through the **View** menu. By default, the standard toolbar is displayed. If you want to disable the standard toolbar from the application window, select **View > Toolbars > Standard**. This switches the visibility property of the toolbar.

Figure 5 Rose RealTime standard toolbar



Create New Model

Opens the **Create New Model** dialog. There are four frameworks listed: **Empty**, **RTC**, **RTC++**, and **RTJava**. Additionally, any framework model that you create to be used as a template, appears in the dialog.

To create a new Rose RealTime model containing all the classes required for development for C, C++, or the Java language, click the framework for the specified language. The **Model** browser appears with the packages and classes populated in the **Logical View** and **Component View**.

If you have a model open when you select the create model, you are asked if you want to save the current model. Selecting **No** discards all changes since your last save. Selecting **Yes** saves your changes and opens a new model, or displays the Load Model dialog automatically.

For information on creating a framework, see *Creating a Custom Framework for Rose RealTime Models* on page 377.

Note: The **Empty** framework is useful for creating use case designs but should not be used for developing RealTime applications.

Open Existing Model

Opens the Load Model dialog. If you have a model open when you select the open model, you are asked if you want to save the current model. Selecting **No** discards all changes since your last save. Selecting **Yes** saves your changes and opens a new model. See *Opening Models* on page 135 for more information.

Save Model

Opens the Save Model to dialog. Enter a new filename. After the model is named and saved, selecting this button automatically saves your changes to the current model without displaying the dialog.

Print Diagram

Opens the Print Specification dialog, that allows you to specify how and where diagrams are printed. To change printer setup select **File > Print Setup**.

Cut

Removes icons or relationships from your model. An item (or items) must be selected to activate the icon. Cutting an element also cuts associated relationships. You can cut multiple-selected items.

Copy

Copies a component to a new location of the same model - or a new model - without affecting the original component.

Paste

Pastes a component, that has previously been cut or copied to the clipboard, to another location.

Undo

Undoes the last operation performed. Not all operations can be undone. If the Undo tool is dimmed, the last operation cannot be undone.

Redo

Redoes the last operation that was undone.

Build Component

Initiates a model verification, generates the source code for the component, and invokes the external compiler and linker to create an executable version of the component. Only the model elements that have changed will be generated and recompiled.

Stop Build

Stops a build in progress.

Load Process

Loads the components instances specified in the Build Settings dialog. The component must be successfully built before it can run.

If the Attach Target observability flag was set on the Component Instance Specification dialog and a Target observability Port number filled in, the execution interface is displayed allowing you to control the execution of the model.

Run Component Instance

Loads the component instances specified in the Build Settings Dialog. The component must be successfully built before it can run.

View Browser(s)

Displays or hides all existing browsers. Existing browsers are those that have been created, but not closed.

View Description

Displays the Description Window, that contains the Documentation Pane and the Code Pane.

View Output

Displays the Output window.

Browse Class Diagram

Opens a class diagram in the Class diagram editor (see *Using the Class Diagram Editor* on page 152). Selecting this command opens a dialog allowing you to select from available class diagrams to open.

Browse Use Case Diagram

Opens a Use Case diagram in the Use Case diagram editor (see *Using the Use Case Diagram Editor* on page 144). Selecting this command opens a dialog allowing you to select from available use case diagrams to open.

Browse Collaboration Diagram

Opens a collaboration diagram using either the Collaboration diagram editor (see *Using the Collaboration Diagram Editor* on page 191) or the capsule structure Editor (see *Using the Structure Editor* on page 180). Selecting this command opens a dialog allowing you to select from available collaboration diagrams to open.

Browse Sequence Diagram

Opens a sequence diagram in the sequence diagram editor (see *Using the Sequence Diagram Editor* on page 216). Selecting this command opens a dialog allowing you to select from available sequence diagrams to open.

Browse Component Diagram

Opens the component diagram in the Component diagram editor (see *Using the Component Diagram Editor* on page 281).

Browse Deployment Diagram

Opens the deployment diagram in the Deployment diagram editor (see *Using the Deployment Diagram Editor* on page 284).

Browse Parent

Displays the "parent" of the selected diagram or specification. If you have a specification selected, the specification for the parent of the “named” item is displayed.

Browse Previous Diagram

Displays the last displayed diagram. To go more than one diagram back, you can click the small down arrow next to the button, and then click a diagram on the list.

Fit in Window

Centers and displays any diagram within the limits of the window. This command changes the zoom factor so that the entire diagram displays.

This command does not change the state of the diagram. Changes made after clicking **Fit In Window** may require that you re-click **Fit In Window** to center and resize the diagram again.

Undo Fit in Window

Reverts the diagram and window sizing back to its appearance prior to the **Fit in Window** operation.

Scale to Fit

Scales the diagram to fit within the current diagram window geometry.

Help Contents

Activates the online help system.

Context Sensitive Help

Activates the online help system and opens the help about that particular topic.

Menus

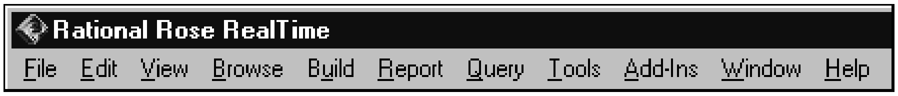
This section provides information on the Rose RealTime menus.

Menu bar

The menu bar provides drop-down menus for all operations on models and model elements.

Some menu items are context-sensitive, and only operate when certain types of model elements are selected. Context-sensitive menu items are grayed-out when they are not applicable.

Figure 6 Main menu bar


Rational Rose RealTime
File Edit View Browse Build Report Query Tools Add-Ins Window Help

The menu bar contains the following menus:

- File menu
- Edit menu
- Parts menu
- View menu
- Browse menu
- Build menu
- Report menu
- Query menu
- Tools menu
- Add-Ins menu
- Window menu
- Help menu

Not all menus are displayed at all times. Some of these menus appear only in context of particular diagrams. The **Parts** menu, for example, appears only when a capsule structure diagram or state diagram is open.

File menu

The operations available on the **File** menu may vary according to the current active window or the type of element selected.

File menu operations

New

Opens the **Create New Model** dialog. There are four frameworks listed: **Empty**, **RTC**, **RTC++**, and **RTJava**. Additionally, any framework model that you create to be used as a template, appears in the dialog.

To create a new Rose RealTime model containing all the classes required for development for the C, C++, or Java language, click the framework for the specified language. The **Model** browser appears with the packages and classes populated in the **Logical View** and **Component View**.

For information on creating a framework, see *Creating a Custom Framework for Rose RealTime Models* on page 377.

Note: The **Empty** framework is useful for creating use case designs but should not be used for developing RealTime applications.

A new model is unnamed until it is saved by a **Save** or **Save As** command.

Any open models are closed before a new model is created. You are prompted to save changes if necessary.

By default, a new model contains one empty class diagram - the main class diagram for the top level of the new model. You should place packages and classes representing your highest-level abstractions in this diagram. The new model is automatically created as a controlled unit.

Open

Loads a model or model kernel from a model file. A file browser is opened to let you to select a .rtmdl file. If there is an accompanying workspace (.rtwks) file, the tool asks if you want to open it instead.

Any open models are closed before opening another model. You are prompted to save changes if necessary.

If the selected model file's access control in the platform file system is read-only, the application write-protects the associated model.

When opening the model, the tools checks whether the model's saved character set is the same as the current system default charSet. If not, a dialog displays the following warning:

"Non system default character set in file."

See *Opening Models* on page 135.

Open Workspace...

Opens an existing workspace. A file browser is opened allowing you to select a .rtwks file.

Save Workspace

Saves the workspace. This action creates four files: a .rtwks file containing configuration management settings; a .rtmdl file containing the representation of the model itself; a .rtto file containing Target observability items, including probes and inject messages; and a .rtusr file containing various application settings.

Save Workspace As...

Saves the workspace. This action creates four files: a .rtwks file containing configuration management settings; a .rtmdl file containing the representation of the model itself; a .rtto file containing Target observability items, including probes and inject messages; and a .rtusr file containing various application settings. A file browser is presented to specify the file name and location.

Save Model

Saves the model. Writes the model out as a .rtmdl file. If the model has not been saved before, a file browser is presented to specify the file name and location.

If a destination model file's access control in the platform file system is read-only, an ERROR! dialog is displayed indicating the software cannot write that file.

To control temporary and backup files created during a Save procedure, refer to the Customizing the Diagram Toolbox in the **Tools > Options** dialog.

Save Model As...

Saves the model. Writes the model out as a .rtmdl file. A file browser is displayed to specify the file name and location. This command displays the **Save Model To** dialog, in which you can specify the new file name and the location where you want to save the current model.

Import...

Imports a model file created by another tool. See *Importing a File* on page 289 for more information.

This command displays the Read Petal dialog so you can specify the petal file you want to import. Use this command to import the contents of a petal file into the current model. This command requires the active window to contain a class or component diagram.

When you import a petal file that contains elements, the diagram in your active window is used to select a destination. If the active window is the top level diagram, these elements are imported into the top level of the current model. Otherwise, these elements are imported into the package that encloses the diagram in the active window.

Each imported element is compared to the corresponding element in the current model. If any of the elements in the petal file already exist in the current model, error messages are sent to the log. If an imported package contains elements that already exist in the current model, a dialog is displayed telling you that these elements have not been imported. All diagrams in the model are appropriately updated, including those imported from the petal file.

When you import a petal file that contains a complete model, that model is opened. If a model is already open with unsaved changes, the Save Confirmation dialog is displayed, prompting you to save your changes before closing the current model and opening the model contained in the petal file.

Export Model...

Exports model files in alternative formats. Primarily used to exchange models with other versions of Rational Rose and other Rational Rose tools.

This command displays the Write Petal dialog so that you can specify the name and location of the petal file. Use this command to export selected items from the current model to a petal file. The Export command displays the name of the element type selected. If nothing is selected, Export Model is displayed. You can export

- the entire model
- Classes
- Logical Packages
- Component Packages

Begin by displaying diagrams containing the items you want to export. Select the specific classes, logical and component packages to be exported, and pull down the **File** menu. The Export command indicates the number of items selected. If no items are selected, the entire model is exported. If any item not on the above list (such as a relationship or adornment) is selected, the Export command is not executable.

When a logical or component package is exported, all diagrams it contains are also exported. When individual classes are exported, however, only their state transition diagrams are exported with them. Exporting the entire model does export all diagrams contained within it.

Exporting to a petal file is useful when you want to transfer

- elements from one model to another
- a model or its elements between different computing platforms
- a model or its elements to a new software release

Print...

Prints the model. *Printing* on page 123

Print Setup...

Changes the print setup before printing.

Edit Path Map

Edits or creates pathmap variables. Opens the Virtual Path Map dialog. Using the dialog, you can create an entry to represent a mapping between a virtual path symbol and an actual pathname. This feature allows you to work with models moved or copied among workspaces and archives by redefining the actual directory associated with the user-defined symbol.

Recent Files

Opens recently edited models.

Recent Workspaces

Opens recently used workspaces.

Exit

Exits the application.

Edit menu

Undo

Undoes the last operation. Some operations may not be undone. If Undo is not possible, the **Undo** menu item is grayed-out.

Redo

Redoes the last undone operation.

Cut

Removes the selected item and places it in the buffer. When you cut an item, all relationships for that item are also cut. For example, if A is a generalization of B and you cut A, the generalization is also cut.

This command works only on the graphic representation of a diagram. It does not change the current model. Not all elements can be cut and pasted. If after selecting an element or group of elements the **Cut** menu item is grayed-out, the cutting and pasting of one or more of those elements is not supported; for example, you cannot cut and paste multiple elements in a capsule state diagram.

Copy

Copies the selected item into the buffer. Use this command to copy the currently selected item or items to the clipboard. From the clipboard, you can

- paste items into other diagrams
- paste items into documents you create with any standard word-processing software

The Copy command provides a simple means of importing a class from one package to another.

If a relationship is copied and your selection does not include the items at both ends of that relationship, you cannot paste that relationship. In this circumstance a Warning dialog appears, allowing you to cancel or continue. You can also disable subsequent warnings for the remainder of your session.

This command works only on the graphic representation of a diagram. It does not change the current model.

Paste

Pastes whatever is currently in the buffer into the selected destination (the active window).

Delete

Deletes the currently selected item(s) from a diagram or specification.

When used in a diagram, the Delete command removes each selected icon from the current diagram. The model is not changed unless the deleted icon is unnamed.

In all but the behavior diagrams of a capsule, delete never deletes the model object. In the structure and behavior diagrams of a capsule delete always deletes the model object.

When you delete an item, all relationships associated with that item are also deleted.

In collaboration and interaction diagrams, you are prevented from deleting an icon representing a component or relationship if there are no other icons representing that component or relationship in the active diagram. In a collaboration diagram, you are prevented from deleting an icon representing an object if that object has one or more links, and you are prevented from deleting a link if that link has one or more messages. In all of these cases, you can use the Delete from Model command.

Duplicate

Duplicates a selected model item into the package that owns the diagram and validates its name in the context.

Select All

Selects all elements on the current diagram or all text in the current editor.

This command is useful when you want to:

- generate reports on all the classes in a single diagram using commands on the **Report** menu
- populate a class diagram with all of the information about those classes using commands on the **Query** menu
- change the font size or characteristics for all of the text in a diagram using commands on the **Options** menu

To deselect all items, click at a point on the diagram that is not already highlighted.

Note: The following two items are only available from Sequence diagrams.

Attach Text Label

Attaches Text Label to a graphic element.

Detach Text Label

Detaches Text Label from a graphic element.

Delete from Model

Deletes the selected item(s) from the model.

This command can be used from diagrams to delete items from the current model. When you delete an item from the model, all icons representing that item are removed from any diagrams in which they appear. The specification for the item is also deleted.

This command cannot be used in specifications. To delete an item from the model via a specification, use the Delete command.

Relocate

As the analysis and design of an application proceeds, it is common to refine the application's logical and/or physical architecture from one iteration to the next. Such refinements can include:

- relocating a class or logical package from one logical package to another
- relocating a component or component package from one component package to another

The Relocate command supports these refinements. It allows you to relocate a model element (class, component, package, or association) to a new logical or component package.

The component will now be contained by the component package containing the current diagram. However, relocating a component or component package has no effect on any diagram in the model.

Diagram Object Properties

Lets you customize various software features. The characteristics you set via this menu item affect only the selected icon(s).

Line Attributes...

Lets you select between rectilinear or oblique line styles, and routing styles. You can undo and redo any changes you make. Select **Line Attributes > Edit...** to edit the properties of line attributes.

Find...

Use this command to find any item in the model. This command displays the Find dialog so that you can type the search string.

The dialog provides a drop-down list of previous search strings. Select an existing search string or type the name of the item to find. The search result is displayed in the Find tab of the Output window (**View > Output**) in a three-column list, including the name, type, and location. Optionally, you can choose to have the result displayed in the Find 2 tab.

To search for groups of items or to search code, you can use the * wildcard character:

- A* matches any name beginning with the letter A
- *A matches any name ending with the letter A
- *A* matches any name containing the letter A

This command lists only the first 250 items that match a name containing a wildcard character. To search for "*", use "*".

The * wildcard is especially useful for finding any classes or packages that were automatically renamed in a model that was upgraded from a previous release. For example, you can search for every model item that was renamed by typing: *#* (star pound star). Every model item that has a # in its name is found.

This command searches for the named item and displays a list of diagrams in which that item appears. You can double-click on an entry to display that diagram.

Replace...

Use this command to find and replace any item in the model. This command displays the Replace dialog so that you can type the search string and the replace with string.

The dialog provides drop-down lists of previous search and replace strings. Select an existing search or replace string or type the names of the item to find and replace. The search result is displayed in the Find tab of the Output window (**View > Output**) in a three-column list, including the name, type, and location. Optionally, you can choose to have the results displayed in the Find 2 tab.

When you click Replace, a secondary dialog appears with options t o Find Next, Replace, Replace All, and Cancel. After the replace operation has taken place, you can query the Find tab in the Output window to view the results of the search and replace.

Reassign...

Each icon in a diagram represents an element in the current model. Use this command to make a selected icon represent a model element other than the one it now represents. For example, you can assign an existing class to a different diagram element.

This feature is useful if you want to assign an element to use the same named item from a different name space.

To make an icon represent another element, select the icon and then click Reassign from the **Edit** menu. The dialog lists the packages in the model on the left and a list of the valid elements to choose from on the right. Choose the model element that the selected icon will represent. This affects only the selected icon. Other icons representing the original model element (on all diagrams) maintain their original representation.

Here is an example: Assume you have three classes named car, buggy and wagon. With buggy selected, click **Edit > Reassign**. In the dialog select wagon. You are changing the underlying model of the diagram element buggy to use wagon instead of buggy. (Buggy may still exist in the model but you are given the option of deleting it.) Additionally, if buggy had an inheritance relation drawn to car, then wagon adopts that inheritance relation.

The Reassign command does not work within Capsule Collaboration (Structure) diagrams.

Compartment...

Opens the Edits Compartments dialog. This dialog allows you to arrange how attributes and operations are displayed within a class or package icon on a diagram.

Change Into

Changes the selected model element into another (related) kind of modeling element.

In the process of refining your model, you may find it necessary to change a model element from one kind to another. For example, you may want to change a class into a capsule once you have decided that the class has a state machine and a logical thread of control.

You can use the commands on the **Change Into** submenu to change a model element from one type to another. You can transform

- a class into another type of class
- a relationship into a different type of relationship

You can also transform an element as follows:

1. Choose the icon on the diagram toolbox.
2. Press the `ALT` or `META` key.
3. Click on the element you want to change.

This command changes the model element and updates all diagrams containing this element.

When a relationship type is changed, this command removes the original relationship from all diagrams, but does not automatically add the new relationship to these diagrams. Use the Filter Relationships command to display the new relationship in specific diagrams.

Parts menu

Note: The **Parts** menu is only available on capsule structure and state diagrams.

Edit Inside

Shows the internal composition of a contained state or capsule role's class. Allows you to perform edits (with some limitations) on elements contained inside the selected state or capsule role without having to open a new editor. This is most useful for seeing the internal connections of transitions to substates and of relay ports to other contained capsule roles.

Remove/Exclude

Removes a local or exclude an inherited element from the current class. For example, if you are creating a subclass of an existing capsule class, and the superclass defines a state that is not applicable in the subclass, you may remove it in the subclass diagram using the Remove/Exclude command. Any excluded elements still appear in the navigator area of the structure or behavior editor, but have the symbol **x** beside them to indicate that they have been removed.

Inherit

Causes a previously excluded element to be reinherited. Reinherited elements are added back to the diagram.

Aggregate

Applies only to states or capsule roles. For states, the aggregate command creates a new composite state containing the selected states to be aggregated. For capsule roles, the aggregate command creates a new capsule class to hold the capsule roles that are being aggregated. This command also replaces the aggregated capsule roles in the structure diagram where the command was executed with a single capsule role of the newly-created capsule class.

Decompose

Applies only to composite states or capsule roles that are aggregates. Breaks the selected state into its immediate substates, or breaks the selected capsule role into the capsule roles contained within the aggregate capsule class.

Promote

Moves the selected element up in the class hierarchy. The element is moved into the immediate superclass and is inherited by the subclasses (including the current class). If there is any name conflict between this element and another element in the superclass or in any of its subclasses, the promote command fails.

Demote

Moves the selected element down in the class hierarchy. The element is removed from the current class, and is pushed down into all immediate subclasses, as if it had been defined locally on the subclasses.

Lock Position(s)

Locks the element in position on the diagram. Once the element is locked it cannot be moved around the diagram unless it is unlocked.

Unlock Position(s)

Allows the element to be moved around within the diagram.

View menu

Toolbars

Toggles the display of the The toolbar and Toolboxes.

Status Bar

Toggles the display of the status bar at the bottom of the window, which provides textual information about selected items and current operations.

Browsers

Toggles the display of all application browsers, as well as create a new one.

Description

Toggles the display of the Description Window, which contains the Documentation Pane and the Code Pane.

Output

Toggles the display of the output window.

Filter

Filters label information on diagrams.

Zoom

Zooms in on the current diagram. Select the zoom level.

Scale to Window

Scales the current diagram down to fit entirely within the current diagram window border. Scales according to the outer boundaries of the diagram - for example, the outer state border of the state diagram - and not simply the area around visible diagram elements.

Page Breaks

Toggles the visual indication of where page breaks appear on diagrams when printed. The printer specified in the Printer Setup determines the exact location of page breaks. You can also change this setting through the rose.ini file.

Refresh

Redraws the current diagram.

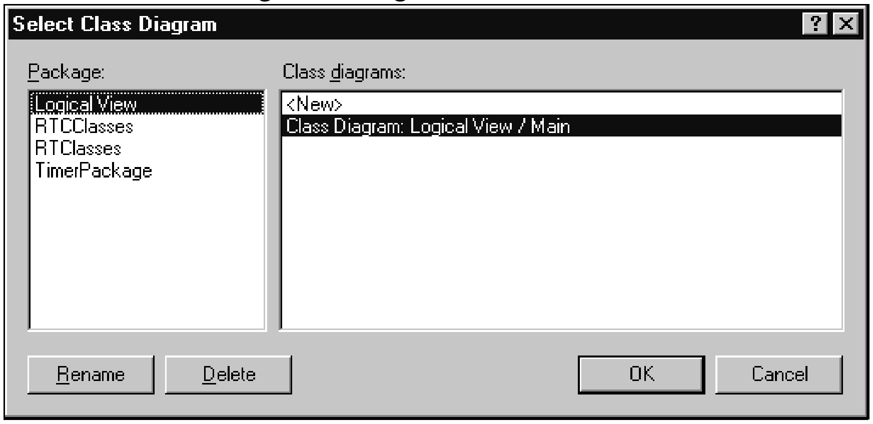
Browse menu

Use commands on the **Browse** menu to navigate through the diagrams and specifications that represent your model.

Select Diagram dialog

When you select a diagram type from the **Browse** menu, a dialog appears for that type of diagram. For instance, when you select **Browse > Class Diagram...** the Select Class Diagram dialog appears (see Figure 7).

Figure 7 Select Class Diagram dialog



Using commands from the dialog, you can display, rename, create, and delete diagrams.

Browse menu operations

Class Diagram...

Opens a Select Class Diagram dialog, allowing you to select a diagram to open, or to create a new diagram. The dialog also allows you to rename or delete diagrams.

Use Case Diagram...

Opens a Select Use Case Diagram dialog, allowing you to select a diagram to open, or to create a new diagram. The dialog also allows you to rename or delete diagrams.

Collaboration Diagram...

Opens a Select Collaboration Diagram dialog, allowing you to select a diagram to open, or to create a new diagram. The dialog also allows you to rename or delete diagrams.

Sequence Diagram...

Opens a Select Sequence Diagram dialog, allowing you to select a diagram to open, or to create a new diagram. The dialog also allows you to rename or delete diagrams.

Component Diagram...

Opens a Select Component Diagram dialog, allowing you to select a diagram to open, or to create a new diagram. The dialog also allows you to rename or delete diagrams.

Deployment Diagram...

Opens a Select Deployment Diagram dialog, allowing you to select a diagram to open, or to create a new diagram. The dialog also allows you to rename or delete diagrams.

State Diagram

This menu item is only activated when you have selected a capsule, class, or protocol on a diagram. Use this command to display the state diagrams associated with the selected class or protocol in a class diagram. You can also use this command from a class specification to display the state diagram for that class. This operation also opens a state diagram for the top-level of the selected capsule.

Structure Diagram

This menu item is only activated when you have selected a capsule on a diagram. This operation opens a structure diagram for the selected capsule.

Open Superclass

Opens the corresponding diagram on the immediate superclass. Only applies when the current diagram is a capsule collaboration diagram or a state diagram (capsule, protocol, or data class).

Show Subclasses

Displays a list of subclasses in a Choose Capsule Role Dialog. Only applies to capsules. Selecting a capsule from the displayed subclass list and clicking **OK** opens the corresponding diagram for the selected capsule. Only applies when the current diagram is a capsule structure diagram or a state diagram (capsule, protocol, or data class).

Go Inside

Replaces the current diagram window contents with the corresponding diagram for the selected capsule role (for capsule structure diagrams) or substate (for capsule state diagrams). For example, selecting a substate on a capsule state diagram and choosing **Browse > Go Inside** causes the substate's state diagram to replace the current state diagram in the same window. Only applies when the current diagram is a capsule structure diagram or a state diagram (capsule, protocol, or data class).

Go Outside

Replaces the current diagram window contents with the corresponding diagram for the selected capsule role (for capsule collaboration diagrams) or substate (for capsule state diagrams). For example, selecting a substate on a capsule state diagram and choosing **Browse > Go Outside** causes the substate's state diagram to replace the current state diagram in the same window. Only applies when the current diagram is a capsule structure diagram or a state diagram (capsule, protocol, or data class).

Expand

Opens the subdiagram associated with an item. Packages/subsystems have a default main diagram that you can expand to if you select a package or a subsystem in another diagram.

Parent

Selecting **Browse > Parent** or clicking the Browse Parent icon on the toolbar displays the “parent” of the selected diagram or specification. If you have a specification selected, the specification for the parent of the “named” item is displayed. For example, if you select a substate selected in a state diagram, choosing **Browse >Parent** opens a state diagram on the parent state from a substate.

Specification...

Opens the specification dialog for selected item(s) on the current diagram.

Top Level

Use this command to display:

- the top level main class diagram
- the top level main component diagram
- the deployment diagram for your model

Current Diagram	New Current Diagram
Class diagram	the main top level class diagram
State diagram	the main top level class diagram
Collaboration diagram	the main top level class diagram
Sequence diagram	the main top level class diagram
Component diagram	the main top level component diagram

Package specification	the main top level class diagram
Class specification	the main top level class diagram
Object specification	the main top level class diagram
Component package specification	the main top level component diagram
Component specification	the main top level component diagram
Process specification	the deployment diagram for your model
Device specification	the deployment diagram for your model
Connection specification	the deployment diagram for your model

Referenced Item

Displays a diagram or specification referenced by the selected item. In particular, you can

- display a diagram showing the class of which the selected object is an instance
- display the diagram where the class, use case, or package is actually defined
- display the operation specification for a message, provided that the message is tied to an operation; note that you must select the message label before executing this command

If the selected icon represents an object, this command finds the object's parent class and displays a diagram in which that class appears. If the class appears in multiple diagrams, the diagram in which the class was created is displayed.

If the selected icon represents a class that was created in a different logical package, this command displays a diagram from the logical package in which the class appears. If the class appears in multiple diagrams from that logical package, the diagram in which the class was created is displayed.

Previous Diagram

Brings to front or opens the last diagram that was current.

Build menu

Build

Opens the Build dialog from which you can choose the Build Level.

Quick Build

Builds the component incrementally.

Rebuild

Forces a complete build of a component. All classes references by the component will be verified, regenerated, compiled, and linked.

Clean

Removes all files from the output directory.

Stop Build

Stops the build in progress.

Run (F5)

Loads the component instances specified in the Build Settings Dialog. The component must be successfully built before it can run.

If the Attach Target observability flag was set on the Component Instance Specification dialog and a Target observability Port number filled in, the Target observability interface is displayed allowing you to control the execution of the model.

Note: The following four items only apply when a Target Observability session is running.

Start (F5)

Starts the execution of the component instances. If the component instances are in the reset state, execution begins with all fixed capsules being initialized (initial transitions fired). If the component instances are in the stop state, execution resumes.

Stop (Shift+F5)

Stops the execution of the component instances at the current point of execution and remembers the state of all capsules. Execution is stopped as soon as each currently running transition is finished. The stop button does not halt execution in the middle of a transition action.

Step (F10)

Steps through the next deliverable message. Pressing the step button while in the stopped state causes the next message of the highest available priority to be delivered. Any associated transitions are executed. Execution stops again as soon as the last transition segment for that message has finished executing.

Restart (Ctrl+Shift+F5)

Resets the component instances, resetting all fixed and destroying all dynamic capsule instances. The running component instance is terminated and a new one is run.

Load

Loads the components instances specified in the Build Settings dialog. The component must be successfully built before it can run. The Load command spawns an external process in which the model executable runs. You will likely see an external command window appear.

The Attach Target observability flag must be set on the Component Instance Specification dialog, and a Target Observability Port number filled in for the model to be loaded within the tool.

The execution interface will be displayed allowing you to control the execution of the model. See Execution basics for more information on the execution tools.

Reload

Kills the existing model process and runs the model again. The execution interface stays open.

Shutdown

Kills the existing model process and closes the execution interface.

Settings...

Displays the Build Settings Dialog. You must use this dialog to specify the active component before you can build the component.

Add Class Dependencies...

Runs a script that checks for any missing dependencies between model elements and adds them. The script checks dependencies found in attributes or operations. It does not check for code-level dependencies.

Component Wizard...

Activates the **Component Wizard** to help you through the steps of creating and deploying a component.

Report menu

Generates lists of diagrams in which the selected class is a supplier in a relationship, or in which instances of the selected class appear. These lists can be used to navigate to the diagrams they contain.

Show Usage...

Obtains a list of all the locations where the selected item is used (a supplier in a relationship).

This command displays a list of diagrams in the Show Usage dialog. Double-click on a diagram from the list to display the diagram.

Show Access Violations...

Obtains a list of access violations in the model. An access violation occurs when an element in one package references an element in another package that is not visible to it.

The rules for determining if an element B in package P2 is visible to an element A in package P1 are as follows:

- P1 and P2 are the same package OR
- B has its visibility set to Public AND
- there is a dependency from P1 to P2 OR
- there is a dependency from P1 to a package that contains P2 OR
- there is a dependency from a package that contains P1 to P2 OR
- there is a dependency from a package that contains P1 to a package that contains P2 OR
- P2 is marked as global (in the Detail tab of the Specification dialog)

The **Show Access Violations** menu item is available for class diagrams and component diagrams.

To check for access violations in the Logical View:

1. Open a class diagram.
2. With nothing selected in the diagram, choose the **Show Access Violations** menu item.
3. If there are any access violations, they are listed in a dialog. You can open an editor that shows the cause of a violation by selecting it in the dialog and clicking Browse (or by double-clicking on the violation). The list of violations can be sorted by clicking on either the Violator or the Supplier column headings.

To check for access violations in a specific set of classes, select those classes on the class diagram before choosing the **Show Access Violations** menu item. Selecting a package on a class diagram is equivalent to selecting each class in that package.

To check for access violations in the Component View:

1. Open a component diagram.
2. With nothing selected in the diagram, choose the **Show Access Violations** menu item.
3. If there are any access violations, they are listed in a dialog.

To check for access violations in a specific set of components, select those components on the component diagram before choosing the **Show Access Violations** menu item. Selecting a package on a component diagram is equivalent to selecting each component in that package.

The access violations calculation examines the existing class or component relationships in the model. For this reason you should ensure that the relationships are complete by building the model.

Show Code Occurrences...

Shows code occurrences in the Find tab of the Output window.

Show References...

Opens a dialog listing all the references to the selected model element from other model elements (either in non-documentation properties fields or in diagrams). Does not show detailed code references (use Find).

Documentation Report

Generates a data dictionary from the model.

Show Part Of Ancestors

This option is only available when your current diagram is a structure diagram. Opens a dialog listing all the capsules that contain this capsule as a capsule role.

Show Part Of Descendants

This option is only available when your current diagram is a structure diagram. Opens a dialog listing all the contained capsule roles of this capsule.

Query menu

The **Query** menu provides commands that control which model elements appear in the current diagram. Use case diagrams, class diagrams and component diagrams support the **Query** menu functionality.

Add <element> commands

Some menu items are only available when certain diagrams are active.

Use these commands to populate the current diagram with icons representing one or more of the selected elements from the model. You can use this command to populate a new (empty diagram) or to add elements to an existing diagram. In either case, you must create or display the diagram first.

If relationships exist among the elements you are adding, or if relationships exist between added elements and any elements already appearing in the diagram, icons representing these relationships and their adornments will also appear in the diagram. Use the Filter Relationships command to directly control which kinds of relationships appear in the diagram.

Add Classes...

Adds classes from the browser to the current diagram. Brings up a dialog with a list of available classes to choose from. This menu item is only visible when a Use Case or Class Diagram is open.

Add Capsules...

Adds capsule classes from the browser to the current diagram. Brings up a dialog with a list of available capsule classes to choose from. This menu item is only visible when a Use Case or Class Diagram is open.

Add Protocols...

Adds protocol classes from the browser to the current diagram. Brings up a dialog with a list of available protocol classes to choose from. This menu item is only visible when a Use Case or Class Diagram is open.

Add Use Cases...

Adds use cases from the browser to the current diagram. Brings up a dialog with a list of available use cases to choose from. This menu item is only visible when a Use Case or Class Diagram is open.

Add Components...

Adds components from the browser to the current diagram. Brings up a dialog with a list of available components to choose from. This menu item is only visible when a Component Diagram is open.

Add Interfaces...

Adds classes from the browser to the current diagram. Brings up a dialog with a list of available classes to choose from. This menu item is only visible when a Component Diagram is open.

Expand Selected Elements...

Allows you to specify relationship level and client/supplier criteria for choosing additional elements. All settings in the dialog are remembered from the previous use.

Use this command to show additional model elements in the current diagram. This command enables you to add icons to the current diagram elements having a specified relationship with a selected element or set of elements. For example:

- All classes that inherit from a selected class
- The classes from which a selected class inherits
- The classes that a selected class associates to
- The classes that directly use a set of selected classes
- All components that a component uses
- All packages that a component uses
- All interfaces that a component uses
- All components that a package uses
- All packages that a package uses
- All interfaces that a package uses
- All interfaces that a component realizes

From the Expand Selected Elements dialog, you can display the *Class Specification - Relations tab* on page 243 for class diagrams or for component diagrams to specify relationship-kind and access-kind criteria for choosing additional elements.

Note: The level cannot be changed if you select Expand indefinitely.

Hide Selected Elements...

Specifies the elements whose icons are to be removed from the current diagram.

Use this command to remove icons representing components from the current diagram. The components represented by these icons are not deleted from the model.

By default, this command removes only the components whose icons are selected. You can optionally remove icons representing components that are clients or suppliers of the selected components.

Filter Relationships

Displays the Relations tab from the class and use case diagrams, and the Visibility Relations dialog from the component diagram. Both enable you to specify which kinds of relationships can appear. The filter relationship dialog remembers the last set of filter settings used. Use this command to control which kinds of relationships appear in the current diagram.

Tools menu

Layout

Opens a submenu of options for rearranging the diagram:

Layout Diagram

Analyzes the location of all icons in the current diagram, determines the optimal location for the icons, and redraws the diagram.

Align/Distribute...

Opens the Align and Distribute dialog. The selected objects are arranged according to the choices made in the dialog. Alignment and distribution operations can be performed in both horizontal and vertical arrangements.

Change View Spread

Creates space in a diagram for adding new views or creates a cleaner appearance. There are a number of different ways the views can be spread out by specifying the Spread Technique.

- Uniform - indicates that the views are spread out across the diagram uniformly by the percentage. If a view is at location (100,100) and they specified a horizontal percentage of 10% and a vertical percentage of -10%, the new location of the view would be (110, 90). This affects all views in the diagram the same way.
- Constant Radial - indicates that the views spread outward/inward from a central point. The preview displays a crosshair that specifies where the spread starts. It can be moved around interactively in the preview window with the mouse. The views spread out a constant distance based on the diagram size.
- Decreasing Radial - is similar to Constant Radial except that the views spread progressively less far the farther away from the central point they are.
- Increasing Radial - is similar to the Constant Radial except that the views spread progressively more the farther away from the central point they are.

The preview allows the user to play with the settings until the desired spread is achieved.

Autosize All

Resizes all node views in the diagram to fit their labels.

Make Same Size

Makes two or more node views the same size in either height or width, or both. You can choose from smallest, average, or largest of all the selected views to make the new width and height. The dialog provides a preview screen.

Create

Opens a submenu of options for creating elements to place on the current diagram. Use the commands on the **Create** menu to place the icons in the active diagram.

When you choose an item from the **Create** menu, the corresponding diagram toolbox tool becomes active and the pointer changes to a cross (for a node) or an arrow (for a relationship). You can then use the mouse to position the pointer and place the new item.

The contents of the **Create** menu change to match the current diagram's toolbox.

Check Model

Provides a way of re-executing the model validation that happens at open time.

Check Model is designed to be used when you are saving your model to multiple controlled units to ensure that all the units are consistent with one another. This is especially useful when parallel development is going on in multiple controlled units, since it is possible for different units to get out of sync with one another.

In a model, where one item holds a reference to another item, it is possible that a reference exists, but there isn't an item in the model of the right kind or with the right name. In that instance, the reference is unresolved.

Check Model checks the reference:

- to the supplier of any kind of relationship, uses, instantiation, metaclass, logical package import, module visibility, connection, and so forth
- from a view on a diagram to an item in the model

Import Code...

Opens a file browser allowing an external code file to be selected and imported. See *Importing Rational Rose Generated Code* on page 141.

Model Properties

Use the commands on the **Model Properties** submenu to display or modify the model properties associated with the model and its elements, or to display or modify model property sets.

Edit

Opens the Options Dialog with the C++ Tab if nothing is selected. This tab is used to display or modify model property values or model property sets.

Replace

Loads model property sets from the specified model property (.pty or .rtpty) file into the current model. This command deletes all model property sets in the current model, replacing them with the imported model property sets. A model component attached to a model property set that is replaced becomes attached to the replacement model property set. A model element attached to a model property set which is not replaced becomes attached to its default model property set. To make the replaced model properties a permanent part of the current model, you must save the model.

Export

Saves the current model's model property sets to a specified model property file (.pty file). When you export model properties, all of the model property sets stored with the model are written to a file that can be imported into another model.

This command displays a dialog in which you can specify the location and name of the model property file to be exported.

Add

Adds new model properties from a model property set contained in a model property file.

Update

Modifies the existing model properties in the current model by adding and/or changing them to include the model properties in the update model property set. A model element attached to a model property set that is updated becomes attached to the updated model property set. A model element attached to a model property set that is not updated becomes attached to its default model property set.

This command opens a File Browser so you can specify the location and name of the model property file to be used to update the existing model properties.

To make the updated model properties a permanent part of the current model, you must choose the Save command from the **File** menu.

Options...

Opens the Options Dialog, which provides control over many general model properties. (See "Toolset Options" on page 384.)

Source Control

Opens a submenu of operations for interacting with a source control/configuration management (CM) system. For information on Source Control options, see Source Control Fundamentals in the *Guide to Team Development* for Rational Rose RealTime.

Configure...

Opens the Model Specification dialog on the Source Control tab, which allows you to specify options relating to source control, to specify the source control system to use, and to generate unique identifiers for all elements of the model.

Note: Generating unique identifiers affects the entire model. Review *Unique Ids* on page 131 **before** setting this option.

Get Entire Model

Requests a given version of all files from the CM tool, and then loads the new files.

Synchronize Entire Model

Synchronizes the status of the model elements with the current source control tool and reloads any files that have been changed outside of the toolset.

Refresh Status of Model

Synchronizes the status of model elements displayed in the model browser with the status as reported by the CM tool.

Select Checked out Units in Browser

Selects all the units in the browser that are currently checked out.

Show Unit Versions

Shows a dialog containing a list box, which displays the version of each unit.

Submit All Changes to Source Control

Determines what changes have not yet been submitted to source control, then prompts you to add/checkin these changes as appropriate.

Synchronize Model with File System...

Throws away any unsaved edits and reloads all files from the file system.

Open Script

Opens a file browser to select a Rose REI or RRTEI script to open for editing. See The Script Editor Window in the *Rational Rose RealTime Extensibility Interface Reference.*

New script

Opens the script editor to create a new Rose REI or RRTEI script. For more information, see The Script Editor Window in the *Rational Rose RealTime Extensibility Interface Reference.*

From the script editor, you can invoke several dialogs.

Add Watch

Use the Add Watch dialog to add a variable to the Script Editor's watch variable list. For more information, see Adding Watch Variables in the *Rational Rose RealTime Extensibility Interface Reference.*

Modify Variable

Use the Modify Variable dialog to change the value of a selected watch variable. For more, information, see Adding Watch Variables in the *Rational Rose RealTime Extensibility Interface Reference*.

Find

Use the Find dialog to locate instances of specified text quickly anywhere within your script. For more information, see Finding Specified Text in the *Rational Rose RealTime Extensibility Interface Reference*.

Replace

Use the Replace dialog to automatically replace either all instances or selected instances of specified text. For more information, see Replacing Specified Text in the *Rational Rose*
RealTime Extensibility Interface Reference.

Calls

Use the Calls dialog to determine the procedure calls by which you arrived at a point in your script when you are stepping through a subroutine. For more, information, see Displaying the Calls dialog in the *Rational Rose RealTime Extensibility Interface Reference*.

Go To Line...

Use the Go To Line dialog to jump directly to a specified line in your script. For more information, see Moving the Insertion Point to a Specified Line in Your Script in the *Rational Rose RealTime Extensibility Interface Reference.*

Dialog Editor

Use the Dialog Editor to insert or edit a dialog in your script. For more information, see Working with the Dialog Editor in the *Rational Rose RealTime Extensibility Interface Reference.*

Aggregation Tool ...

Activates the **Aggregation Tool** to help you create and modify attributes. For more information see Aggregation Tool.

Attribute Tool ...

Activates the **Attribute Tool** to help you create and modify attributes. For more information see Attribute Tool.

Operations Tool ...

Activates the **Operations Tool** to help you create and modify attributes. For more information see Operation Tool.

Model Integrator

Opens the Model Integrator tool.

Web Publisher

Opens the Web Publisher tool.

C++ Analyzer

Opens the C++ Analyzer tool.

Add-Ins menu

Add-In Manager

Opens the Add-In Manager dialog to activate or deactivate add-ins.

Several add-ins are shipped with the Rose RealTime product, including:

- C++ language code generators
- Component Wizard
- Add Dependencies
- Generate Documentation
- C language code generators

Other add-ins will be released through Rational RoseLink partners. See the Rational Rose RealTime web site for links to RoseLink partner add-ins.

Window menu

The **Window** menu has commands for manipulating the windows within the Rose RealTime environment, and a list of all the currently open windows. Use commands from the **Window** menu to control the automatic placement of multiple diagram and specification windows within the application window. You can also use commands on the **Window** menu to redisplay windows that have been covered by other windows or iconified.

To quickly bring a particular window to the forefront, select the name of the window from the menu.

Cascade

Arranges all windows in an even-stepped arrangement. The windows are all sized to a standard size, and the title bar of each window is visible while the body of each window is covered by the next window in front. The most recently-viewed window is completely visible.

Tile Horizontally

Arranges all windows horizontally within the Rose RealTime window. The application window is divided into equal-size areas, with one diagram or specification window in each area. The most recently visited window is placed in the upper left corner. Less recently-visited windows are placed to the right of more recently-visited windows.

Tile Vertically

Arranges all windows horizontally within the Rose RealTime window. The application window is divided into equal-size areas, with one diagram or specification window in each area. The most recently visited window is placed in the upper left corner. Less recently-visited windows are placed below more recently-visited windows.

Arrange Icons

Arranges collapsed windows evenly along the bottom of the Rose RealTime window.

Close

Closes the currently active window.

Close All

Closes all currently open windows.

Window Selectors

The open windows within your application are listed on the **Window** menu with a set of numerical selectors. The windows are listed by title. Selecting one of these items from the menu opens the specified window and brings it to the forefront.

Help menu

What's This?

Opens the context-sensitive Help.

Contents

Opens the Help Table of Contents.

Search...

Opens the Help Search.

Index...

Opens the Help index.

Using Help

Opens a Help topic explaining how the Help system works.

Tutorials

Opens the Tutorials book from which you can choose tutorials based on your skill level and background.

Example Models

Opens the Example Models book.

Keyboard Shortcuts

Opens a Help topic on keyboard shortcuts.

Welcome to Rational Rose RealTime

Opens the Startup Screen.

About Rational Rose RealTime

Opens the About Rose RealTime dialog, which shows information on the product version, add-ins, support contacts, and so forth.

Browsers

Model elements are created and viewed through Browsers. The primary browser is the Model browser, which provides access to all elements of the current model. Browsers list the model elements - usually in a hierarchical way - allowing elements to be expanded to show additional information. Also, most browser lists can be filtered in various ways.

Model browser

The browser is an easy-to-use alternative to menus and toolbars for visualizing, navigating, and manipulating items within your model.

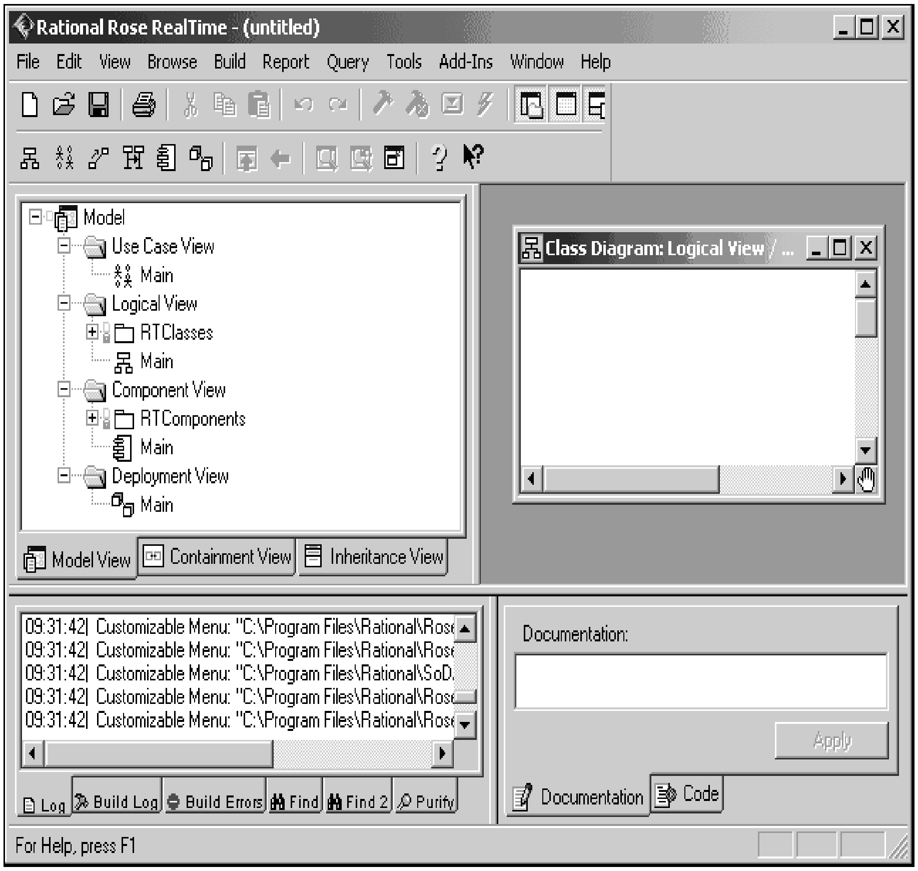
The browser is a hierarchical navigational tool that lets you view the names and icons representing use case, collaboration, deployment and class diagrams, as well as model elements such as logical packages, classes, interfaces, associations and component packages associated with the model.

Model browser contents

The model browser displays all of the elements of the model, organized into the four main views: Use Case View, Logical View, Component View, and Deployment View.

Figure 8 shows the main application window with the model browser open.

Figure 8 Rose RealTime application window with model browser



```
09:31:42| Customizable Menu: "C:\Program Files\Rational\Rose
09:31:42| Customizable Menu: "C:\Program Files\Rational\Rose
09:31:42| Customizable Menu: "C:\Program Files\Rational\SoD.
09:31:42| Customizable Menu: "C:\Program Files\Rational\Rose
09:31:42| Customizable Menu: "C:\Program Files\Rational\Rose
```

Each of the views within the model is shown as a separate folder in the model browser. All model elements created within a view are displayed as sub-elements of that view folder. Views can be expanded and collapsed by clicking on them.

Tabs

There are three tabs on the bottom of the Model Browser:

- Model View tab - shows all the packages, classes and diagrams in the model.
- Containment View tab - shows the containment hierarchy of the capsule classes in the model.
- Inheritance View tab - shows the inheritance hierarchy of the capsule classes, data classes, and protocols in the model.

A fourth tab - the RTS tab - appears when a component instance is run with Target observability enabled. This tab provides a run-time view of the model, showing the list of capsule incarnations, and providing buttons to control the model execution (see *Rose RealTime Execution Interface* on page 320).

Navigating

The plus sign (+) sign next to an icon indicates the item is collapsed, and additional information is located under the entry. Click on the + sign and the tree is expanded. Conversely, a minus (-) sign indicates the entry is fully expanded.

Double-clicking on the diagram name or icon displays the diagram. Double-clicking on any other item displays the associated specification.

Displaying the Browser

When the Browser is first displayed, it is docked along the left edge of the frame. To move the window, click and drag on the border. The window outline indicates the window state: a thin, crisp line indicates the window is docked, while a thicker, hashmark-type border indicates it is floating.

To disable a browser, select it from **View > Browsers**. The check mark is removed along with the display of the browser.

Characteristics unique to the browser state (docked or floating) are discussed below.

Docked:

- The window can be moved within the dockable region of the frame, but it remains positioned along the border.
- The size remains fixed. The free side is resizable.
- A ToolTip displays the icon title when partially covered by the browser border.
- The window can be docked on any border.

Floating:

- The window can be moved to any location, and is always displayed on top of the diagram.
- Size can be changed via click and drag along the border in a vertical or horizontal direction.

Refreshing the Browser

With the mouse positioned inside the browser, click **Refresh** from the shortcut menu.

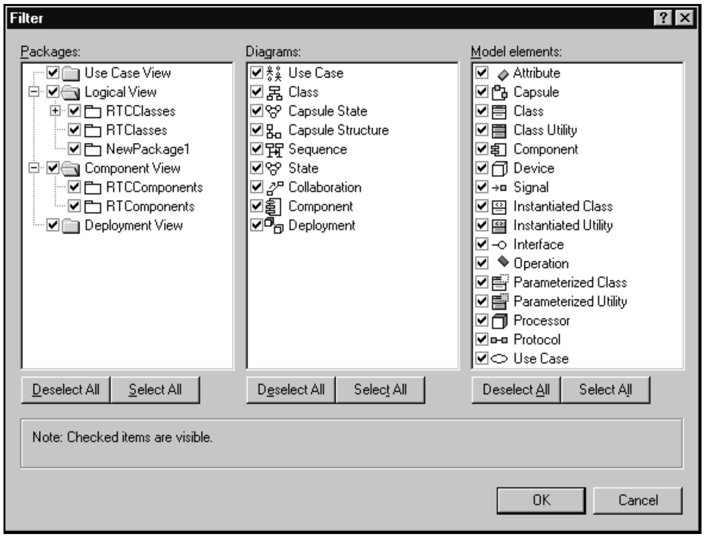
Multiple Browsers

You can have multiple versions of the same browser, and apply filtering that is different between them. For instance, you could open up a second browser and set its filter to show only protocols and their signals.

Filtering

You can filter various packages, diagrams, and model elements using the Filter dialog.

Figure 9 Filter dialog



Packages:
- ✓ Use Case View
- ✓ Logical View
  - ✓ RTCClasses
  - ✓ RTClasses
  - ✓ NewPackage1
- ✓ Component View
  - ✓ RTCComponents
  - ✓ RTComponents
- ✓ Deployment View

Diagrams:
- ✓ Use Case
- ✓ Class
- ✓ Capsule State
- ✓ Capsule Structure
- ✓ Sequence
- ✓ State
- ✓ Collaboration
- ✓ Component
- ✓ Deployment

Model elements:
- ✓ Attribute
- ✓ Capsule
- ✓ Class
- ✓ Class Utility
- ✓ Component
- ✓ Device
- ✓ Signal
- ✓ Instantiated Class
- ✓ Instantiated Utility
- ✓ Interface
- ✓ Operation
- ✓ Parameterized Class
- ✓ Parameterized Utility
- ✓ Processor
- ✓ Protocol
- ✓ Use Case

Note: Checked items are visible.

Diagram Editors

There are several different kinds of diagrams that can be created and edited through Rose RealTime. Each diagram allows you to specify or document a different aspect of the model. Some diagrams are accessible in only one view, while other diagrams are found in more than one view. Each icon on a diagram represents an element in the model. Since diagrams are used to illustrate multiple views of a model, each model element can appear in none, one, or several of a model's diagrams. This means you can control which components and properties appear on each diagram.

The following is the complete list of diagrams available in Rose RealTime:

- Class diagram
- Use case diagram
- Collaboration diagram
- Sequence diagram
- Structure diagram
- State diagram
- Component diagram
- Deployment diagram
- Structure Monitor diagram
- State Monitor diagram

See the individual diagram topics for information on creating or modifying that particular diagram.

Each editor is displayed in a separate window. The diagram editors all have associated toolboxes.

Adding Icons to a Diagram

To create or add icons to a diagram, you can

- use tools on the toolbox
- use the drag and drop capabilities of the Model browser, which you can undo and redo from the Edit menu
- invoke commands from the Query menu, which add icons representing specific model elements

You can cut, copy, and paste icons between different diagram windows using commands on the Edit menu. The **Edit** menu also provides commands that enable you to select, find, and rename icons. The Browse menu provides commands to navigate among diagrams, as well as create, rename, and delete them. Commands on the Report menu provide additional diagram navigation capabilities. You can print diagrams using the Print command (see *Print Specifications* on page 123).

Opening Specifications

Clicking the Specification command from the Browse menu displays the specification for the model component represented by the selected icon.

Popup menu

Clicking the right mouse button on an icon activates the popup menu, which enables you to modify properties (for icons that represent relationships) or select properties to be displayed within the icon. Select the Open Specification command from the popup menu to open the specification dialog.

Background Popup menu

The Background popup menu changes according to the diagram in which you click. Following are some basic menu items:

Zoom

Zooms in on the current diagram. Select the zoom level.

Scale to Window

Enables the automatic resizing of icons to accommodate text.

Layout

Opens a submenu of options for rearranging the diagram:

Layout Diagram

Analyzes the location of all icons in the current diagram, determines the optimal location for the icons, and redraws the diagram.

Align/Distribute

The selected objects are arranged according to the choices made in the dialog. Alignment and distribution operations can be performed in both horizontal and vertical arrangements.

Change View Spread

Creates space in a diagram for adding new views or creates a cleaner appearance. There are a number of different ways the views can be spread out by specifying the Spread Technique.

- Uniform - indicates that the views are spread out across the diagram uniformly by the percentage. If a view is at location (100,100) and they specified a horizontal percentage of 10% and a vertical percentage of -10%, the new location of the view would be (110, 90). This affects all views in the diagram the same way.
- Constant Radial - indicates that the views spread outward/inward from a central point. The preview displays a crosshair that specifies where the spread starts. It can be moved around interactively in the preview window with the mouse. The views spread out a constant distance based on the diagram size.
- Decreasing Radial - is similar to Constant Radial except that the views spread progressively less far the farther away from the central point they are.
- Increasing Radial - is similar to the Constant Radial except that the views spread progressively more the farther away from the central point they are.

The preview allows the user to play with the settings until the desired spread is achieved.

Autosize All

Resizes all node views in the diagram to fit their labels.

Make Same Size

Makes two or more node views the same size in either height or width, or both. You can choose from smallest, average, or largest of all the selected views to make the new width and height. The dialog provides a preview screen.

Select in Browser

Selects an element from the diagram in the browser.

Filter

Use these options to filter information on diagrams.

Scroll Bars

Diagram windows provide vertical and horizontal scroll bars to pan across diagrams larger than the window.

Overview Navigator

You can also use the Overview Navigator - the hand located in the bottom right-hand corner of the diagram - to navigate around a diagram.

Toolboxes

Every diagram has an associated toolbox, which contains icons of tools that can be applied to that diagram. If the current diagram is write-protected, the diagram toolbox is not displayed. The diagram toolbox is also only available when a diagram is displayed.

There are several tools that are common to every toolbox:

Selector tool

Use to select objects for moving, resizing, and so forth.

Zoom tool

Use to zoom in on a portion of the diagram. Click on the tool and then click on the part of the diagram you want to zoom in on.

Text tool

Use to add text anywhere in the structure diagram.

Note tool

Use to annotate the diagram with textual notes. This is useful for marking up the diagram with explanations, review comments, and so forth. You can also hyperlink to diagrams or URLs.

Constraint tool

Use to add UML constraints to any diagram. A constraint can be anchored to a view element by using the anchor tool. Currently, constraints do not have any semantic meaning to the tool. There are RRTEI APIs to add or remove, and enumerate constraints in a diagram.

Note anchor tool

Use to anchor a note to a particular element on the diagram. Allows notes to be moved with the element they are anchored to.

Lock Selection tool

Use to make the tool selections stay locked. That is, if the lock is on, then the next tool you select will stay selected after you've completed the operation. This allows you to perform a number of operations with a particular tool without having to reselect the tool after each operation.

You may also hold down the shift key to keep a tool selected. The last selected tool stays active until you release the shift key.

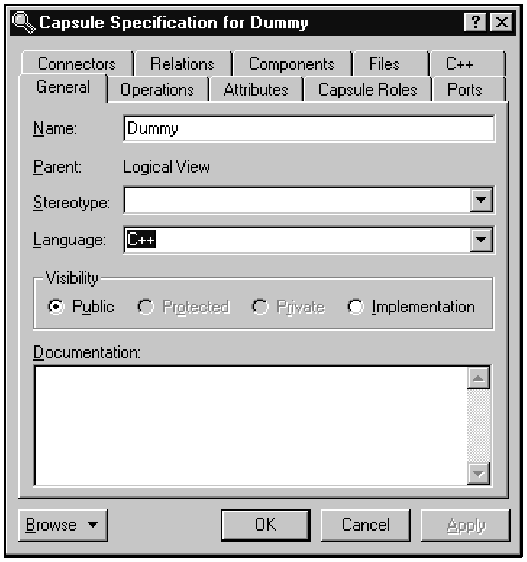
These tools are available in the following toolboxes:

- Use Case Diagram Toolbox
- Class Diagram Toolbox
- Structure Diagram Toolbox
- State Diagram Toolbox
- Collaboration Diagram Toolbox
- Sequence Diagram Toolbox
- Component Diagram Toolbox
- Deployment Diagram Toolbox

Specification Dialogs

Specification dialogs are used to edit the properties of any element in the model. All specification dialogs contain at least a Name and Documentation field. Some specification dialogs contain many fields split across different tabs. The example below shows a specification dialog with multiple tabs. All properties of a modeling element are accessible through the specification dialog for that element. Many of the properties on the specification dialog are also visible/editable through one or more diagram editors.

Figure 10 Sample specification dialog for a capsule


Capsule Specification for Dummy

Name: Dummy

Parent: Logical View

Stereotype:

Language: C++

Visibility: [✓ Public] Protected Private Implementation

Documentation:

Specification dialogs are resizable. The tab you are in is remembered so that the next time you open the specification dialog you go to the same tab. The position and size of specification dialogs are saved with the workspace.

Spreadsheet-type functionality for list controls within a specification dialog

When the list control has focus, the following applies:

- F2 or Enter key puts the field in inline edit or drop down combination mode. Press the Enter key again to accept the data and move to the next row in the column.
- The Tab key also accepts data (when editing a cell) and goes to the next column of the same row. If you are in the last column, it moves to the first column of the next row. If you are in the last column of the last row, it inserts a new row and begins inline editing.
- When not editing a cell, the Shift + Tab combination works as the reverse of Tab; that is, it moves to the previous column in the same row, or if you are in the first column it moves to the last column of the previous row. If you are in the top-left most cell, it does nothing.

Browse button

Clicking Browse displays four choices:

- Select in Browser - highlights the selected element in the browser
- Browse Parent - opens the specification for the parent of the selected element.
- Browse Selection - opens the specification for the currently selected element.
- Show Usage - displays a list of all diagrams in which the currently selected element is the supplier, or in the case of a collaboration diagram, a list which shows the usage of a message.

OK button

Accepts any changes and close the dialog.

Cancel button

Ignores any changes that were made and close the dialog.

Apply button

Commits any changes that were made.

Help button (?)

Opens the online help for the current specification dialog.

Exit button (X)

Closes the specification dialog.

Tabs

Many dialogs include a number of tabs across the top for grouping different specification information. The Files tab, Relations tab, Components tab, Attributes tab, Operations tab, and Unit Information tab may be displayed for many different model elements. Unit Information tab only appears if the model is being controlled as units.

Files tab

A list of referenced files is provided here. The files list popup menu allows you to insert and delete references to files or URLs.

You can link external files to model elements for documentation purposes.

Relations tab

The relations list displays relations between classes as specified in diagrams. Relations can be inserted, deleted, and moved up and down in the list. Each relation has a corresponding Association specification for editing the relation attributes.

A check-box provides filtering control over which relations are displayed:

Show Inherited shows any relations inherited from a superclass.

Components tab

The components list displays a list of components to which this class has been assigned. Components can be inserted, deleted, and moved up and down in the list. Each component has a corresponding Component Specification for editing the component attributes.

A check-box provides filtering control over which components are displayed:

Show all components displays the list of all components in the model.

Right-clicking on a component brings up the Components popup menu.

Attributes tab

The UML asserts that attributes are data values (string or integer) held by objects in a class. Thus, the Attributes tab lists attributes defined for the class. The attribute definition can be modified through the Attribute Specification dialog.

Note: Attributes and relationships created using this technique are added to the model, but do not automatically appear in any diagrams. That is, adding an attribute affects the code generation for the class and a compilation dependency between the class of the container and the class of the attribute, but these relationships are not graphically visible in the model.

The descriptions for each field follow:

- Visibility Adornment (Unlabeled):
  - Public - the attribute is publicly visible, and is accessible to all clients.
  - Protected - the attribute may be accessed only by subclasses, friends, or by operations of this class.
  - Private - the attribute is accessible only by the class itself or by its friends.
  - Implementation - the attribute is accessible only by other operations in this class.
- Stereotype - displays the name of the stereotype.

- Name - displays the name of the attribute.
- Class - identifies where the attribute is defined.
- Type - this can be a class or a traditional type, such as int.
- Initial - displays the initial value of an object.

The Attribute tab is active for all class types.

Show Inherited

Click this option to see attributes inherited from other classes. If there is no check mark in this field, you can view only attributes associated with the selected class.

Note: Rose RealTime allows you to directly modify any attribute shown in the attributes list by displaying the attribute specification dialog. You should be careful when modifying base class attributes for it may have implications on other elements in your model which reference or are subclassed from the base class.

Creating New attributes

You can add an attribute relationship by selecting **Insert** on the popup menu or by pressing the insert key. A new attribute with a default name is added.

Moving and copying attributes

To move an attribute from one Specification sheet to another, drag and drop it. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

To copy an attribute from one Specification sheet to another, drag and drop it while holding down the `CTRL` key. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

Operations tab

Operations denote services provided by the class. Operations are methods for accessing and modifying Class fields or methods that implement characteristic behaviors of a class.

The Operations tab lists the operations that are members of this class. The actual definition of the operation is accessible from the Operation Specification.

The operations are listed with the following fields:

- Visibility Adornment (Unlabeled); the visibility of the operation is indicated with an icon. Following are the visibility options:
    - Public - the operation is accessible to all clients.

    - Protected - the operation is accessible only to subclasses, friends, or to the class itself.
    - Private - the operation is accessible only to the class itself or to its friends.
    - Implementation - the operation is accessible only by operations of this class.
- Stereotype - displays the name of the stereotype.
- Signature - displays the name of the operation.
- Class - identifies which class defines the operation.
- Return Type - identifies the type of value returned from the operation.

The Operation tab is active for all class types. In the class diagram, you can display operation names in the class compartment.

Show Inherited

Click this option to see operations inherited from other classes. If there is no check mark in this field, you can view only operations associated with the selected class.

Note: Rose RealTime allows you to directly modify any operation shown in the operations list by displaying the operations specification dialog. You should be careful when modifying base class operations for it may have implications on other elements in your model which reference or are subclassed from the base class.

Creating New Operations

To enter an operation in the Class Specification, select **Insert** from the popup menu. A new operation with a default name is added to the operations list.

Moving and copying Operations

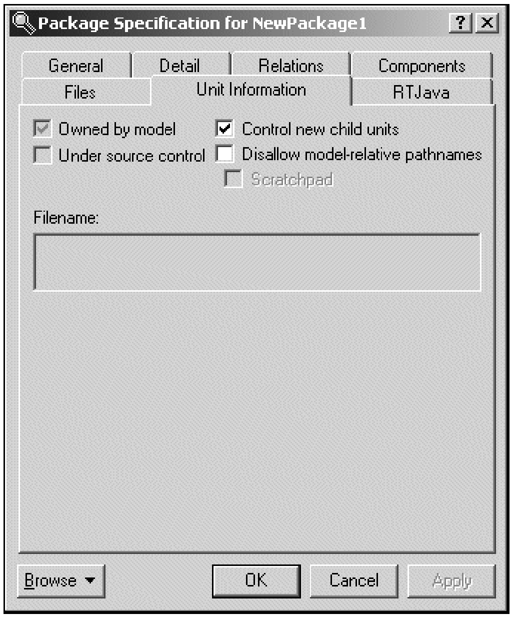
To move an operation from one Specification sheet to another, drag and drop it. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

To copy an operation from one Specification sheet to another, drag and drop it while holding down the `CTRL` key. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

Unit Information tab

The specification dialog for a controlled element includes a Unit Information tab.

Figure 11 Unit Information tab



[✓] Owned by model
[ ] Under source control
[✓] Control new child units
[ ] Disallow model-relative pathnames
[ ] Scratchpad

Owned by model

Indicates whether the unit is owned by this model or whether it is owned by another model and shared into this model. This setting is not directly editable.

Under source control

Indicates whether this element has been added to source control. This setting is not directly editable.

Control new child units

Controls whether newly created controllable elements in this package will be individually controlled by default. This check box is only displayed in the Unit Information tab for a package.

Disallow model-relative pathnames

Informs Rose RealTime to not use the implicit $@ virtual pathmap symbol when saving units located anywhere within this package. This check box is only displayed in the Unit Information tab for a package.

Scratchpad

Indicates that the package is a scratch pad. This check box is only enabled in the Unit Information tab for a package that is not under source control.

Filename

Displays the name of the file that is used to save this controllable unit. This field is not directly editable.

Version

Displays the version identifier for this controlled unit. If this information is not known, then '<unknown>' is displayed. The ability to extract this version information depends on the source control tool being used. If a unit is not under source control, then this field is not displayed.

Scratch Pad Packages

When working on a model in a team environment, it is common for a developer to create temporary model elements that are not intended to be shared with the rest of the team. For example, a developer may create a temporary component when unit testing a change to a capsule class. If the model is under source control, then the developer will also not want these temporary elements to be checked in with the other changes they are making.

In order to support temporary work within a controlled model, Rose RealTime supports scratch pad packages. A scratch pad package is a package that will never be added to source control. Also, in a model that is under source control, changes can be made to a scratch pad package without the toolset requiring that package be checked out. This allows multiple team members to make temporary changes within the scratch pad without encountering any contention issues.

Elements can be moved into or out of a scratch pad package by dragging them to another package in the browser. Elements can also be copied into (or out of) a scratch pad package using control-drag.

The controllable elements within a scratch pad package cannot be individually controlled. If a controlled unit is moved into a scratch pad package, then it will no longer be controlled.

To create a scratch pad package:

1 Create a package and give it a descriptive name, for example, ScratchPad.

2 Select the package in the browser and choose **File > Control Unit**.

3 Open the Specification dialog for this package and change to the Unit Information tab.

4 Select the Scratchpad and click `OK`.

5 Save the package containing the scratch pad. Optionally you can also save the scratch pad. If the containing package is under source control, it should be checked out and checked in.

Searching and Sorting

Using Sort

Sorting in the Browser

Sorting in the browser enables you to arrange classes, attributes, operations, and packages in alphabetical order.

Follow these steps to place items alphabetically in the browser:

1 Highlight a class or package icon.

2 Right-click on the icon.

3 Click Sort in the shortcut menu.

Note: The arranged items in the browser are not saved when you close the application.

Right-clicking in an application browser displays the popup menu, in which Sort is an item. Choose between Alphabetical Order and Internal Order. Sort settings are saved for each browser in your workspace.

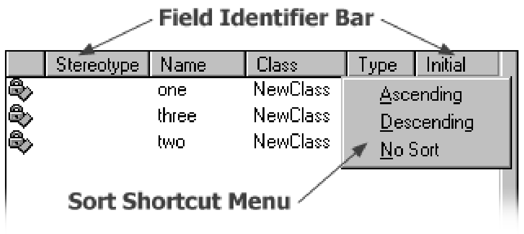
Sorting in the Class Specification

Sorting in class specification enables you to arrange attributes and operations three ways.

Follow these steps to arrange items in class specification:

1 Highlight an attribute or operation.

2 Right-click on any Field Identifier bar located in class specification.



3 Click either **Ascending, Descending,** or **No Sort** from the shortcut menu options.

- Ascending - lists attributes and operations in case-sensitive alphabetical order.
- Descending - lists attributes and operations in reverse, case-sensitive alphabetical order.
- No Sort - lists attributes and operations in the order they are specified in the model.

Find dialog

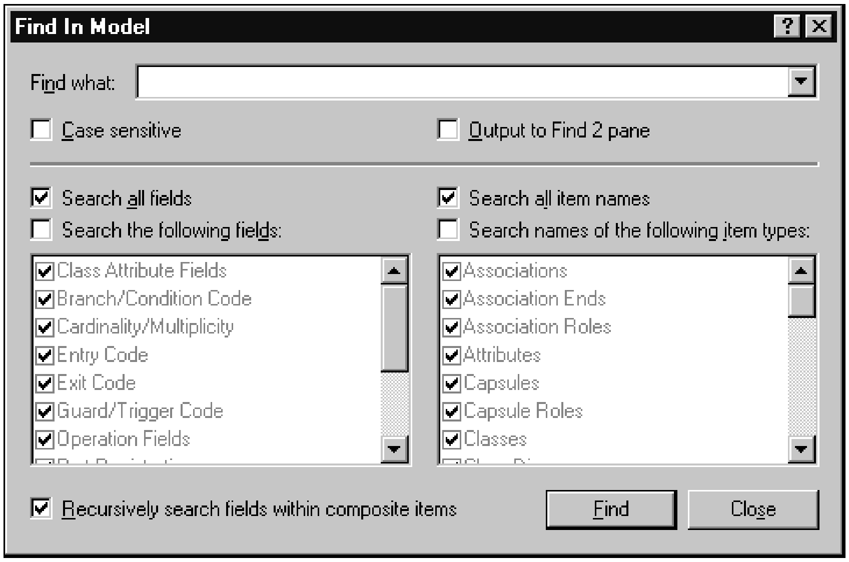
You can use the Find dialog to search detail code segments or to search for model elements by name. Results are displayed in the Find tab of the Output window (**View > Output**) or optionally in the Find 2 tab of the Output window.

Searching code

Use the Find dialog to find all uses of a class in detail code or a class that is a property of another class. For example, you could find attributes of a specific type or signal data types of a specific name.

The Find dialog searches for the specified text in all detail level code in a model: transitions, entry/exit code, choice points, guards, and operations. It also searches the dimension and type property of all attributes, the data type of signals, documentation, and language tab properties.

Figure 12 Find dialog



Searching for model elements by name

You can also search the model for elements whose name matches the string specified as the Item to find. Selective searching based on the type of element is also supported, for example, search only capsules or state diagrams.

Once a search is complete, all matches are shown in the Find tab of the Output window (**View > Output**). For most matches, you can either double-click or press the Browse button to go directly to the specification dialog or diagram.

Selective searching

You can toggle several items at once by pressing the **Shift** key, then selecting a set of elements types from the list, then using the **SPACEBAR** to toggle the check boxes on or off.

Replace dialog

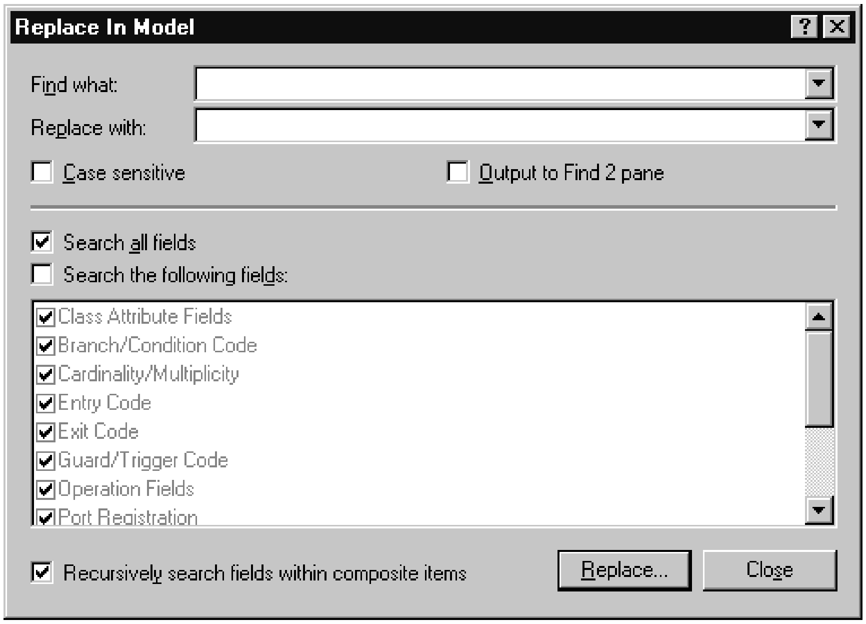
You can use the Replace dialog to search and replace detail code segments or to search and replace for model elements by name. Results are displayed in the Find 2 tab of the Output window (**View > Output**).

Searching code

Use the Replace dialog to find and replace all uses of a class in detail code or which is a property of another class. For example, you could find and replace attributes of a specific type or signal data types of a specific name.

The Replace dialog searches and replaces for the specified text in all detail level code in a model: transitions, entry/exit code, choice points, guards, and operations. It also searches the dimension and type property of all attributes, the data type of signals, documentation, and language tab properties.

Figure 13 Replace dialog



Replace In Model

Find what:

Replace with:

☐ Case sensitive

☐ Output to Find 2 pane

☑ Search all fields

☐ Search the following fields:

☑ Class Attribute Fields
☑ Branch/Condition Code
☑ Cardinality/Multiplicity
☑ Entry Code
☑ Exit Code
☑ Guard/Trigger Code
☑ Operation Fields
☑ Port Registration

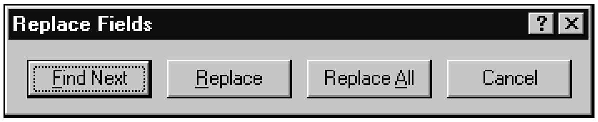
☑ Recursively search fields within composite items

Searching for Model Elements by Name

You can also search and replace the model for elements whose name matches the string specified as the Item to find. Selective searching and replacing based on the type of element is also supported, for example, search and replace only capsules or state diagrams.

When you click Replace, a secondary dialog (Figure 14) appears with options t o Find Next, Replace, Replace All, and Cancel. After the replace operation has taken place, you can query the Find tab in the Output window to view the results of the search and replace.

Figure 14 Replace Fields dialog



Selective searching

You can toggle several items at once by pressing the `SHIFT` key, then selecting a set of elements types from the list, then using the **SPACEBAR** to toggle the check boxes on or off.

Wizards and Tools

3

Component Wizard

To run your model, you must build it and then execute it on a processor. A component describes how to build a set of capsules and classes. The Component Wizard helps you to quickly create C++ and C Executable components. It allows you to configure the following component properties:

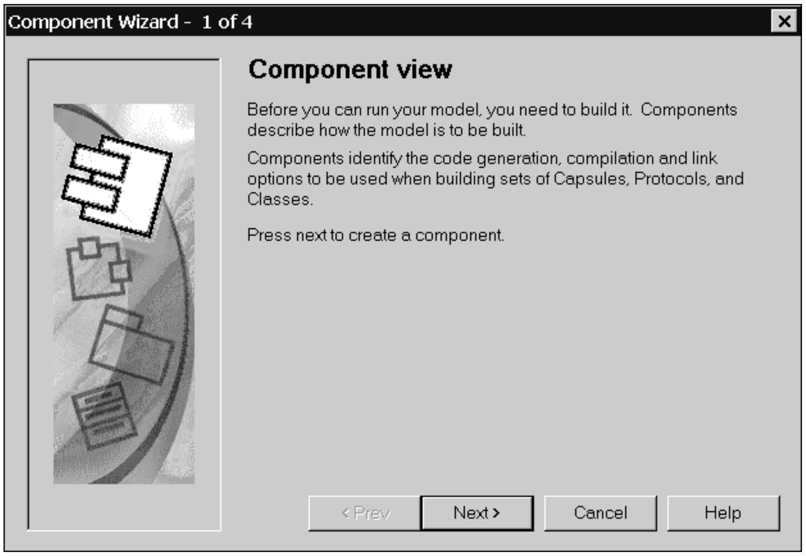
- Language
- Top level capsule
- Output directory
- Executable name
- Target Configuration

The Component Wizard is available from the **Build > Component Wizard** menu item.

Note: You cannot create Libraries and External Libraries with the **Component** wizard.

To provide the initial information for a component:

1 From the **Build** menu, click **Component Wizard.**



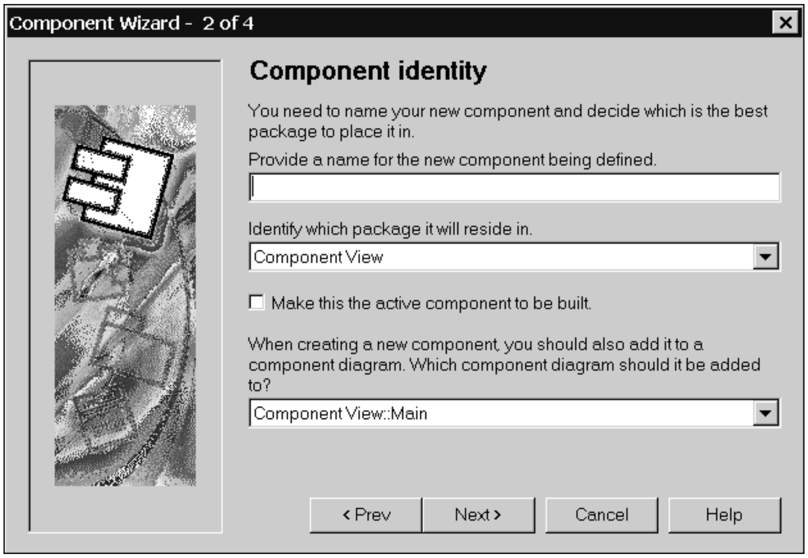
Component view

Before you can run your model, you need to build it. Components describe how the model is to be built.

Components identify the code generation, compilation and link options to be used when building sets of Capsules, Protocols, and Classes.

Press next to create a component.

2 Click **Next** to start.



Component Wizard - 2 of 4

Component identity

You need to name your new component and decide which is the best package to place it in.

Provide a name for the new component being defined.

Identify which package it will reside in.

Component View

☐ Make this the active component to be built.

When creating a new component, you should also add it to a component diagram. Which component diagram should it be added to?

Component View::Main

3 Specify a meaningful name for the component.

4 Specify the location of the package in the **Model View** tab in the browser. The default location is the **Component View** package.

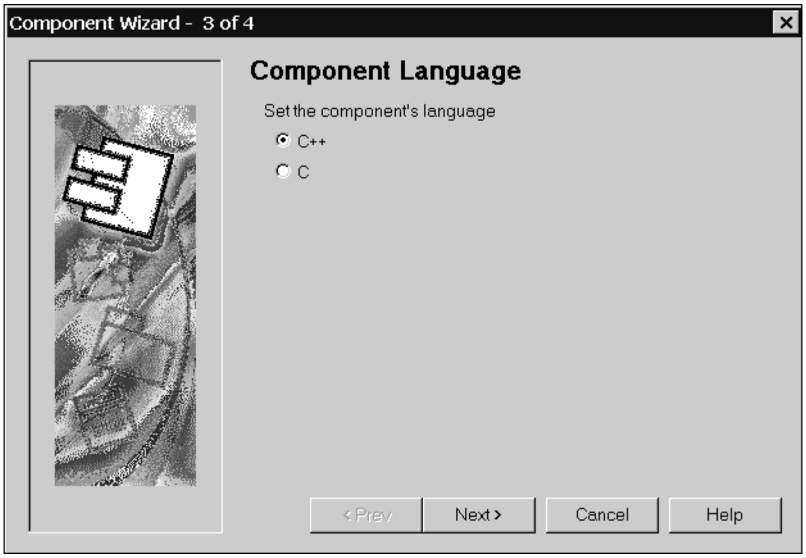
5 Indicate whether this component is the active component.

If you find yourself building and running the same component and component instances often, set this component as an active component. When a component is configured as being active, the Toolbar build icons and menu items become available for easy access to common build and run commands. In addition, you can configure which component instances (executables) automatically run when you click **Run**. You can set this option later by selecting the component from the **Model View** tab in the browser, then right-click and select **Set As Active**.

6 Select the name of the diagram to add the component instance.

Adding the component instance to the diagram provides you with a graphical representation of the components in your model.

7 Click **Next**.


Component Wizard - 3 of 4

Component Language

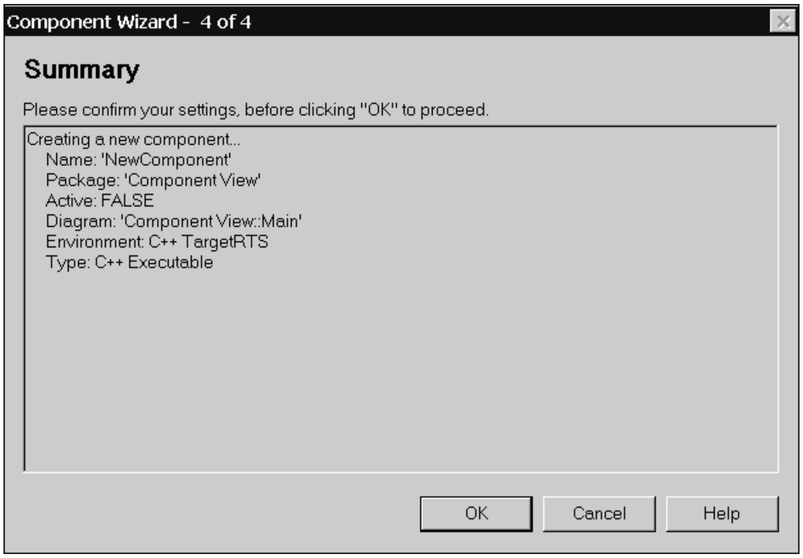
Set the component's language

(●) C++

( ) C

8 Select a language for the component.

9 Click **Next**.


Component Wizard - 4 of 4

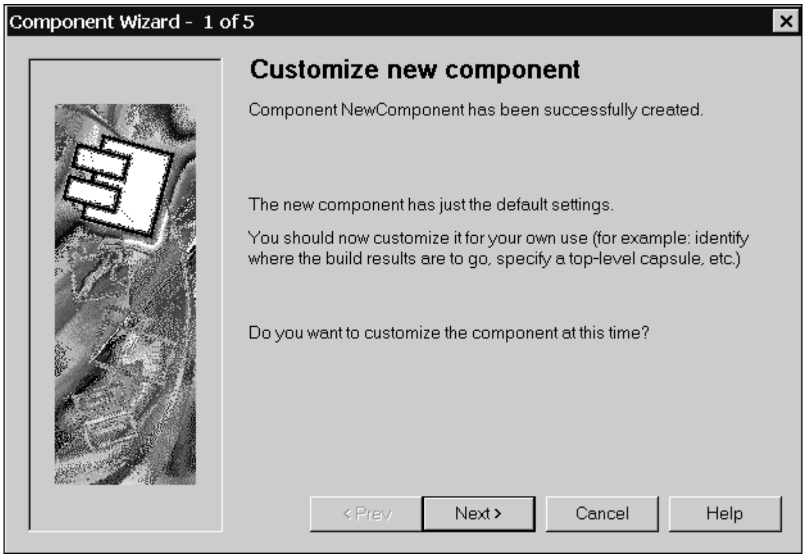
Summary

Please confirm your settings, before clicking "OK" to proceed.

```
Creating a new component...
   Name: 'NewComponent'
   Package: 'Component View'
   Active: FALSE
   Diagram: 'Component View::Main'
   Environment: C++ TargetRTS
   Type: C++ Executable
```

10 Review the summary of specified settings.

11 Click **OK** to proceed.


Component Wizard - 1 of 5

Customize new component

Component NewComponent has been successfully created.

The new component has just the default settings.

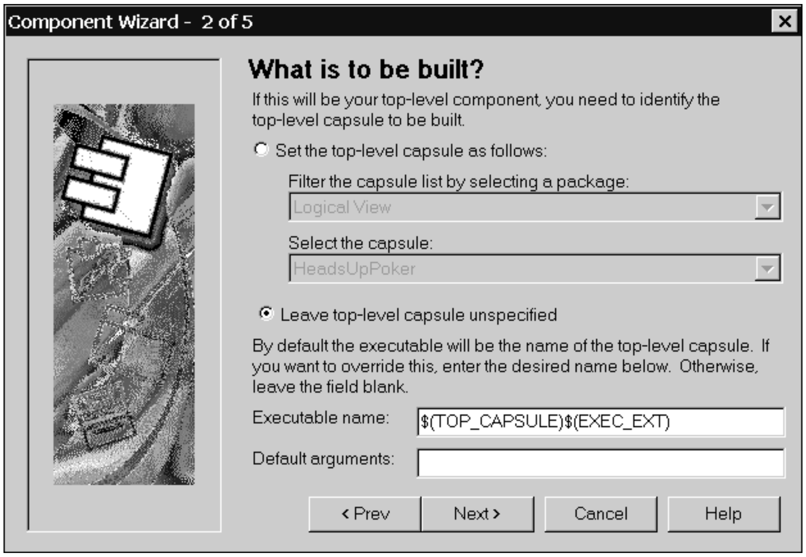
You should now customize it for your own use (for example: identify where the build results are to go, specify a top-level capsule, etc.)

Do you want to customize the component at this time?

12 Click **Next** to customize the component (recommended).

Or . . .

Click **Cancel** to create the component without customization and to close the **Component Wizard**.


Component Wizard - 2 of 5

What is to be built?

If this will be your top-level component, you need to identify the top-level capsule to be built.

○ Set the top-level capsule as follows:

Filter the capsule list by selecting a package: Logical View

Select the capsule: HeadsUpPoker

● Leave top-level capsule unspecified

By default the executable will be the name of the top-level capsule. If you want to override this, enter the desired name below. Otherwise, leave the field blank.

Executable name: $(TOP_CAPSULE)$(EXEC_EXT)

Default arguments:

13 You can either set the top level capsule or leave it unspecified.

If you specify a top level capsule to compile for this component, the top capsule will define the compilation closure for the component. All classes, including capsule and protocol classes referenced directly or indirectly by the top capsule are then compiled as part of the component.

14 Specify an **Executable name**.

You can specify the name, or a name with an absolute path, of the executable that is created when the component is built. By default, the executable name is set to the name of the component's top level capsule.

Note: If an absolute path is not used in the **Executable name** box, the location of the executable will be in the following component build output directory:

<output_dir>/build

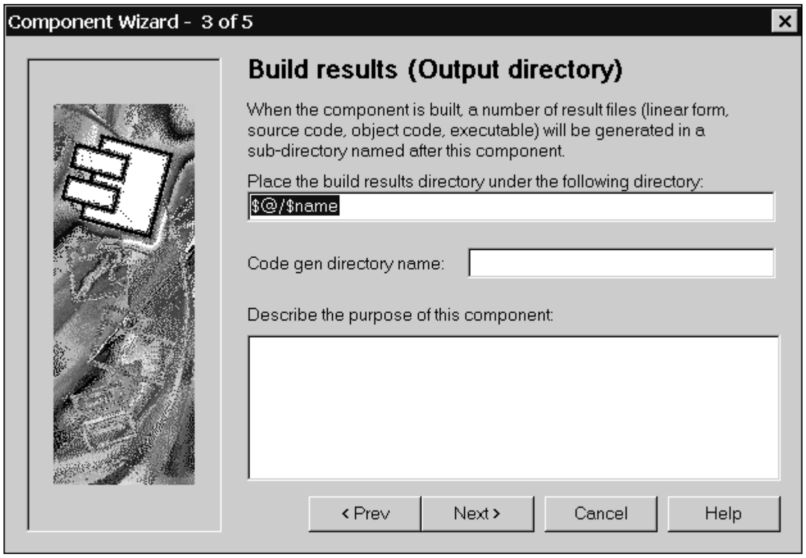
15 Specify any **Default arguments**.

Some platforms do not permit the passing of command line arguments to an executable at load time. As a result, the **Default arguments** box provides a mechanism for getting execution arguments into the executable. You can use `RTMain::argStrings()` to retrieve any passed command line argument within your model. Type a comma-separated list of quoted arguments into this box, such as:

"134.434.344.4","barneyht","delay=98"

The **Default arguments** box is only for targets that cannot accept command line arguments. Targets that can accept command line arguments ignore anything in the **Default arguments** box.

16 Click **Next**.


Component Wizard - 3 of 5

Build results (Output directory)

When the component is built, a number of result files (linear form, source code, object code, executable) will be generated in a sub-directory named after this component.

Place the build results directory under the following directory:

$@/$name

Code gen directory name:

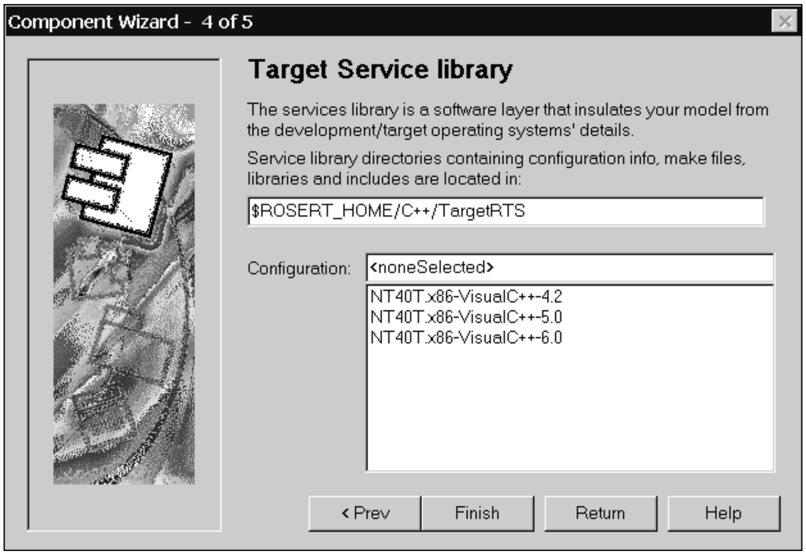
Describe the purpose of this component:

17 Specify the location for the build results.

18 Specify the name of the Code gen directory.

19 Provide a description that identifies the purpose of this component.

20 Click **Next**.


Component Wizard - 4 of 5

Target Service library

The services library is a software layer that insulates your model from the development/target operating systems' details.

Service library directories containing configuration info, make files, libraries and includes are located in:

$ROSERT_HOME/C++/TargetRTS

Configuration: <noneSelected>

NT40T.x86-VisualC++-4.2
NT40T.x86-VisualC++-5.0
NT40T.x86-VisualC++-6.0

21 Specify the location of the Services Library directories that contain the configuration information, make files, libraries and include files.

Specify the path to the root directory for the specific Services Library desired. This name must be specified as a full path to the root directory of the Services Library.

The Target Services directory contains all the scripts and programs to generate and compile a component. If this directory is not configured correctly, you will not be able to successfully generate or compile.

By default this field references the Services Library in your Rose RealTime home directory $ROSERT_HOME/C++/TargetRTS. You can change this location to any other directory that contains the C++ Services Library.

22 Select a **Configuration**.

This property uniquely identifies the configuration of the Services Library used to compile and link the component. The configuration name is composed of three parts: os.processor-compiler-version.

For example, the configuration for a Windows NT 4.0 multi-threaded platform with an x86 processor built with Microsoft Visual C++ version 6.0 is:

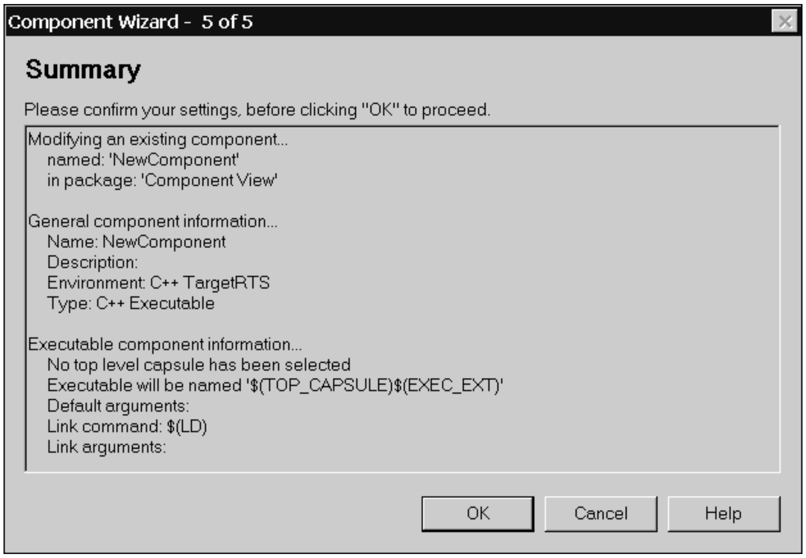
```
NT40T.x86-VisualC++-6.0
```

To view the valid configuration names, examine the directories located in the \lib subdirectory of the Services Library root. If you build different configurations of the Services Library, the new configuration appear in this list.

23 Click **Finish**.

Or . . .

Click **Return** to modify previous settings.

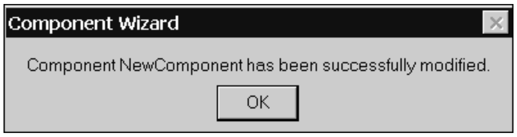


Please confirm your settings, before clicking "OK" to proceed.

```
Modifying an existing component...
    named: 'NewComponent'
    in package: 'Component View'

General component information...
    Name: NewComponent
    Description:
    Environment: C++ TargetRTS
    Type: C++ Executable

Executable component information...
    No top level capsule has been selected
    Executable will be named '$(TOP_CAPSULE)$(EXEC_EXT)'
    Default arguments:
    Link command: $(LD)
    Link arguments:
```

24 Review the contents of the Summary window.

25 Click **OK** to create the component.



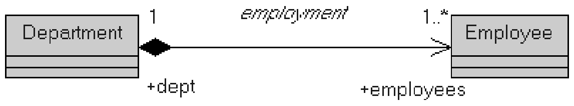
Component NewComponent has been successfully modified.

26 Click **OK** and verify your component in the **Model View** tab in the browser.

Aggregation Tool

The **Aggregation Tool** enables you to quickly create aggregate and composite associations. An aggregation association is a special form of association that specifies the whole-part relationship between an aggregate (whole) and the component (part). There are many examples of aggregation relationships: within a Department there are Employees, and a Computer is composed of a number of Devices.

Figure 15 Aggregation Association

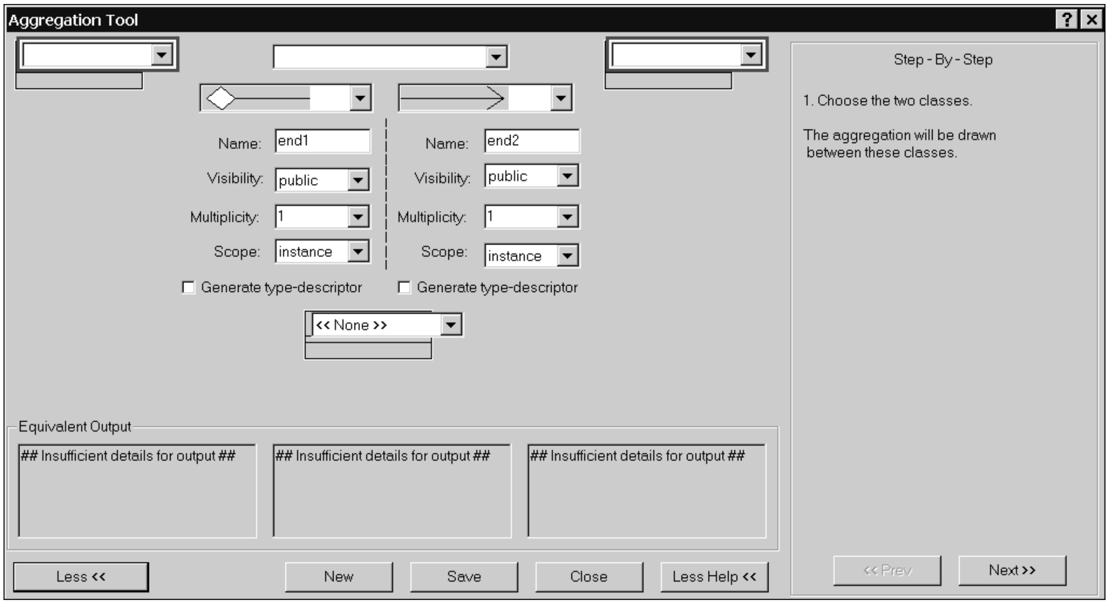


An aggregation is just a special kind of association. Use the **Aggregation Tool** to create a new aggregation, or modify an existing one. You can access the **Aggregation Tool** from the following:

- **Tools > Aggregation Tool . . .**
- select an association in a class diagram, then right-click and select **Aggregation Tool . . .**
- select two classes, then right-click and select **Aggregation Tool . . .**

Note: If you use the **Aggregation Tool** to create or modify an aggregation, the model diagrams are not automatically updated. To update your model diagrams, you must use **Query > Filter Relationships**.

Figure 16 Aggregation Tool


Aggregation Tool

Step - By - Step

1. Choose the two classes.

The aggregation will be drawn between these classes.

Name: end1	Name: end2
Visibility: public	Visibility: public
Multiplicity: 1	Multiplicity: 1
Scope: instance	Scope: instance
☐ Generate type-descriptor	☐ Generate type-descriptor

<< None >>

Equivalent Output

Insufficient details for output

Insufficient details for output

Insufficient details for output

Class Name

Specifies the name of a class included in the association.

Association Name

Specifies the name of the association. This name should describe the nature of the relationship between the two classes.

Aggregation Association Type

Specifies the type for the association end. The end types are:

- ◇——— - represents an aggregation
- ◆——— - represents a composition
- ——— - an association end (other end can be <——— or ———)
- <——— - an association end (other end will be ———)

Name

Specifies the name for the association end. The end of each association is called an association end or an end. You can label ends with an identifier that describes the role that an associate class plays in the association. This name should describe the nature of the end for the specific class.

Visibility

Specify the type of visibility for each end of an aggregation. There are four types of visibility:

- **Public** - the attribute is visible to other classes.
- **Protected** - the attribute is visible only to subclasses and *friend* classes.
- **Private** - the attribute is not visible to other classes, except designated *friend* classes.
- **Implementation** - the attribute is never visible to other classes.

See the **Equivalent output** box in the **Attribute Tool** dialog to view an approximation of the output for the specified **Visibility**.

Multiplicity

Specifies the number of instances that can exist for this end of the association at any given time. You can either select a multiplicity from the drop-down list or specify your own by typing directly into this box.

When you specify a multiplicity at one end of an association (near end), for each object of the class at the opposite end (far end), there may be that many objects at the other end (near end).

See the **Equivalent output** box in the **Aggregation Tool** dialog to view an approximation of the output for the specified **Visibility**.

Scope

Specifies the target scope; whether there is only a single instance of the feature for all instances of the classifier. There are two types of scope:

- Instance -the instances of the client own the supplier class. This means that each instance of the classifier holds its own value.
- Classifier - the client class (not the client's instances) owns the supplier class. The most common use of this type of scope is for private data members that must be shared among a set of instances; no other instances have access to that attribute.

The result is that the data member is scoped to the classifier in the other end class.

Generate Type Descriptor

Creates a type descriptor for the external types in order to describe the types to the Services Library. A type descriptor is a unique string that represents the class. If a descriptor is not generated, the Services Library will not be able to encode or decode the attribute.

Association Class

Specifies the name of a class that defines the relationship between the two classes. Use the association class to model properties of associations. The properties are stored in the class and linked to the association relationship. Link Attributes are degenerate association classes comprised only of attributes.

Equivalent Output

Displays a "best" approximation of the expected output for the options selected in the **Aggregation Tool** dialog.

For a new aggregation, the **Equivalent Output** boxes display the following:

```
## Insufficient Information For Output ##
```

This means that there are insufficient options specified for the selected or new aggregation.

To view all of the **Equivalent Output** code in a single pane for the selected aggregation, right-click inside the **Equivalent Output** box. The pop-up window resizes to contain all of the code from the **Equivalent Output** box.

Note: You can copy code from the **Equivalent Output** box; however, the code within this box is only an approximation and may not represent the precise code segment. Therefore, when copying from the Equivalent Output box, use caution.

More/Less button

Click this button to expand or collapse the advanced area of the tool. This area displays any association classes and additional options for this aggregation.

New

Click this button to add a new aggregation. Ensure that you click **Save** before you click **New** to save any changes to the previous aggregation.

Save

Saves the settings for the current aggregation.

More Help/Less Help button

Hides or displays a help pane which provides step-by-step instructions to create an aggregation.

Step By Step Area

Provides step-by-step instructions to create an aggregation. For additional information, see the Online Help.

Language Specific Options

The advanced area contains options for language-specific features. For information on these options, refer to the Online Help.

Attribute Tool

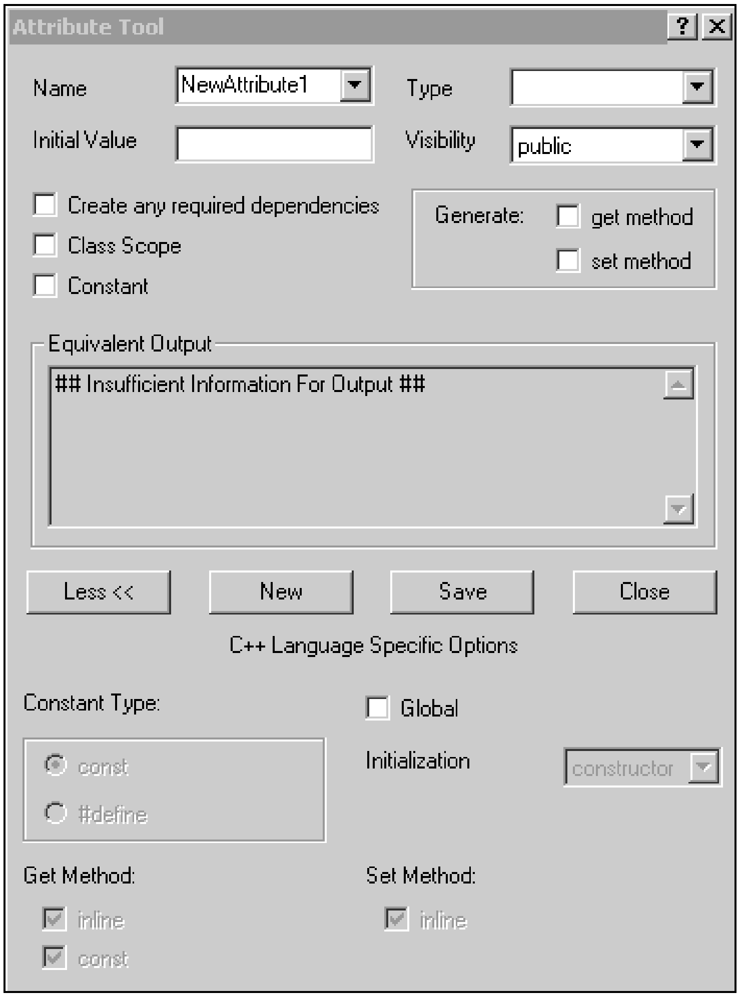
The **Attributes Tool** enables you to quickly create and set options for an attribute. An attribute is a named property of a class that defines the values that instances of the property can hold.

Use the **Attribute Tool** to create a new attribute, or modify an existing attribute. You can access the Attribute Tool from the following:

- class
- capsule
- class role
- capsule role
- interaction instance

Note: If you use the **Attribute Tool** to add an attribute, the model diagrams are not automatically updated; however, the **Model View** tab in the browser updates immediately and displays the new attribute. To update your model diagrams, you must select the attributes from the **Model View** tab in the browser, then drag them to the appropriate element in the model diagram.

Figure 17 Attribute Tool dialog



Name: NewAttribute1
Type:
Initial Value:
Visibility: public
Create any required dependencies
Class Scope
Constant
Generate: get method, set method
Equivalent Output
Insufficient Information For Output
C++ Language Specific Options
Constant Type: const, #define
Global
Initialization: constructor
Get Method: inline, const
Set Method: inline

Name

A name for the attribute. Use the name box to:

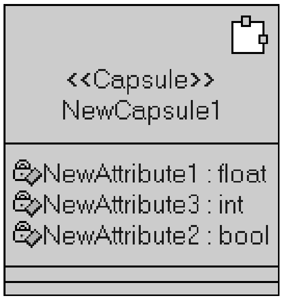
- Specify a name for a new attribute
- Select an existing attribute from the drop-down list
- Change the name of an existing attribute

Type

Attribute types can be classes or language-specific types. When the attribute is a data value, the type is defined as a language-specific type. You can enter the type directly in the **Type** box, or select a type from the drop-down list. Rational Rose RealTime displays the type opposite the attribute name and updates the information in the model (Figure 18).

See the **Equivalent output** box in the **Attribute Tool** dialog to view an approximation of the output for the attribute when it is of the selected type.

Figure 18 Display of Attributes and corresponding types

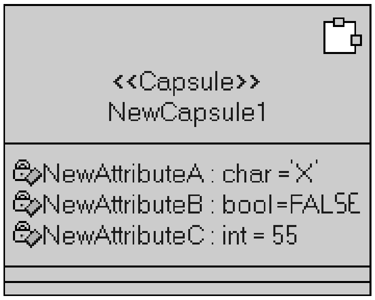


Initial Value

Assign an initial value to your class attribute. Type directly in the **Initial Value** box. Rational Rose RealTime displays the initial value opposite the attribute name and updates the information in the model (Figure 19). If you select **Constant**, you must specify a value in the **Initial Value** box.

See the **Equivalent output** box in the **Attribute Tool** dialog to view an approximation of the output when you specify a value for **Initial Value**.

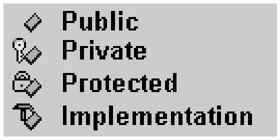
Figure 19 Display of Initial Values for Attributes



Visibility

Specify the type of visibility for an attribute. There are four types of visibility:

- **Public** - the attribute is visible to other classes.
- **Protected** - the attribute is visible only to subclasses and *friend* classes.
- **Private** - the attribute is not visible to other classes, except designated *friend* classes.
- **Implementation** - the attribute is never visible to other classes.



See the **Equivalent output** box in the **Attribute Tool** dialog to view an approximation of the output for the specified **Visibility**.

Create any required dependencies

Specifies whether you want dependencies created for this attribute. If you select this option, after you click **Save**, the **Dependencies** dialog displays. For additional information on the **Dependencies** dialog, see *Dependency Tool* on page 112.

Class Scope

Specify the class scope for the attribute. Selecting this option means that there is a single instance of the attribute for all instances of the class (for example, a static member in C++). If this options is not selected, then each instance of the class has a separate attribute instance.

See to the **Equivalent output** box in the **Attribute Tool** dialog to view an approximation of the output when selecting **Class Scope**.

Constant

Specifies whether the attribute is a constant. This means that this attribute cannot take on a new value. If you select **Constant**, you must specify a value in the **Initial Value** box.

See the **Equivalent output** box in the **Attribute Tool** dialog to view an approximation of the output when selecting **Constant**.

get Method

Generates an operation to retrieve the value of this attribute.

See the **Expected output** box in the **Attribute Tool** dialog to view an approximation of the output when selecting **get method**.

set method

Generates an operation to assign a value to this attribute.

See the **Expected output** box in the **Attribute Tool** dialog to view an approximation of the output when selecting **set method**.

Equivalent Output

Displays a "best" approximation of the expected output for the options selected in the **Attribute Tool** dialog.

For a new attribute, the **Equivalent Output** box displays the following:

```
## Insufficient Information For Output ##
```

This means that there is insufficient information specified for the selected or new attribute.

To view all of the **Equivalent Output** code in a single pane for the selected attribute, right-click inside the **Equivalent Output** box. The pop-up window resizes to contain all of the code from the **Equivalent Output** box.

Note: You can copy code from the **Equivalent Output** box; however, the code within this box is only an approximation and may not represent the precise code segment. Therefore, when copying from the Equivalent Output box, use caution.

More/Less button

Click this button to expand or collapse the advanced attribute area. This area contains additional language-specific options to set for an attribute.

New

Click this button to add a new attribute. Ensure that you click **Save** before you click **New** to save any changes to the previous attribute.

Save

Saves the settings for the selected attribute.

Language Specific Options

The advanced area contains options for language-specific features. For information on these options, refer to the Online Help.

Operation Tool

The **Operation Tool** enables you to quickly create and set options for an operation. An operation is the implementation of a service requested from any object of the class that affects its behavior.

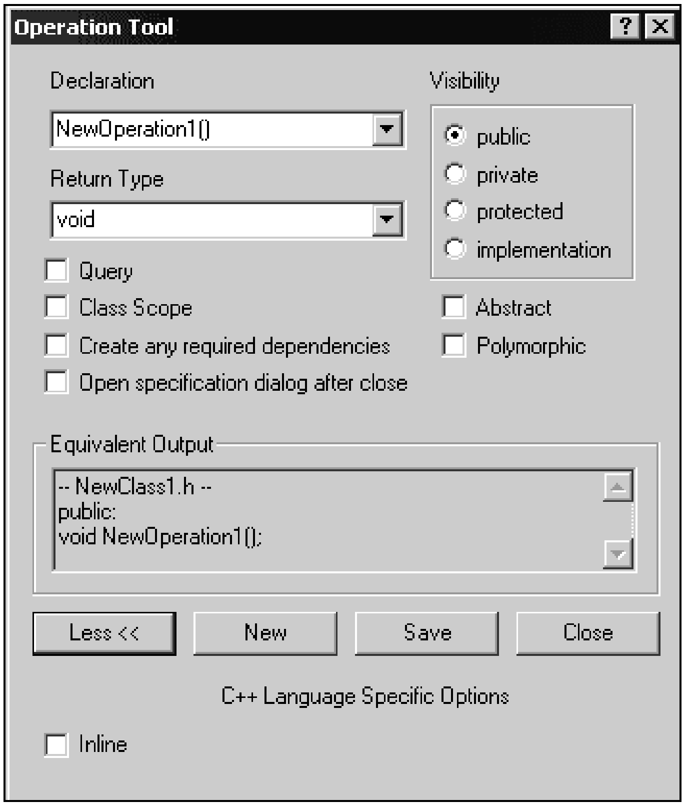
Use the **Operation Tool** to create a new attribute, or modify an existing attribute. You can access the **Operation Tool** from the following:

- class
- capsule
- class role
- capsule role
- interaction instance

Note: If you use the **Operation Tool** to add an operation, the model diagrams are not automatically updated; however, the **Model View** tab in the browser updates immediately and displays the new operation. To update your model diagrams, you must select the new operations from the **Model View** tab in the browser, then drag them to the appropriate element in the model diagram.

Note: The **Operation Tool** does not handle pointers to functions or templates; only simple parameters.

Figure 20 Operation Tool dialog



Declaration: NewOperation1()

Return Type: void

Visibility: [✓ public] private protected implementation

```
-- NewClass1.h --
public:
void NewOperation1();
```

Declaration

Specifies the name and any parameters and functions for the operation. Each parameter must have a valid type and name; a value is optional. For example, the following declarations are valid:

```
myDec1(int foo)
```

and

```
myDec2(int foo = 55, bool barA = False)
```

and

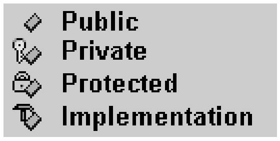
```
myDec3(RTTime t)
```

Note: If you specify a variable type and variable name that is a class in a parameter, you must also select **Create any required dependencies**. After you click **Save** on the **Operation Tool** dialog, the **Dependency** dialog displays and you must generate the required dependencies.

Visibility

Specify the type of visibility for an operation. There are four types of visibility for operations:

- **Public** - the operation is visible to other classes.
- **Protected** - the operation is visible only to subclasses and *friend* classes.
- **Private** - the operation is not visible to other classes, except designated *friend* classes.
- **Implementation** - the operation is never visible to other classes.



See the **Equivalent output** box in the **Operation Tool** dialog to view an approximation of the output for the specified **Visibility**.

Return Type

For operations that are functions, set this field to identify the class or type of the function's result. You can specify a class name that does not yet exist in your model; however, Clicking Save and closing the Operation Tool does not automatically create the class.

See the **Equivalent output** box in the **Operation Tool** dialog to view an approximation of the output when specifying a **Return Type**.

Query

When selected, it specifies that the operation is read-only and does not modify the object's state.

Note: If the **Return Type** is set to a class, the **Create any required dependencies** box must be set so that the **Dependency Tool** may generate the required dependencies

See the **Equivalent output** box in the **Operation Tool** dialog to view an approximation of the output when selecting **Query**.

Class Scope

Specifies class scope for the operation. Selecting this option means that the operation behaves the same way regardless of the state of any individual object in the class. Otherwise, the operation operates on individual class instances because its calculations are based on the object state, or because it modifies the object state.

See the **Equivalent output** box in the **Operation Tool** dialog to view an approximation of the output when selecting **Class Scope**.

Abstract

When selected, it indicates that the operation is an abstract definition that should be overridden by specific implementations in subclasses.

See the **Equivalent output** box in the **Operation Tool** dialog to view an approximation of the output when selecting **Abstract**.

Create any required dependencies

Specifies whether you want dependencies created for this operation. If you select this option, after you click **Save**, the **Dependencies** dialog displays. If you specified a class variable type and name, such as (RTTime t) in the **Declaration** box, or a **Return Type** that is a class, you must select this option. After you click **Save** on the **Operation Tool** dialog, the **Dependency** dialog displays and you must generate the required dependencies.

For additional information on the **Dependencies** dialog, see *Dependency Tool* on page 112.

Polymorphic

When selected, it indicates that the operation should be inherited by all subclasses.

See the **Equivalent output** box in the **Operation Tool** dialog to view an approximation of the output when selecting **Polymorphic**.

Open Specification dialog after close

Informs the Operation tool to open the **Specification** dialog of the current operation after closing the tool. A **Specification** dialog opens for each operation that you set this option.

Note: When opening the **Operation Specification** dialog, the tab that you selected last is the tab that displays.

Equivalent Output

Displays a "best" approximation of the expected output for the options selected in the **Attribute Tool** dialog.

For a new operation, the **Equivalent Output** box displays the following:

```
## Insufficient Information For Output ##
```

This means that there is insufficient information specified for the selected or new operation.

To view all of the **Equivalent Output** code in a single pane for the selected operation, right-click inside the **Equivalent Output** box. The pop-up window resizes to contain all of the code from the **Equivalent Output** box.

Note: You can copy code from the **Equivalent Output** box; however, the code within this box is only an approximation and may not represent the precise code segment. Therefore, when copying from the **Equivalent Output** box, use caution.

More/Less button

Click this button to expand or collapse the advanced operation area. This area contains additional language-specific options to set for an operation.

New

Click this button to add a new operation. Ensure that you click **Save** before you click **New** to save any changes made to the previous operation.

Save

Saves the settings for the selected attribute.

Language Specific Options

The advanced area contains options for language-specific features. For information on these options, refer to the Online Help.

Dependency Tool

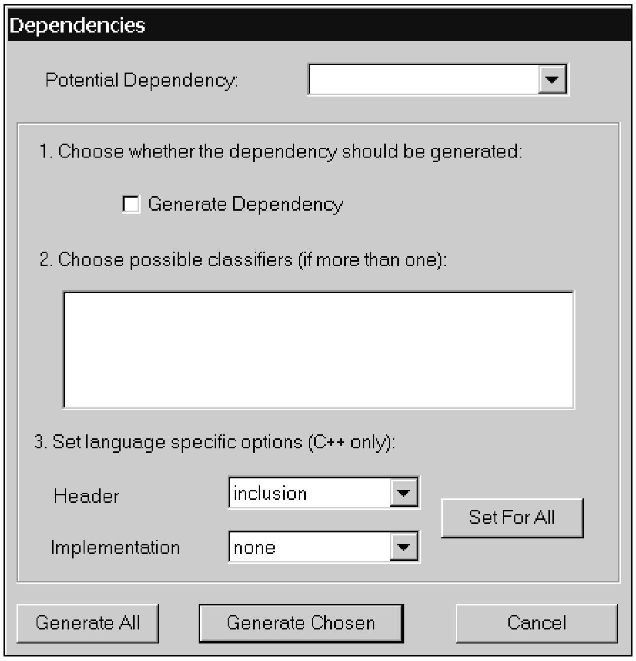
The Dependencies dialog enables you to create any required dependencies for the following:

- attributes (using the **Attribute Tool**)
- operations (using the **Operation Tool**)

A dependency is a relationship that indicates that a change to one thing may affect another thing that uses it.

You can access the Dependencies dialog by selecting **Create any required dependencies** and then clicking **Save** on the **Attribute Tool** dialog or the **Operation Tool** dialog.

Figure 21 Dependencies dialog


Dependencies
Potential Dependency:
1. Choose whether the dependency should be generated:
Generate Dependency
2. Choose possible classifiers (if more than one):
3. Set language specific options (C++ only):
Header
inclusion
Implementation
none
Set For All
Generate All
Generate Chosen
Cancel

Potential Dependency

Specifies the name of a possible dependency for the selected operation or attribute. The drop-down list contains the names of the classes for which a dependency can be created.

1. Choose whether the dependency should be generated:

Generate Dependency

Generates the dependency selected from in the **Potential Dependency** box. If this option is not selected, a dependency will not be generated for the selected class. Use this option to exclude the generation of dependencies, for example, select only those potential dependencies for which you do not want a dependency, and then click this option for each potential dependency so that it is set, the click **Generate Chosen**.

3. Choose possible classifiers (if more than one):

Specifies the name of the classifier for the selected dependency.

3. Set language specific options:

Header

Specifies the directive that is generated in the header file. When the C++ generator produces code for an element (the client) that uses another element (the supplier), the C++ generator produces either an **include** directive referencing the file that contains the supplier class, or a **forward reference** to the supplier.

You can configure which directive (**include statement** or **forward reference**) is generated in the header file (.h).

Implementation

Specifies the directive that is generated in the implementation file. When the C++ generator produces code for an element (the client) that uses another element (the supplier), the C++ generator produces either an **include** directive referencing the file that contains the supplier class, or a **forward reference** to the supplier.

You can configure which directive (**include statement, forward reference**, or **none**) is generated in the implementation file (.cpp).

Set For ALL

Sets the specified **Header** and **Implementation** options for all dependencies, regardless of what was previously specified for individual potential dependencies.

Generate All

Generates all dependencies, regardless of whether the **Generate Dependency** option (Step 1) is set for each of the potential dependencies.

Generate Chosen

Generates only the potential dependencies in the **Dependencies** dialog for which the **Generate Dependency** option (Step 1) was set.

Close

Closes this dialog without generating any dependencies, regardless of the options you selected.

Other Application Windows 4

Contents

This chapter is organized as follows:

This chapter describes other application windows, including the Description Window, which contains the Documentation Pane and the Code Pane, and the Output Window.

Description Window

The Description window contains the Documentation Pane and the Code Pane. You can toggle between the two by clicking their tabs.

Displaying the Description Window

By default, the Description window is closed. To view the window, select **View > Description**.

Only one Description window can be open at a time, but as you select different items, the window is updated accordingly. If you select an item that has no documentation or code associated with it, or you select multiple items, or you do not have an item selected, you are notified accordingly.

When the window is first displayed, it is docked to the bottom left corner. To move the window, click and drag on the border. The window outline indicates the window state: a thin, crisp line indicates the window is docked, while a thicker, hashmark-type border indicates it is floating.

Characteristics unique to the window state (docked or floating) are discussed below:

Docked

- The window can be moved within the dockable region of the application.
- The size can be changed using the splitter bars.
- The title can be displayed through a tool tip (simply place your cursor anywhere in the window). There is no title available when the window is docked.
- The window can be docked at any time.

Floating

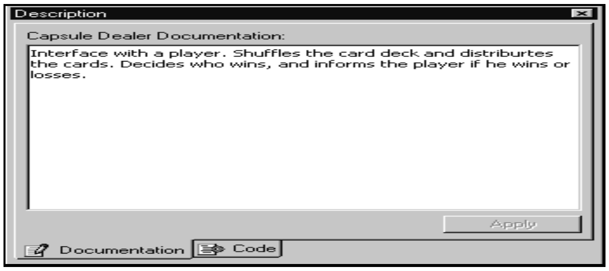
- The window can be moved to any location, and is always displayed on top of the diagram.
- Size can be changed via click and drag along the border in a vertical or horizontal direction.

The window title displays the description. The static text displays the name of the element who's code or documentation you are viewing.

Documentation Pane

You can use the Documentation pane (Figure 22) to edit or view the documentation associated with the currently selected model element. Scroll bars are added when necessary and word wrap is employed.

Figure 22 Documentation pane

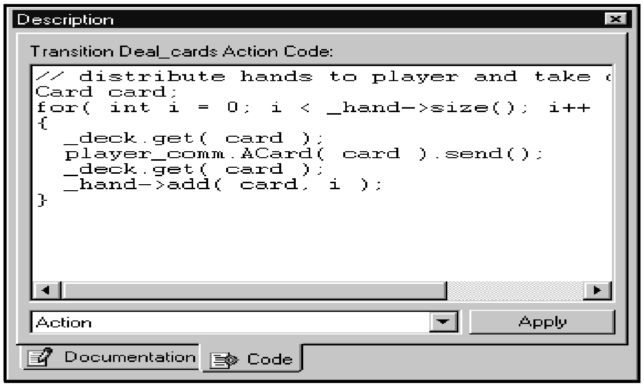


Interface with a player. Shuffles the card deck and distriburtes the cards. Decides who wins, and informs the player if he wins or losses.

Code Pane

You can use the Code pane (Figure 23) to edit or view the code associated with the currently selected model element. Scroll bars are added when necessary and word wrap is employed.

Figure 23 Code pane



```
// distribute hands to player and take c
Card card;
for( int i = 0; i < _hand->size(); i++
{
   _deck.get( card );
   player_comm.ACard( card ).send();
   _deck.get( card );
   _hand->add( card, i );
}
```

Pulldown menu

Using the pulldown menu, you can move between different sections of code, for example, between entry and exit code for a state. You can also add a trigger to a transition.

Popup menu

Using the popup menu, you can

- import and export files containing code
- print the code
- select individual words, lines, or all of the code you are viewing
- specify the font
- use the Search feature to Find and Replace
- launch an external editor

Note: If you use an external editor that requires a console terminal, you must specify an application, such as **`xterm`**, that provides the terminal, followed by the editor command itself.

Note: Example on Solaris: /usr/openwin/bin/xterm -e /bin/vi

Note: Example on HPUX: /usr/bin/X11/xterm -e /bin/vi

Adding Documentation to Model Elements

All model elements can have documentation associated with them.

To add documentation to a model element, you can use the Documentation window or follow these steps:

1. Right-click on the model element in the model browser or in a diagram.
2. Select **Open Specification** from the selected item's menu.
3. Click on the General tab if it is not the tab currently displayed.
4. Enter the documentation for the element in the documentation area.
5. Close the Specification dialog by clicking **OK**.

For long, complex, or formatted documentation, you may want to link an external file (such as an MS Word document) to a model element. See *Inserting a Diagram into an MS Word Document* on page 373.

Note: If you add documentation from the Documentation pane, you must click Apply button for it to take effect.

Adding Code to Model Elements

All model elements can have code associated with them.

To add code to a model element, you can use the code window or

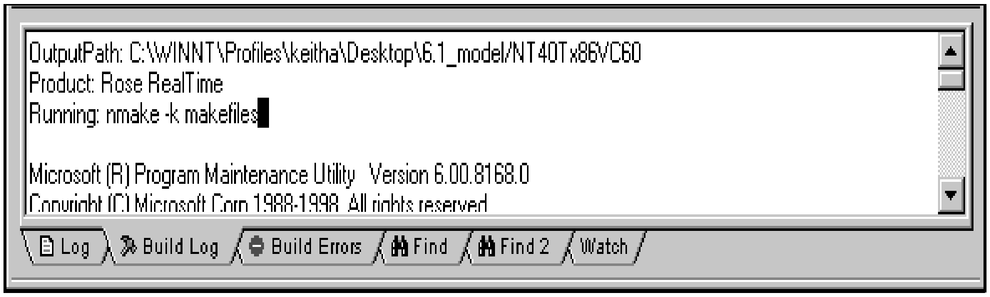
1. Right-click on the model element in the model browser or in a diagram.
2. Select **Open Specification** from the selected item's menu.
3. Click on the language-specific tab if it is not the tab currently displayed.
4. Enter the code.
5. Close the Specification dialog by clicking **OK**.

Output Window

The Output window (see Figure 24) is a dockable window that contains the following tabs:

- Log Tab
- Build Log tab
- Build Errors tab
- Find Tab
- Watch Tab (RTS only)

Figure 24 Output window



```
OutputPath: C:\WINNT\Profiles\keitha\Desktop\6.1_model/NT40Tx86VC60
Product: Rose RealTime
Running: nmake -k makefiles

Microsoft (R) Program Maintenance Utility   Version 6.00.8168.0
Copyright (C) Microsoft Corp 1988-1998. All rights reserved.
```

Log Tab

The Log tab is used by several commands to report progress, results, and errors. Messages posted to the log are usually prefixed with a time stamp.

To display the Log tab, select **View > Output**. The application posts the messages to the log window regardless of whether it is displayed.

You can save the contents of the log window to a file or you can choose to automatically save messages to a file as they are posted. Both options are available from the popup menu.

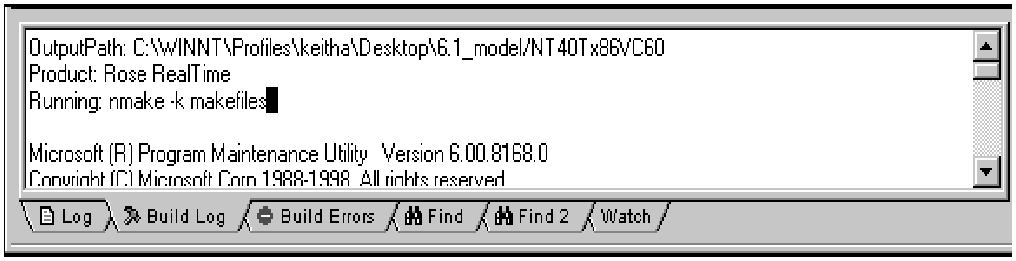
Double-clicking usually brings you to the error source.

Build Log tab

The Build Log tab stores the contents of the compilation and code generation log. Select **View > Output** and click the Build Log tab to open it. Compilation or code generation messages are posted to the Build Log tab regardless of whether it is visible.

You can save the contents of the Build log tab to a file. You can also choose to automatically save messages to a file as they are posted.

Figure 25 Build Log tab



```
OutputPath: C:\WINNT\Profiles\keitha\Desktop\6.1_model/NT40Tx86VC60
Product: Rose RealTime
Running: nmake -k makefiles

Microsoft (R) Program Maintenance Utility   Version 6.00.8168.0
Copyright (C) Microsoft Corp 1988-1998. All rights reserved.
```

The Build Log tab contains the raw output stream from the build. You can examine the contents of this window to get a context on any error message displayed in the build messages list.

Related Topics

Components

Related Tasks

Creating a Component

Assigning an Active Component

Starting a Build

Reviewing the Build Results

Build Errors tab

The Build Errors tab contains a parsed version of the output stream. It is important to review the Build Log tab because some errors cannot parsed by the error parser.

The Build Errors tab contains a Location column that gives the class/code segment name pair. The Context column provides the context of the problem. The Message column gives a description of the problem. These messages are taken directly from the compiler error stream and therefore reflect the accuracy of the compiler that you are using. Further, errors within your code segments may lead to errors being reported in system-generated files.

Double clicking on an error or warning in the Build Error tab brings you to the location in the model of the problem that caused the error or warning. See Common build errors for a short summary of common generic build errors.

Unknown compiler message stream

It is possible that the compiler being used reports errors in ways that are not understood by Rose RealTime. There are no standards for error reporting by compilers and linkers. Hence, the error parser is often targeted for a particular compiler and linker. If you are using an unsupported compiler, Rose RealTime will probably not be able to understand the error output from the parser and may inaccurately report errors. You have to rely on the raw output stream to see the direct output of the compiler, rather than going by the errors reported by the Build Errors tab.

Find Tab

The Find tab works in conjunction with the Find dialog. See *Find dialog* on page 87.

Watch Tab

Capsule instance attributes can be inspected at run-time and modified from the Watch tab of the Output window. The Watch tab has two columns: the name of the attribute and its value.

To add an attribute instance, or variable, to the watch window, open a state monitor and drag-and-drop the attribute from the Attributes folder into the watch window.

You can also edit the value of a variable by selecting the Value field then entering another value for the variable.

Refreshing the Watch Values

The watch values are refreshed when a message is received by the state of the capsule instance. If the state monitor the watch was created is closed, the watch value stops being updated. If the state monitor is closed, you can manually force an update of a watch value by right-clicking on the watch item and selecting **Refresh** from the popup menu.

Printing

5

Contents

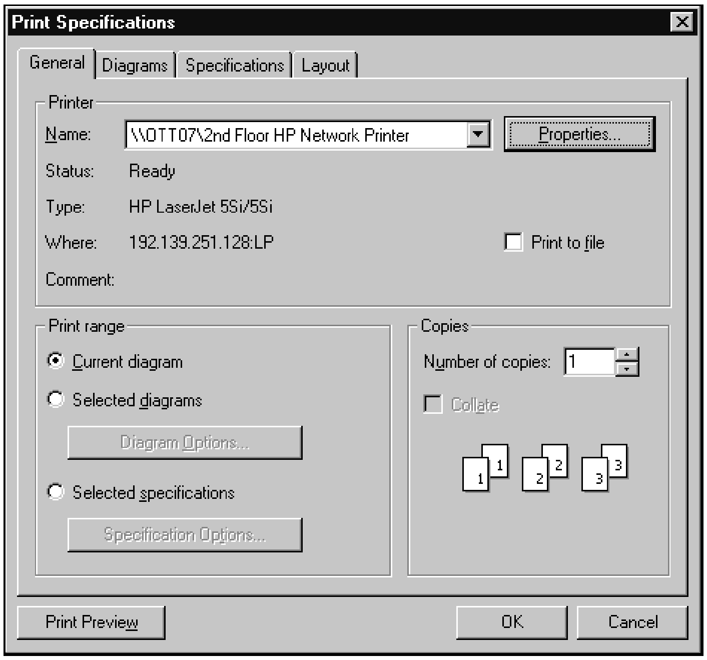
This chapter is organized as follows:

This chapter describes how to print from the application using the Print Specifications and Print Setup dialogs.

Print Specifications

The Print Specifications dialog lets you print diagrams. As well, you can adjust the parameters of diagrams you want to print, including size, orientation, and layout.

Figure 26 Print Specifications dialog



Name: \\OTT07\2nd Floor HP Network Printer
Status: Ready
Type: HP LaserJet 5Si/5Si
Where: 192.139.251.128:LP
Comment:
Print to file: unchecked
Print range: (●) Current diagram ( ) Selected diagrams ( ) Selected specifications
Number of copies: 1

The dialog has four tabs:

- General Tab
- Diagrams tab
- Specifications tab
- Layout Tab

All tabs contain the **Print Preview** button, which you can click to see how your diagram will appear before you route it to a printer.

General Tab

The General tab contains three fields:

- Printer field
- Print Range field
- Copies field

Printer field

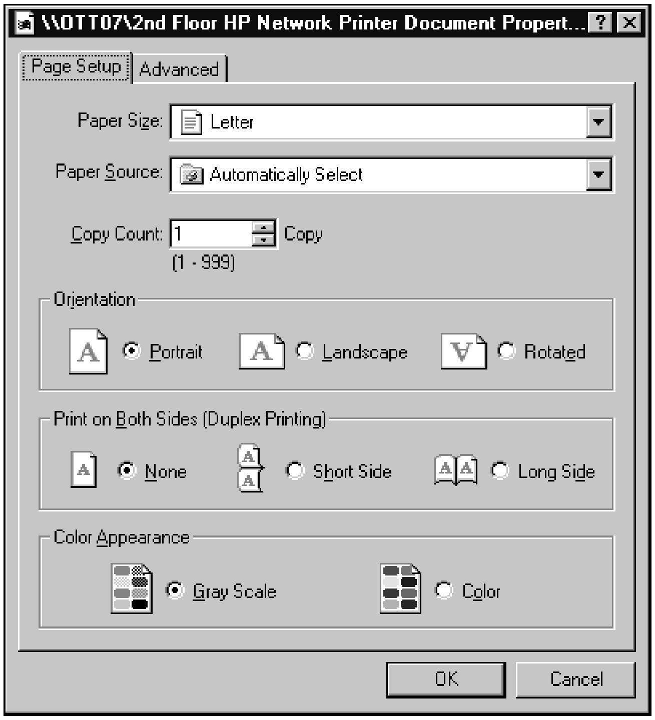
The Printer field lets you select the name of the printer you want to use from a drop-down menu. You can check the Print to File option if you want to print a diagram to a file, instead of routing it directly to a printer. You are prompted to specify a filename and location.

As well, there is a **Properties** button.

Properties dialog

Clicking the Properties button opens the Properties dialog, which contains two tabs: Page Setup and Advanced.

Figure 27 Properties dialog


\\OTT07\2nd Floor HP Network Printer Document Propert...

Page Setup | Advanced

Paper Size: Letter

Paper Source: Automatically Select

Copy Count: 1 Copy
(1 - 999)

Orientation: [✓ Portrait] Landscape Rotated

Print on Both Sides (Duplex Printing): [✓ None] Short Side Long Side

Color Appearance: [✓ Gray Scale] Color

The Page Setup tab lets you select:

- Paper Size
- Paper Source
- Copy Count
- Orientation
- Print on Both Sides
- Color Appearance

The Advanced tab lets you fine tune printing parameters, including Paper/Output, Graphic, and Document Options.

Print Range field

The Print Range field lets you select between the Current Diagram, Selected Diagrams, and Selected Specifications. Current Diagram is the default. Clicking Selected Diagrams and then Diagram Options opens the Diagrams tab. Clicking Selected Specifications and then Specification Options opens the Specifications tab.

Copies field

The Copies field lets you specify the number of copies you want to print, and whether you want multiple copies collated.

Diagrams tab

The Diagrams tab contains the options that you can choose from to generate a printout of one or more diagrams. It contains five fields: Use case diagrams, Class diagrams, Component diagrams, Deployment diagrams, and Interaction diagrams.

The first four fields contain the following buttons:

- Top Level - prints only the diagrams at the top level of the model.
- Entire Structure - prints all the diagrams.
- None - prints none of the diagrams.

Additionally, there is a checkbox, Include State Diagrams. The checkbox is not applicable unless you have chosen Top Level or Entire Structure in the Use Case or Class Diagrams fields.

The Interaction Diagrams List Control lets you select each object message or message trace diagram containing objects whose specifications you want to print. The All button selects all object and interaction diagrams in the list. The None button deselects all object and interaction diagrams in the list.

Specifications tab

The Specifications tab contains the options that you can choose from to generate a printout of one or more specifications. It contains six fields: Use case specifications, Class specifications, Component specifications, Deployment specifications, Options, and Interaction specifications.

The first three fields contain the following buttons:

- Current - prints only the specifications for the current diagram.
- Entire Structure - prints all the diagrams.
- None - prints none of the diagrams.

The Options field contains the following checkboxes:

- Selected classes only - is only applicable when you have chosen Current or Entire structure in the Class diagrams field. When you check this box, only the specifications for those classes that are currently surrounded by selection handles in the indicated class diagrams are printed.
- Operation specifications - is only applicable when you have chosen Current or Entire structure in the Class diagrams field. When you check this box, only the operation specifications that are associated with the classes in the indicated diagrams are printed.
- State specifications - is only applicable when you have chosen Current or Entire structure in the Class diagrams field. When you check this box, all the state-transition specifications for all the state diagrams that are associated with the classes in the indicated diagrams are printed.

  Note: This checkbox controls only the printing of use case specifications. You can print the associated operation and state-transition specifications by checking the Operations Specifications and State Transitions boxes.
- Selected associations only - is only applicable when you have chosen Current or Entire structure in the Class diagrams field. When you check this box, only the operation specifications for those associations with the classes in the indicated diagrams are printed.
- Selected components only - is applicable only when you have chosen Current or Entire structure in this field. When you check this box, only the specifications for those components that are currently surrounded by selection handles in the indicated component diagrams are printed.
- Selected devices only - is only applicable when you have chosen Current or Entire structure in the Deployment specifications field. When you check this box, only the specifications for the selected devices are printed.
- Selected processors only - is only applicable when you have chosen Current or Entire structure in the Deployment specifications field. When you check this box, only the specifications for the selected processors are printed.

The Interaction Diagrams List Control lets you select each object message or message trace diagram containing objects whose specifications you want to print. The All button selects all object and interaction diagrams in the list. The None button deselects all object and interaction diagrams in the list.

Layout Tab

The Layout tab contains the options that you can choose from to change the position and size of the diagrams you want to print. If your print job is larger than the available paper, you can tile your work so that it is spread across several pieces of paper. Assemble the separate pages to create the whole image.

The Layout tab contains two fields: Positioning and Options.

Positioning field

The Positioning field contains the options that you can choose from to change the size of the diagrams you want to print.

- As In Diagram - prints diagram as you see it on screen.
- Fit To Page - fits each diagram to one page.
- Tile - enables the Options field.

Options field

The Options field contains the following options:

- Overlap - lets you set the percentage of the images on each tile overlap on adjacent tiles.
- Print Crop Marks - lets you align tiled printouts.
- Preserve Aspect Ratio- lets you maintain the diagram's proportions.

Related Topics

File menu

Print Setup

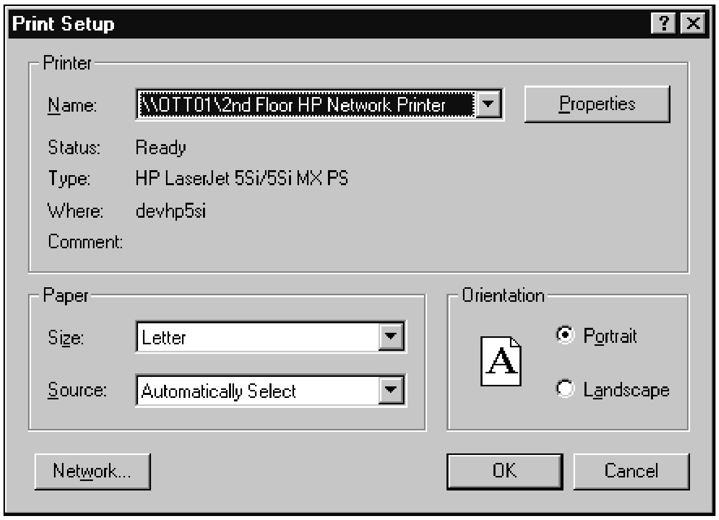
Related Tasks

Inserting a Diagram into an MS Word Document

Print Setup

The Print Setup dialog lets you set up print options, generally. The dialog contains three areas: Printer, Paper, and Orientation. As well, there is a Network button. Clicking this button opens the Connect to Printer dialog, which lists shared printers on the network. Click on a particular printer to route your print jobs to it.

Figure 28 Print Setup dialog


Print Setup

Printer

Name: \\OTT01\2nd Floor HP Network Printer

Status: Ready
Type: HP LaserJet 5Si/5Si MX PS
Where: devhp5si
Comment:

Paper

Size: Letter
Source: Automatically Select

Orientation

[✓ Portrait] Landscape

Printer field

The Printer field lets you select the name of the printer you want to use from a drop-down menu. You can also click the **Properties** button, which opens the Properties dialog.

Paper field

The Paper field lets you specify the size and source of the paper you want to use to print.

Orientation field

The Orientation field lets you choose between Portrait and Landscape orientations.

Related Topics

Printing

File menu

Opening and Saving Models 6

Contents

This chapter is organized as follows:

Unique Ids

Unique ids are unique internal names associated with model elements. They are used internally by Rational Rose RealTime, and not all model elements require unique ids. Rational Rose RealTime includes a feature that helps Model Integrator by generating unique ids for those model elements that would otherwise not require them, for internal use. For Model Integrator, an element with a unique id is easier to merge.

RRTEI users will find traceability easier when they set this option. Unique ids improve the traceability of model elements of other tool integrations that use RRTEI.

It is necessary to plan and choose when to incorporate the new unique ids into the project model since virtually all controlled units will be modified implicitly. Additionally, the generated new ids are dependent on time and location. For example, generating unique ids for a given model at different times, or on different machines, produces different ids.

The following model elements do not have unique ids, unless you set this option:

- Protocol In Signals ()
- Protocol Out Signals ()
- States (CompositeState)
- Capsule Roles (CapsuleRole)
- Ports (Port)
- Port Roles (PortRole)
- Capsule Structure diagram (CapsuleStructure)

- Classifier Role (ClassifierRole)
- Transitions (Transition)
- Junction Point (JunctionPoint)
- Choice Point (ChoicePoint)
- Connectors (Connector)
- (Guards)
- (Events)
- (EventGuards)
- Parameters ()
- Element hyperlinks (ExternalDocument)

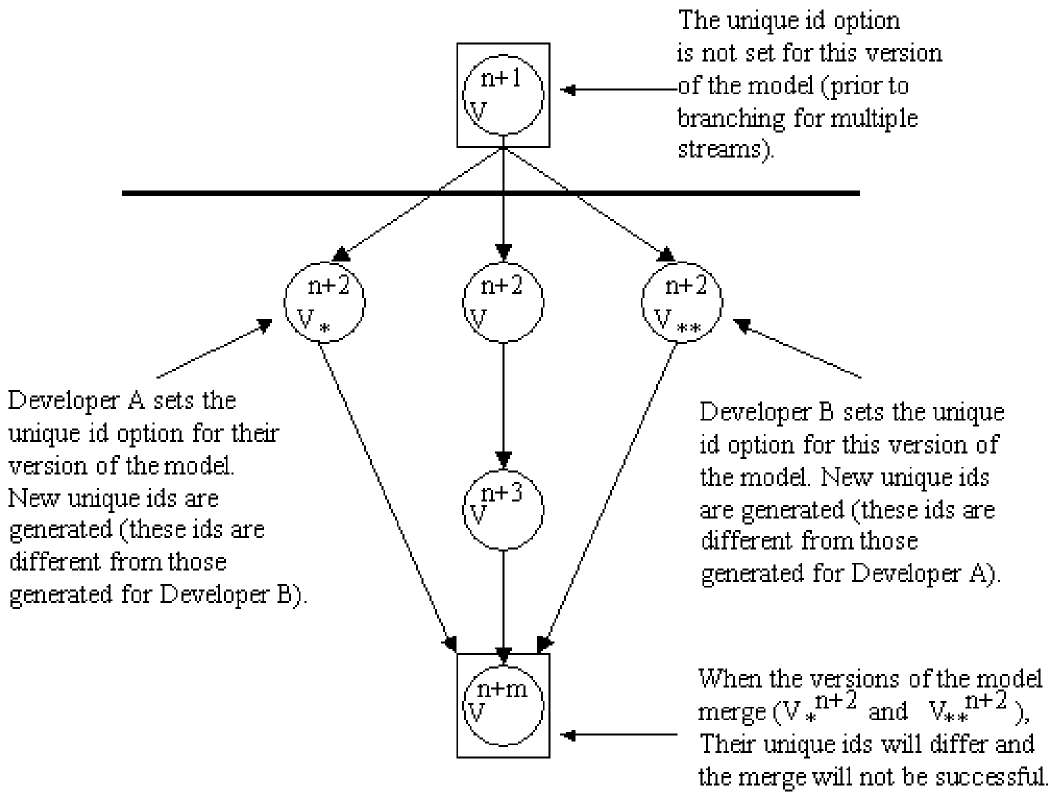
Caution: We strongly recommend any team involved in parallel development use this option.

Note: *Setting this option creates unique ids for model elements that currently do not have them. This typically affects most of the model, so you will be prompted to check out those parts when setting this option.*

When saving the model, the size of the affected file increases by approximately 20%, and the time to load the model also increases.

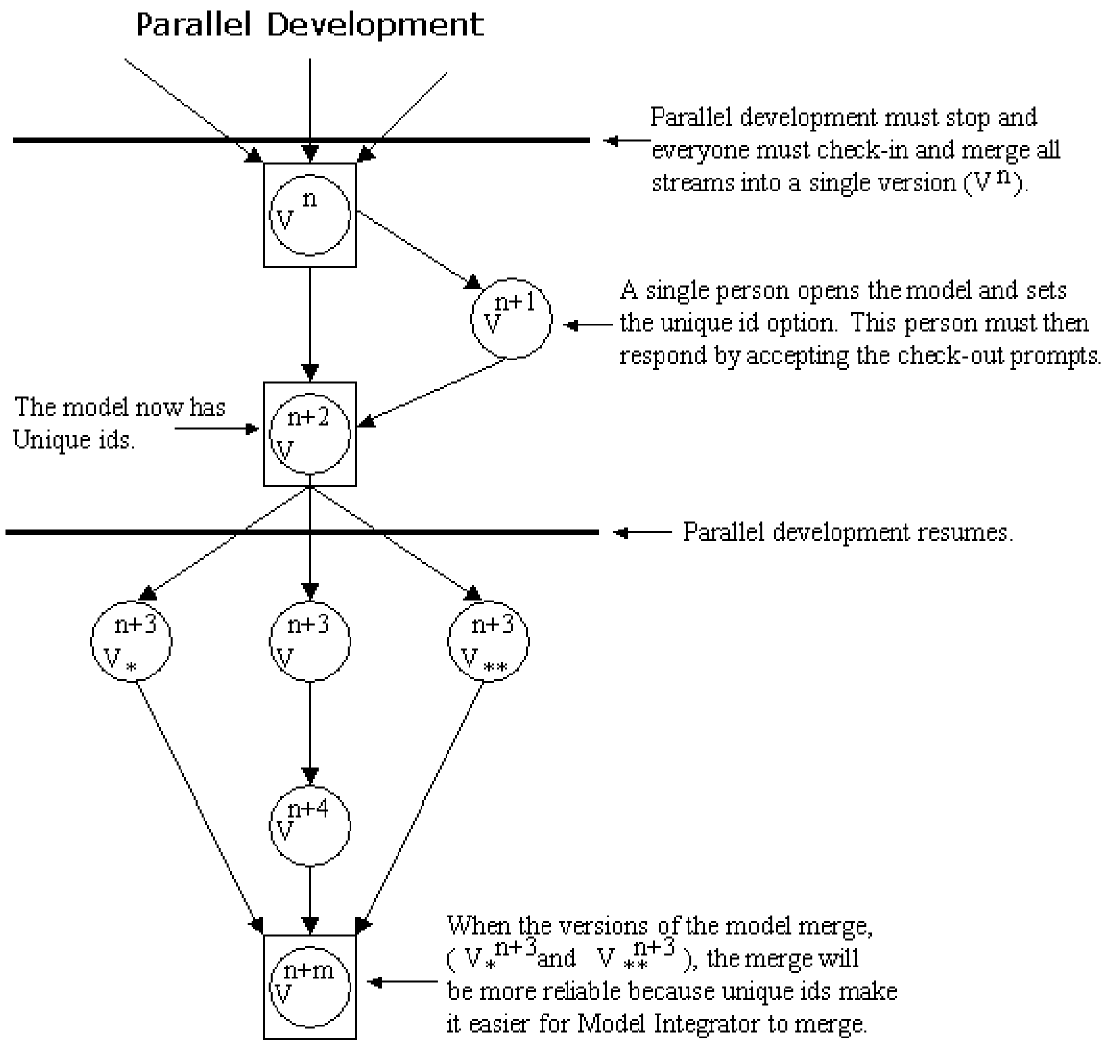
Caution: Do not set this option for multiple streams as shown in Figure 29; otherwise, objects with similar characteristics will be treated differently since their unique id's will differ.

Figure 29 Incorrect Merge Scenario


The unique id option is not set for this version of the model (prior to branching for multiple streams).
n+1 V
n+2 V*
n+2 V
n+2 V**
Developer A sets the unique id option for their version of the model. New unique ids are generated (these ids are different from those generated for Developer B).
Developer B sets the unique id option for this version of the model. New unique ids are generated (these ids are different from those generated for Developer A).
n+3 V
n+m V
When the versions of the model merge (V*n+2 and V**n+2), Their unique ids will differ and the merge will not be successful.

An example of when to set this option is shown in Figure 30.

Figure 30 A Correct Merge Scenario


Parallel Development
Parallel development must stop and everyone must check-in and merge all streams into a single version (V n).
V n
V n+1
A single person opens the model and sets the unique id option. This person must then respond by accepting the check-out prompts.
The model now has Unique ids.
V n+2
Parallel development resumes.
V* n+3
V n+3
V** n+3
V n+4
V n+m
When the versions of the model merge, (V* n+3 and V** n+3), the merge will be more reliable because unique ids make it easier for Model Integrator to merge.

Note: *This option must be set prior to branching.*

For information on how to enable the Unique ids, see *Model Specification* on page 135.

To clear the unique id option, follow the same procedure in Figure 30.

Caution: *If you clear this option, your merge results will not be as reliable.*

Opening Models

To open an existing Rose RealTime, Rose, or ObjecTime Developer model, click on the Open Model icon on the toolbar or select **File > Open**.

A dialog appears prompting for the model file name. You can select from among different types of models to open through this dialog, including: Rose RealTime models (.rtmdl), Rose models (.mdl) and ObjecTime Developer models stored as linear form (.lf).

Note: Opening a model discards any existing model that you currently have open. The tool prompts you to save changes first.

Note: In the NT version of the toolset, typing `%ROSERT_HOME%` in the file name takes you to the directory that the environment variable holds. Use the same % notation on Unix to specify environment variables.

Model Specification

A Model Specification enables you to display and modify the properties of the top level element.

To display a Model Specification, right-click on the top level element and choose **Open Specification**.

Specification Content

The Model Specification contains the following tabs:

- General tab
- Source Control tab
- Files tab
- Unit Information tab

General tab

Name

This field identifies the name of the model.

Generate unique identifiers for all elements

When this checkbox is selected, unique identifiers are generated for all elements in the model.

Caution: *Before setting this option, ensure that you have reviewed Unique Ids* on page 131.

Documentation

This field contains information on this model.

Source Control tab

The Source Control tab provides options for interacting with a source control / Configuration Management (CM) system.

Files tab

Provides a list of referenced files. The files list popup menu allows you to insert and delete references to files or URLs.

You can link external files to models for documentation purposes.

Unit Information tab

File Name

This field lists the file name of the model.

Owned by model

This field indicates whether this model is owned by another model.

Under source control

This field

indicates whether this model has been added to source control.

A Workspace

A workspace contains basic configuration information for working with a model. This information includes the name of model being worked on, whether source control is enabled, and settings related to how source control and file management will behave when editing the model.

The workspace information is stored in a separate file (a .rtwks file). When a model is opened and a workspace file of the same name exists in the directory, the toolset prompts you to open the workspace instead. If you will be regularly working on a model, it is recommended that you open the workspace corresponding to that model rather than opening the model itself.

The following settings are stored in a workspace:

- the model being worked on
- the source control settings for this model as specified in the Specification dialog for the model
- file management settings

If a model gets renamed, a workspace file that refers to the old model name will not open correctly. You can edit the workspace file directly and change the path name information, or open the model file without the workspace and do a save to create a new workspace.

User-specific Working Environment Settings (.rtusr and .rtwks)

Rose RealTime preserves user-specific working environment settings between toolset sessions. The user-specific working environment consists of:

- options specified in the **Tools > Options** dialog (for example, font size, default label filtering)
- open windows, including their size and position
- active component
- active component instances
- target observability settings such as probes, monitors, inject messages and watch variables

All of these settings are user-specific and are not intended to be shared between users. Settings not related to target observability are saved in a .rtusr file with the same root name as the current workspace. Target observability settings are saved in a .rtto file with the same root name as the current workspace.

These files are saved whenever the current workspace is either saved or closed.

Opening Models from ObjecTime Developer 5.2.1

Rose RealTime can only import Linear Form files from ObjecTime Developer 5.2.1. Other kinds of files, such as binary .update or .context files cannot be imported directly into Rose RealTime.

Note: ObjecTime 5.2.1 users must apply a patch to their toolset in order to export models from ObjecTime that can be read by Rose RealTime. See the ObjecTime support website.

To open an ObjecTime Developer 5.2.1 model:

1. The ObjecTime Developer project file must be saved as a Linear Form file (.lf)
2. To open an ObjecTime Developer model from Rose RealTime, select **File > Open** and choose **Linear Form (.lf)** from the **Files of Type** pull-down menu.
3. Select the file to open and click **Open**.

Files from versions of ObjecTime older than ObjecTime Developer 5.2 will have to be opened in ObjecTime Developer 5.2 and saved as project files first.

Note: Opening a new model discards any existing model that you have. The tool prompts you to save changes first.

Importing requirements

Requirements captured in ObjecTime Developer Models can be converted through a requirements-specific patch for 5.2 and 5.2.1. An HTML file is generated that contains the actual requirements from the OTD models. Links to these requirements are converted when the actual model is imported into Rose RT. The HTML requirements file is stored outside of the Rose RealTime toolset. Place the file in your configuration management library for storage purposes.

See the *ObjecTime Developer Conversion Guide* for information on converting from ObjecTime Developer.

Limitations and Restrictions

When an ObjecTime Developer model is opened in Rose RealTime, the following elements may not be converted:

- dependencies - the dependencies list for classes in ObjecTime Developer is not converted. Dependencies must be recreated using the **Build > Add Class Dependencies...** command. This runs a script that checks the model elements for dependencies and adds them. It does not, however, find references that exist only in detailed code.

Opening Models from Rational Rose

Before starting

Rose RealTime can open files saved with Rational Rose 98 and 98i (.mdl files).

Fixing a Model

When importing a model from Rose 98/98i into Rose RealTime, you are encouraged to resolve any model errors in Rose98 (**Tools > Check Model)** before trying to import the model. In particular it is important to fix unresolved references. In general, Rose98 is not concerned as much about unresolved references; however, they are very important in Rose RealTime as they can result in incomplete code generation and compilation errors.

Tasks

To open a Rational Rose model:

1 To open a Rational Rose model from Rose RealTime: Select **File > Open** and choose **Rose Model (.mdl)** from the **Files of Type** pull-down menu.

2 Select the file to open and click **Open**.

Files from Rose versions older than Rose 98 have to be opened in Rose 98 and saved first.

Note: Opening a new model discards any existing model that you have. The tool prompts you to save changes first.

Import Log Messages

The following messages may appear in the Log after a Rose98 model as been imported.

Message: Warning: Renamed elementClass "oldElementName" to "newElementName".

Description: A loaded model element has been renamed to conform with Rose RealTime's naming requirements. Double-clicking on the warning in the log may (or may not) display the renamed element.

Message: Error: Unresolved reference from... to... by....

Description: The toolset was unable to resolve a reference between two model elements. This is usually the result of loading an incomplete model, for instance when the user has updated only part of a model from CM. The rest of the model needs to be loaded in order for the reference to be resolved. However, in some instances (where toolset stability is an issue) the unresolved model element is removed from the model. If this is the case, the deletion is also recorded in the log window.

Message: Error: Error reading file fileName at line lineNumber or Error message detail.

Description: The error message detail may contain validation errors originating from the internal meta-model, which are not covered here. Possible error message details that originate from the petal reader are listed below.

Message: Invalid syntax.

Description: The file contents cannot be read by the toolset. The user should send the file to customer support with a description of what they were doing when the file was created.

Example

Imported a Rose98 model, made some changes to the Component View, now the file won't reload in RoseRT.

Limitations and Restrictions

When a Rose model is opened in Rose RealTime, the following elements are not converted:

- State diagrams
- Importing Rose98 models containing controllable units is not supported

  If the Rose98 model file contains controllable units. The user should export the model from Rose98 into a single .ptl petal file (**File > Export Model**) which can then be opened with Rose RealTime (**File > Open**, and select **All Files...** in the combo box to display .ptl files.)
- Three-tier class diagrams are not supported in Rose RealTime.

  The Rose98 model file contains a three-tier class diagram, which are not supported in Rose RealTime. The user should create a copy of the Rose98 model that does not contain a three-tier diagram to import into Rose RealTime.

Importing Rational Rose Generated Code

Source code that has been generated from a Rose model and has been edited within the preserved regions may be imported.

To import Rose generated code:

1. Verify that the Rose .mdl file is not newer than the generated code. If so, regenerate the code.
2. Open the Rose model (see *Opening Models from Rational Rose* on page 139).
3. Select **Tools > Import Code**.

   If code was generated from this model using Rational Rose and the model was saved after the code generation was performed, a "Rose Code Import" window is displayed. Otherwise, a "There are no cpp or h files available for import" message is displayed.

   The Rose Code Import Window lists all the .cpp and .h files that were generated from the model and lets you select all or a subset of the files. It also displays the classes that will be affected by each file that is selected. After a file has been imported it will not be listed if code importation is repeated.
4. After you have completed importation and are satisfied with the results, save the model.

Limitations and Restrictions

- No action is taken on empty preserved regions. As a result, constructors, destructors, and operators that are generated by Rose, which have empty preserved regions, are be added to the model.
- Use of the Code Name properties for classes and operations may cause inconsistent naming in the generated code. The inconsistencies may cause compile time errors that can be resolved manually.

Use Case Diagrams 7

Contents

This chapter is organized as follows:

Creating a Use Case Diagram

Use case diagrams are created in the Use Case View of the model browser. A **Main** use case diagram is always present in the Use Case view. The Main use case diagram should be used to describe the relationships between the primary actors and use cases in the system. Other diagrams can be created as required.

To edit the Main use case diagram:

1 Double-click on the Main diagram in the Use Case View package in the Model browser.

The Use Case diagram editor appears (see *Using the Use Case Diagram Editor* on page 144).

2 Place actors and use cases in the diagram by dragging them from the model browser, or by using the tools in the Use Case Diagram Toolbox.

3 Draw relationships among actors and use cases using the toolbox.

To create a new use case diagram:

1 Right-click on the Use Case View package (or any sub-package) in the model browser.

2 Select **New > Use Case Diagram** from the popup menu.

3 Enter the name of the use case diagram.

Using the Use Case Diagram Editor

Use case diagrams present a high-level view of how a system is used as seen from an outsider′s (or actor′s) perspective. These diagrams depict system behavior (also known as use cases). A use case diagram may depict all or some of the use cases of a system.

A use case diagram can contain:

- actors ("things" outside the system)
- use cases (system boundaries identifying what the system   should do)
- interactions or relationships between actors and use cases in the system including associations and generalizations

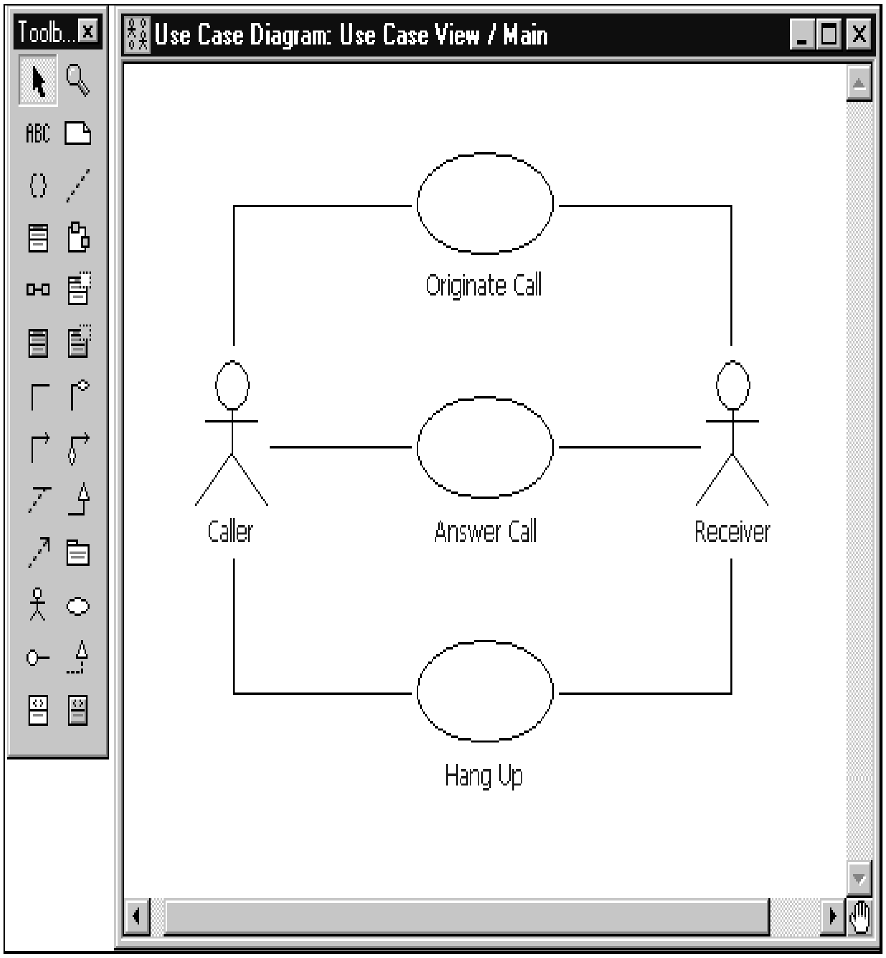
Use case diagrams can be used during analysis to capture the system requirements and understand how the system should work.

The use case diagram editor is used to create a diagram showing use cases and the relationships among use cases, actors and classes. The use case diagram consists of two parts:

- the diagram area
- the Use Case Diagram Toolbox

The window title bar shows the full name of the class diagram.

Figure 31 Use case diagram editor



Usage Tips

Typically, you add a set of related use cases and actors to the diagram. Then, draw the relationships among use cases and actors by selecting one of the relationship tools in the toolbox, selecting one of the related elements and dragging on to the other related element.

Although you have the full set of class diagram tools at your disposal in the Class Diagram Toolbox, there are a limited number of relationships that should be applied to use cases. Valid relationships between use cases are: includes, extends and generalizes. However, there are no specific tools for includes and extends relationships. These should be modeled as unidirectional associations with stereotypes or stereotyped generalizations (UML 1.1).

Relationships between actors and use cases should be modeled as associations or directional associations.

Note: When naming actors, be aware that actors are stereotyped classes, and there is only one name space for all classes in Rose RealTime. For example, if you name an actor **Server**, you will not be able to create another class named **Server** in the Logical View because there will be a name conflict. We suggest using a naming convention, such as adding an ending like "_actor" to actor names.

Use Case Diagram Toolbox

The use case diagram toolbox is the same as the Class Diagram Toolbox.

Defining Use Cases and Actors

8

Contents

This chapter is organized as follows:

Creating a Use Case

To create a new use case:

1 Right-click on the Use Case View in the **Model View** tab in the browser.

2 Select the **New > Use Case** menu option.

A new use case is created with a default name of **NewUseCase1**.

3 Begin typing to change the name.

You can also create new use cases using the use case tool in the use case diagram. The use case can then be filled out using the Use Case Specification dialog. To access the specification dialog, double-click on the use case in the model browser.

Use Case Specification

A Use Case Specification enables you to display and modify the properties and relationships of a use case in the current model.

To display a Use Case Specification, double-click on any icon representing the use case or right-click on the use case in the model browser and chose **Open Specification** from the model browser.

Specification Content

The Use Case Specification contains the following tabs:

- General tab
- Diagram tab
- Relations tab
- Files tab

General tab

In addition to the elements found in standard Specification Dialogs, the General tab contains:

Name

A use case name is often written as an informal text description of the external actors and the sequences of events between elements that make up the transaction. Use-case names often start with a verb. The name can be entered or changed on the specification or directly on the diagram.

Package

This static field identifies the package to which the components belong.

Rank

The Rank field prioritizes use cases. For example, you can use the rank field to plan what iteration in the development cycle a use case should be implemented.

Abstract

An abstract notation indicates a use case that exists to capture common functionality between use cases (uses) and to describe extensions to a use case (extends).

Diagram tab

Diagrams

The Diagrams list box lists all the diagrams owned by the use case. The diagram list consists of two columns. The first (unlabeled) column displays the diagram icon type for the diagram. The second column displays the diagram name. To insert a new diagram in the list, click one of the Insert choices in the popup menu that corresponds to the diagram type.

Relations tab

Relations

The Relations list box lists all the relationships associated with the selected use case. The client and supplier names and type icons are displayed to the right of the relation name. Double-clicking on any column in a row displays the element's specification.

Files tab

A list of referenced files is provided here. The files list popup menu allows you to insert and delete references to files or URLs.

You can link external files to model elements for documentation purposes.

Creating an Actor

Actors can be created in the Use Case View of the **Model View** tab in the browser.

To create a new actor:

1. Right-click on the Use Case View package in the Model browser.
2. Select the **New >Actor** menu option.

   A new actor is created with a default name of NewClass1.
3. Click on the new actor to change its name.

Actors can also be created using the actor tool in the Use Case Diagram Editor or Class Diagram Editor.

Note that an actor is simply a stereotype of a class. You can define many of the same properties on an actor as you can on any other class. To add to the actor's definition, double-click on the actor to open the Actor specification dialog.

Actor specification

An Actor Specification looks identical to a Class Specification, except that the stereotype field is set to actor. However, some of the fields in the class specification are not applicable to actors and are therefore disabled.

Creating Class Diagrams 9

Contents

This chapter is organized as follows:

This chapter describes Creating a Class Diagram, as well as creating various types of relationships and associations.

Creating a Class Diagram

Class diagrams are created in the Logical View of the Model browser. A **Main** class diagram is always present in the Logical view. The Main class diagram should be used to describe the relationships between the primary packages and a layered system. Other class diagrams can be created to communicate key relationships within portions of the model.

To edit the Main class diagram:

1 Double-click on the Main diagram in the Logical View package in the **Model View** tab in the browser.

The Class Diagram editor appears.

2 Place classes, packages, capsules, and protocols in the diagram by dragging them from the model browser, or by using the tools in the toolbox.

3 Draw relationships and associations among the classes, packages, capsules, and protocols using the toolbox.

To create a new class diagram:

1 Right-click on the Logical View package in the **Model View** tab in the browser.

2 Select **New > Class Diagram** from the menu.

3 Enter the name of the class diagram.

There are several additional topics on creating relationships between model elements in the class diagram:

- Creating Association Relationships
- Creating Aggregation Relationships
- Creating Inheritance Relationships
- Creating Dependency Relationships
- Creating Reflexive Relationships
- Creating Package Relationships
- Defining multiplicity in relationships

Using the Class Diagram Editor

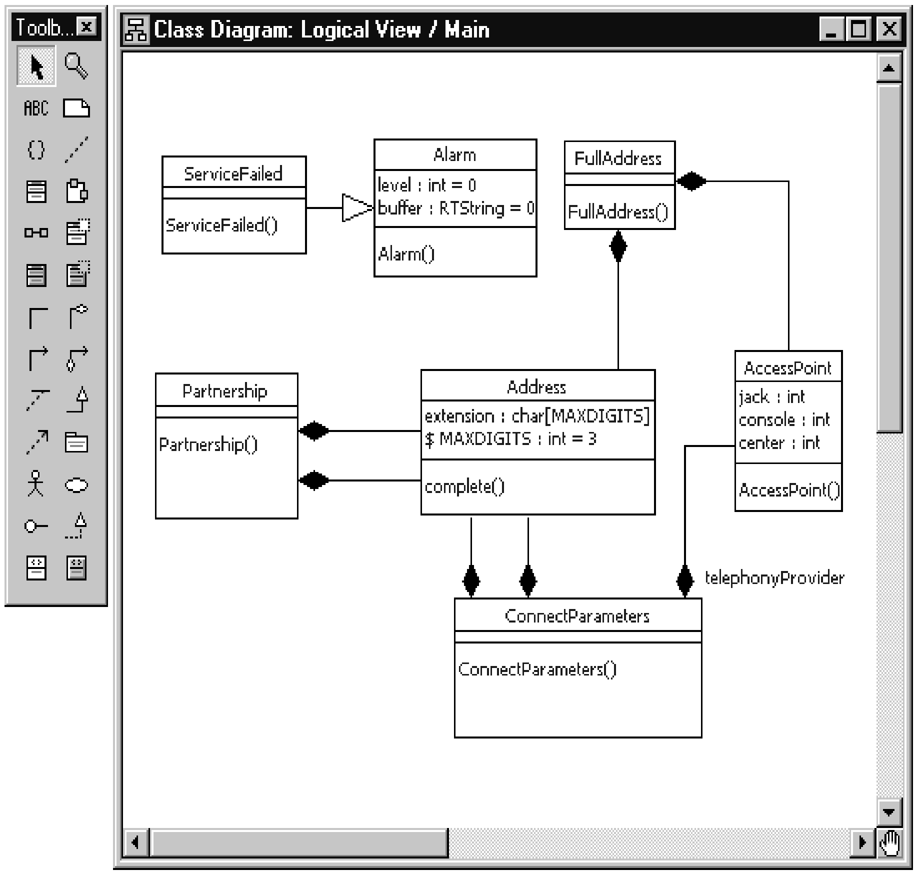
The class diagram editor is used to create a diagram showing classes and associations among the classes. The class diagram consists of two parts:

- the diagram area
- the Class Diagram Toolbox

Elements of the class diagram, such as classes, capsules, use cases and associations, are added using the toolbox.

The window title bar shows the full name of the class diagram.

Figure 32   Class diagram editor



Class diagrams contain icons representing classes, capsules, protocols, packages, interfaces, and their relationships. You can create one or more class diagrams to depict the classes at the top level of the current model; such class diagrams are themselves contained by the top level of the current model. You can also create one or more class diagrams to depict classes contained by each package in your model; such class diagrams are themselves contained by the package enclosing the classes they depict, the icons representing logical packages and classes in class diagrams.

Every class is assigned to a logical package. When you create a class using a creation tool from the class diagram toolbox, the class is assigned to the logical package containing the class diagram.

Diagram Entities

There are four types of entity that you can place on a class diagram:

- Classes
- Capsules
- Protocols
- Packages

Relationships

There are four basic kinds of relationship you can create through the class diagram. Refer to the following topics:

- *Creating Association Relationships* on page 160
- *Creating Aggregation Relationships* on page 167
- *Creating Dependency Relationships* on page 172
- *Creating Inheritance Relationships* on page 168

Creating capsule and protocol aggregations on the class diagram

There are some things that you can do on both the class diagram and the capsule structure diagram, including adding capsule ends and ports. Defining aggregation between a container capsule and a contained capsule results in the creation of a capsule end inside the container capsule. Defining aggregation between a capsule and a protocol results in the creation of a port as part of the capsule. Capsule structure changes made on the class diagram are automatically reflected in the structure editor. Changes made on the structure editor are only reflected on a class diagram if the model elements involved are placed on a class diagram.

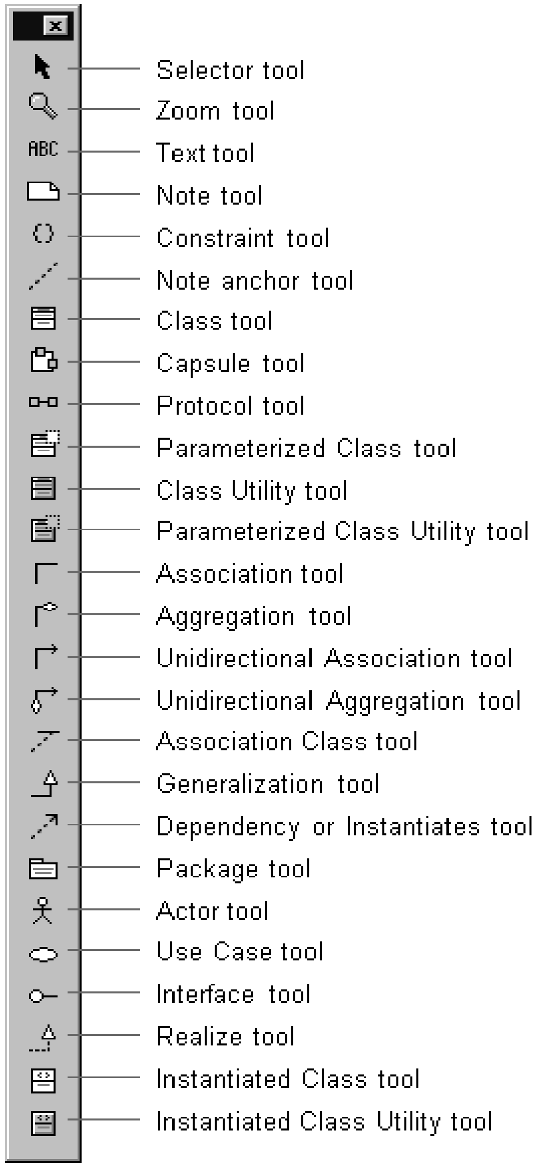
Using the class diagram to visualize existing relationships

You can visualize the existing relationships among these entities. Dragging capsule and protocol classes from a model browser onto a class diagram causes the tool to draw any existing relationships between these elements. For example, if a capsule aggregates another capsule, dragging the two capsule classes on to a class diagram will draw the relationships bases on the filter options chosen in the **Filter Relationship** menu command in the **Query** menu.

Class Diagram Toolbox

The class diagram toolbox contains the following tools (they are not all displayed by default):

Figure 33 Class diagram toolbox



Selector

Use to select objects for moving, resizing, and so forth.

Zoom tool

Use to zoom in on a portion of the class diagram. Click on the tool and then click on the part of the diagram you want to zoom in on.

Text tool

Use to add text to the class diagram.

Note tool

Use to annotate the diagram with textual notes. This is useful for marking up the diagram with explanations, review comments, and so forth. You can drag and drop a diagram or external document from the browser onto a note. Notice that the name of the diagram or external document is underlined. If you double-click on the note, the diagram or external document is opened. You can undo and redo this command.

Constraint tool

Use to add UML constraints to the class. A constraint can be anchored to a view element by using the anchor tool. Currently, constraints do not have any semantic meaning to the tool. There are RRTEI APIs to add or remove, and enumerate constraints in a diagram.

Note anchor tool

Use to anchor a note to a particular element on the class diagram. (See *Using the Class Diagram Editor* on page 152.)

Class tool

Use to place a class on the class diagram. (See *Using the Class Diagram Editor* on page 152.) Pops up a pick list allowing you to choose from an existing class or create a new class.

Capsule tool

Use to place a capsule class on the class diagram. (See *Using the Class Diagram Editor* on page 152.) Pops up a pick list allowing you to choose from an existing capsule class or create a new capsule class.

Protocol tool

Use to place a protocol class on the class diagram. (See *Using the Class Diagram Editor* on page 152.) Pops up a pick list allowing you to choose from an existing protocol class or create a new protocol class.

Parameterized Class tool

Use to place a parameterized class on the class diagram. (See *Using the Class Diagram Editor* on page 152.) Displays a pick list allowing you to choose from an existing class or create a new class. **There is no code generation support for parameterized classes.**

Class Utility tool

Use to place a utility class on the class diagram. (See *Using the Class Diagram Editor* on page 152.) Displays a pick list allowing you to choose from an existing class or create a new class.

Parameterized Class Utility tool

Use place a Parameterized class utility on the class diagram. (See *Using the Class Diagram Editor* on page 152.) Displays a pick list allowing you to choose from an existing class or create a new class. **There is no code generation support for parameterized class utilities**.

Association tool

Use to draw an association between two classes on the class diagram. (See *Using the Class Diagram Editor* on page 152.) Associations can be created between classes (including class utility, parameterized class, and so forth), between capsule classes, and from capsule classes to protocol classes.

Aggregation tool

Use to draw an aggregation between two classes on the class diagram. (See *Using the Class Diagram Editor* on page 152.) Associations can be created between classes (including class utility, parameterized class, and so forth), between capsule classes, and from capsule classes to protocol classes. See *Creating Aggregation Relationships* on page 167 for more information.

Unidirectional Association

Use to draw a unidirectional association between two classes on the class diagram. A unidirectional association is simply an association with navigability limited to one direction. Associations can be created between classes (including class utility, parameterized class, and so forth), between capsule classes, and from capsule classes to protocol classes. See *Creating Relationships* on page 159 for more information.

Unidirectional Aggregate Association

Use to create an association that is unidirectional in the direction it was drawn, with an aggregation at the end. Any association between classes can be converted into this through the specification dialog, as well.

Association Class

Use to link a class with an association between two other classes on a class diagram. Use the Association Specification (by double-clicking on the association after it has been drawn) to specify details of the association semantics. Using the link attribute tool automatically sets the Link Element field on the association to be the class joined to the association with the link attribute tool.

Generalization

Use to indicate that one element is a generalization of another. This is primarily used to indicate a superclass/subclass relationship between classes. Draw the relationship from the specializing element to the generalizing element (that is, from subclass to superclass). Use the Generalize Specification (by double-clicking on the generalization after it has been drawn) to specify details of the generalization semantics.

Adding a generalizes relationship between two classes (including capsule and protocol classes) results in one class being generated as a subclass of the other at code generation time.

Dependency or Instantiates

Use to indicate that one element is dependent on another. This is primarily used to indicate a compilation dependency between classes. Draw the relationship from the dependent element to the dependent-upon element. Use the Dependency Specification (by double-clicking on the dependency after it has been drawn) to specify details of the dependency semantics.

Adding a dependency relationship between two classes (including capsule and protocol classes) results in the dependent class including the `.h` file of the dependent-upon class.

Package

Use to add a package to the diagram. The package is given a default name such as 'NewPackage1'.

Actor

Use to place an actor on a diagram. Displays a pick-list allowing you to select from available classes or create a new class.

Use Case

Use to place a use case on a diagram. This creates a new use case with a default name such as 'UseCase1'.

Interface

Use to place an interface on a diagram. Displays a pick list allowing you to select a class or create a new class.

Realize

Use to indicate that a class realizes an interface or a use case. Draw the relationship from the realizing element to the element being realized.

Instantiated Class

Use to place an Instantiated class on the class diagram. Displays a pick list allowing you to choose from an existing class or create a new class. **There is no code generation support for instantiated classes.**

Instantiated Class Utility

Use to place an Instantiated class utility on the class diagram. Displays a pick list allowing you to choose from an existing class or create a new class. **There is no code generation support for instantiated class utilities.**

Creating Relationships

Relationships among modeling elements take many forms. Most relationships imply an interaction or a dependency between two model elements. The term 'class' in the following descriptions includes capsule and protocol classes as well as "data" (or passive) classes.

See the following topics for the type of relationship you are interested in creating:

- An association is a relationship between two classes (including capsule and protocol classes). An association relationship may have a number of different implications for the generated code, or it may not result in any generated code at all, depending on the specific properties defined on the association. See *Creating Association Relationships* on page 160.
- An aggregation is a more specific form of association that indicates that one class is part of a larger, composite class. That is, one or more instances of one class are considered to be owned by (and are created and destroyed under the control of) an aggregate class. See *Creating Aggregation Relationships* on page 167.
- A dependency relationship indicates that the implementation of one class or package depends on the existence of the definition of another class or package (or some aspect of that class). See *Creating Dependency Relationships* on page 172.
- A generalization relationship indicates that one class inherits properties from (is a subclass of) another class. See *Creating Inheritance Relationships* on page 168.
- A reflexive relationship is one in which an instance of a class may also have associations with other instances of the same class. See *Creating Reflexive Relationships* on page 175.

Creating Association Relationships

Association relationships indicate some form of interaction between two classes. Typically, the association relationship indicates that instances of those classes communicate with each other at run-time.

To create an association relationship in the class diagram editor:

1. Click on one of the two association icons in the class diagram toolbox: the bi-directional association or the uni-directional association. (For more on directionality, see *Changing the Directionality of an Association* on page 175).
2. Click on one of the two classes involved in the association.
3. Drag the association line on top of other class.

   An association line appears between the two classes.

Association Properties

After an association is created between two classes, each of those classes is said to play an end in the association.

There are several properties surrounding the association, including properties of the two ends involved in the association. These properties can be edited by double-clicking on the association to bring up the Association Specification, or by selecting the association and right-clicking. The right-click menu includes properties specific to the end closest to where the mouse was clicked.

The class you terminated the association line on is referred to as End A. The class you clicked on to start drawing the association end is referred to as End B. You can name these ends explicitly through the Association Specification Dialog or via the right-click menu.

Association Specification

An association represents a semantic relationship between two classes. To display the association specification, double-click any association in a class diagram.

Specification Content

The Association Specification dialog consists of the following tabs: General tab, Detail tab, End A and B General tabs, End A and B Detail tabs.

General tab

Name

A name for the association. The name label appears on the class diagram.

Effect on generated code: None.

Parent

The parent the component belongs to (its package) is displayed in this non-editable field.

Stereotype

A stereotype represents the subclassification of an element. It represents a class within the UML metamodel itself (that is, a type of modeling element). Some stereotypes are already predefined, but you can also define your own to add new kinds of modeling types.

Effect on generated code: None.

End A / B

Use these fields to label the ends with names that denote the purpose or capacity wherein one class associates with another. This field is the same as the End field on the End A General and Detail and End B General and Detail tabs. See the End Detail tab for more information.

Element A/B

The Element field specifies the classes of the two elements that this association associates. This field cannot be edited.

Detail tab

Derived

This check box indicates whether the association is computed or implemented directly. The element name for a derived element is adorned by a "/" in front of the name.

Effect on generated code: No code is generated for derived associations.

Link Element

This field lists the attributed associations linked to the association. These attributed associations apply to the association as a whole. Identifies a class representing the association between the two elements.

Effect on generated code: Each of the end classes has a member generated to point to or contain an instance of the link class, depending on the settings of the containment property. The link class has members generated to point to or contain each of the ends of the association.

Name Direction

This field defines the direction of an end. There are three options listed in the pull-down menu associated with the field: <non-directional>, End A and End B.

> Effect on generated code: None.

Constraints

The constraint is an expression of some semantic condition that must be preserved while the system is in a steady state. The constraint on the Detail tab applies to the association as a whole, while the constraint on the Detail A or Detail B tab applies to a particular end.

To apply a constraint, click in the Constraint field and enter the text. Constraints are displayed notationally, surrounded by braces under the end for which it applies.

> Effect on generated code: None.

End A and B General tabs

End A / B

Use this field to label the end with a name that denotes the purpose or capacity wherein one class associates with another. This field is the same as the End A and End B fields on the General tab.

> Effect on generated code: The end name is generated as a member of the class at the other end of the association. That is, if the class at End A is class A and the class at End B is class B, and the name of End A is foo, then class B will have a member named foo of type Class A.

Element

The Element field describes the two elements that this association associates. This field cannot be edited.

Visibility

Specifies the visibility of the data member representing this end in the other class. Visibility options are

- Public - visible to any class
- Protected - visible to this class, any subclasses of this class, and any designated friend classes
- Private - visible only to this class and any designated friend classes
- Implementation - not visible to any other classes

  Effect on code generation: If a data member is generated for this end in the other class, the member will have the visibility specified here. The member is only generated if the other field settings in the End A/B Detail tab are set appropriately.

End A and B Detail tabs

End

A label for the end. This label appears beside the end on the association in the diagram. This field is the same as the End field on the General and End A and End B General tabs. See the field description on the End A/B General tab for more information.

Element

A non-editable field that specifies the classifier for this end.

Constraints

The constraint is an expression of some semantic condition that must be preserved while the system is in a steady state. The constraint on the Detail tab applies to the association as a whole, while the constraint on the Detail A or Detail B tab applies to a particular end.

Effect on generated code: None.

Multiplicity

The multiplicity field defines the maximum number of instances that can exist in this end of the association at any given time. See Multiplicity options for more information.

Effect on code generation: The data member for this end is declared as an array with its size being the largest possible value declared in the multiplicity. If the range is unspecified (e.g., 1..*), the containment value is forced to 'By reference' and a warning is issued if the containment value was originally set to 'By value'.

Aggregation

The Aggregation field has three checkboxes: None, Aggregate, and Composite.

- None - the end is not an aggregate.
- Aggregate - the end is an aggregate; therefore, the other end is a part and must have the aggregation value of none. The part may be contained in other aggregates.
- Composite - the end is a composite; therefore, the other end is a part and must have an aggregation value on none. The part is strongly owned by the composite and cannot be part of any other composite.

Use the Aggregation field to set a direction to either all or part of the relationship among instances of these classes. Only one end of the relationship can be aggregate or composite.

To set the aggregate adornment, click on the Aggregate box in the Association Specification or click **Aggregate** through the shortcut menu. The adornment is a diamond on the relationship.

Effect on code generation: This affects how the other end is stored as a member of this class. Checking the aggregate box allows you to select a containment setting to control how the aggregation will actually be generated in code.

Target Scope

The Target Scope field has two checkboxes: Instance and Classifier.

- Instance - specifies that instances of the client own the supplier class.
- Classifier - specifies that the client class - not the client's instances - owns the supplier class.

You can set this field in the specification or through the shortcut menu.

Effect on code generation: The data member is scoped to the classifier in the other end class.

Friend

The friend field designates that the supplier class has granted rights to a client class to access its non-public parts.

> Effect on code generation: This field currently has no effect on code generation. See Designating friend classes for information on how to specify friends.

Navigable

The Navigable field indicates in which direction the association is traversed. By default, ends are bidirectional and no navigation notation is provided.

To set an end's navigation, click on the Navigable box in the Association Specification or click **Navigable** through the shortcut menu. The navigable arrowhead points in the direction of the end, unless a containment adornment is displayed. Containment adornments override navigable adornments.

> Effect on code generation: If the navigation check box is not checked, it signifies that the class at the other has no visibility of the class at this end of the association; therefore, no member will be generated in the other end class.

Keys/Qualifiers

A key or qualifier is an attribute that uniquely identifies a single target object. The attributes allow 1..n or n..n associations, and reduce the number of instances. The list box displays all keys or qualifiers currently defined.

To enter a key or qualifier, click **Insert** from the popup menu or press the insert key. An untitled entry is placed in the name and type field. To change the entry, select to highlight and type in a new name.

> Effect on generated code: The Keys/Qualifiers entries currently have no effect on generated code.

for sending/receiving data.

Creating Aggregation Relationships

Aggregation relationships are a form of association relationship that indicate one class (the contained class) is a part-of another class (the container, or aggregate class).

To create an aggregation relationship in the class diagram editor:

1. Click on the aggregation icon in the class diagram toolbox.
2. Click on the container class that will contain the class in the diagram.
3. Drag the association line on top of the contained class.

   An association line appears between the two classes, and a diamond (aggregate) symbol appears beside the contained class.

Aggregation usually indicates specific run-time constraints that exist on the relationship:

- An instance of the contained class cannot exist outside of an instance of the aggregate class.
- The creation of an instance of the aggregate class usually results in the automatic creation of an instance of the contained class.
- The destruction of an aggregate instance results in the automatic destruction of any contained instances.

Implementation details of the aggregation, such as whether the contained object is referenced by a pointer or embedded, can be specified through the End A and B General tabs and End A and B Detail tabs.

Creating an Association Class

A relationship in itself may have state and identity distinct from the instances involved in the relationship. In order to implement a relationship, it may be necessary to define a class representing the relationship.

To create an association class:

1. In the class diagram editor, click on the class icon in the class diagram toolbox.
2. Click on the diagram to place a new class.

3 Enter the name of the class.

4 Click on the link attribute icon in the class diagram toolbox.

5 Click on the association class and drag the link attribute line to the association it modifies.

Aggregation Specification

An aggregation represents a special bidirectional semantic relationship between two classes, wherein one or more instances of one class are contained within an instance of the aggregating class.

The aggregation specification dialog is the same as the Association Specification with the **End A aggregate** check-box turned on.

Creating Inheritance Relationships

To define an inheritance relationship:

1 Open the class diagram where you want the inheritance relationship to appear.

2 Click on the Generalization icon in the class diagram toolbox.

3 Click on the intended subclass.

4 Drag the generalization line over the intended relationship.

Creating an Inheritance Tree

To add other subclasses to the inheritance relationship to create an inheritance tree:

1 Using the generalization tool, drag a generalization line from each intended subclass to the inheritance triangle by the intended superclass.

Two separate inheritance relationships can be merged into a tree by moving one inheritance triangle symbol on top of another.

Exclusions

When you create a new generalization between capsules or protocols, the Inheritance Rearrangement dialog may appear prompting you to exclude new superclass properties. This allows the subclass to not inherit certain properties (state machine, capsule structure and protocol signals) defined in the superclass. This is helpful, for example, if your subclass has a state machine and you want to intelligently merge the state machines rather than just blindly inherit the superclass state machine. You can initially exclude the superclass elements, and then gradually re-inherit them as you edit your state machine.

If you select **Copy** or **Cut** from the **Edit** menu, a dialog appears warning you that items whose parents aren't being cut or copied will not get pasted. You have the option of checking the box, Don't warn anymore this session.

See Inheritance for more information.

Generalize Specification

A generalize relationship between classes shows that one class shares the structure or behavior defined in one or more other classes.

The Generalize Specification consists of the following tabs: General and Files.

General tab

Name

A name for the relationship.

Owner

A non-editable field indicating the name of the subclass.

Stereotype

Specify a stereotype to apply to the relationship.

Visibility

Specifies the visibility of the generalization. Visibility options are

- Public - visible to any class
- Protected - visible to this class, any subclasses of this class, and any designated friend classes
- Private - visible only to this class and any designated friend classes
- Implementation - not visible to any other classes

Friendship Required

Select the **Friendship required** check box to specify the supplier class has granted rights to the client class to access its non-public members. In the case of a generalization, the subclass is granted friend access right to superclass members.

Effect on code generation: This field currently has no effect on code generation.

Virtual Inheritance

Select the **Virtual Inheritance** check box to ensure that only one copy of the base class will be inherited by descendants of the subclasses.

Inheritance in Rose RealTime

You can define generalization relationships between classes (including capsule and protocol classes) in Rose RealTime. When a generalization relationship is defined, the specializing class inherits the properties including all attributes, operations, state machine, signals, etc.) of the generalizing class.

For capsule and passive (data) classes, all public and protected operations are inherited, as well as all public and protected attributes.

For capsule classes, the structure elements (the ports and capsule roles) are also inherited by the specializing class.

For protocol classes, the signals are inherited as well as the state machine, if defined.

Promoting and Demoting Elements

Capsule structure elements (ports, capsule roles and bindings), capsule and protocol state machine elements, and protocol signals can all be promoted and demoted in the class hierarchy.

For example, you can select a port from a capsule and demote it, such that it is removed from the generalizing capsule class' structure and moved into each of its subclasses. The port is no longer inherited, it becomes part of the subclass' structure and is removed from the superclass.

As another example of promoting/demoting, you can select a state in a capsule subclass and 'promote it' such that the state is moved into the superclass state machine, and is inherited by all the capsule's subclasses.

To promote an element from a subclass to its immediate superclass:

1 For state machine and capsule structure elements, select the element in the diagram, and choose **Parts > Promote** or **Parts > Demote**.

2 For all capsule structure elements, capsule and protocol state machine elements, and protocol signals, you can also right-click on the element and choose **Promote** or **Demote** from the popup menu.

Potential Conflicts Caused by Promote/Demote

A promote or demote operation may fail if there is a name conflict in the subclass or superclass. For example, if you try to promote a state named Ready from a capsule subclass into its superclass, you will get an error if any other subclass of the superclass also has a state named Ready.

Excluding Elements

In addition to promoting and demoting, you can also exclude certain inherited elements (the same set that can be promoted/demoted) from a capsule subclass or protocol subclass.

An excluded element is removed from the subclass diagram or properties. Note that for structure elements (ports and capsule roles), the excluded element will still be inherited in the code of the subclass, since these elements are generated as members of the superclass and automatically inherited by the subclass. This means, you should not reuse the name of any excluded element or you may cause a name conflict at compile-time.

Reinheriting Excluded Elements

To exclude an inherited element, right-click on the element in the diagram or properties editor and choose **Remove/Exclude** from the popup menu. If this menu entry is not available, the element cannot be excluded. You can reinherit an excluded element by right-clicking on it and selecting **Inherit**. In protocol classes, click the **Show Excluded** check box on the Signals tab to see excluded signals. In a Structure Editor or State Editor right-click on the diagram and select **Filter > Excluded** (turn off the Exclusions filter) to see any excluded elements.

Rearranging inheritance hierarchies

If you choose to make an generalization relationship between two capsules or between two protocols, you will be prompted with a dialog allowing you to exclude the properties of the new superclass. See *Creating Inheritance Relationships* on page 168 for more information.

If you break a generalization relationship between capsule or protocol classes, you will be presented with a dialog option to Absorb all current superclass properties. This allows you to essentially copy the elements that the subclass had previously inherited from the superclass directly into the subclass definition and then break the inheritance relationship between the two classes.

Creating Dependency Relationships

A dependency relationship is a vague form of relationship between two classes that simply indicates that something in one class depends on the definition of something in the other class.

To create a dependency relationship:

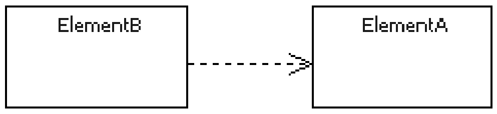
1. Click the dependency tool.
2. Click on the intended dependent class.
3. Drag and drop on to the class that is being depended upon.

Draw a dependency relationship between two classes, or between a class and an interface, to show that the client class depends on the supplier class/interface to provide certain services, such as:

- The client class accesses a value (constant or variable) defined in the supplier class/interface.
- Operations of the client class invoke operations of the supplier class/interface.
- Operations of the client class have signatures whose return class or arguments are instances of the supplier class/interface.

Graphical Notation

A dependency relationship is a dotted line with an arrowhead at one end:



The arrowhead points to the supplier class. In this example, class A is dependent on class B.

Naming

Use the relationship name to identify the type or purpose of the relationship.

Valid Applications

You can draw a dependency relationship between logical packages.

Add Class Dependencies Wizard

A wizard is supplied to automate the creation of dependencies between a large number of classes (for example, after loading a Rose or ObjecTime Developer model).

See Add Class Dependencies.

Dependency Specification

The dependency relationship indicates that the client class depends on the supplier class to provide certain services. One class may use another class in a variety of ways. Typically, a dependency relationship indicates that the operations of the client access members (operations or attributes) of the supplier. Dependencies can also be drawn between packages.

You can change properties or relationships by modifying the icon on the diagram or by editing the specification.

You can also view the specification by double-clicking on the name of the dependency relationship in the Relations tab of the Class Specification.

The associated diagrams or specification are automatically updated.

The Dependency Specification contains the following tabs: General, Files.

General Tab

Name

A name for the dependency relationship.

Class

A non-editable field listing the client class.

Stereotype

Specifies a stereotype to attach to the dependency.

Friendship Required

A check box indicating whether the client class should be generated as a friend of the supplier to provide access to non-public members on the supplier.

Effect on code generation: This field currently has no effect on code generation.

Export Control

Specifies the visibility of the dependency. Visibility options are

- Public - visible to any class
- Protected - visible to this class, any subclasses of this class, and any designated friend classes
- Private - visible only to this class and any designated friend classes
- Implementation - not visible to any other classes

Effect on code generation: None.

Multiplicity from

Describe the multiplicity of the client side of the relationship.

Effect on code generation: None.

Multiplicity to

Describe the multiplicity of the supplier side of the relationship.

Effect on code generation: None.

Creating Reflexive Relationships

An object may sometimes need to communicate with other objects of the same class. In the class diagram, this appears as a class having a relationship with itself. This is called a *reflexive relationship.*

To create a reflexive relationship in the class diagram editor:

1. Click on the association icon in the class diagram toolbox.
2. Click on the class with the intended reflexive relationship.
3. Drag the association line outside of the class border and then back over the class.

   An association line appears drawn from the class back onto itself.

Changing the Directionality of an Association

There are two forms of association that can be created: bi-directional and uni-directional. Bi-directional associations are highly unusual in practice in the development of applications, as a bi-directional association suggests that communication can be initiated in either direction. Most associations between classes in an application are fundamentally uni-directional; that is, an instance of one class always initiates communication to one or more instances of the other class.

To change the directionality of an association after it has been created:

1. Open the association specification dialog by double-clicking on the association in the diagram.
2. Select the **Navigable** check box on the **End A Detail** or **End B Detail** tab to change the directionality.

Creating Package Relationships

Relationships can be defined between packages. A relationship between two packages indicates that one package is dependent on another. A dependency between packages exists when one or more classes in one package initiates communication with a class or classes in another package. The first package is dependent on the second package.

To create a dependency relationship between two packages in the class diagram editor:

1 Click on the dependency icon in the class diagram toolbox.

2 Click on the package that will be the dependent package in the diagram.

3 Drag the dependency line on top of the package being depended on.

   A dependency association appears between the two packages, with an arrowhead pointing from the dependent package to the package it depends upon.

Creating Realize Relationships

A realize relationship between classes and interfaces and between components and interfaces shows that the class realizes the operations offered by the interface.

Naming

Use the relationship name to identify the type or purpose of the relationship.

Valid applications

You can draw a realize relationship between a ClassInterface and a Component Interface. The relationship between a component and an interface can not be drawn explicitly. It is created when an interface is assigned to a component through the browser or a specification editor.

Realize Relationship Specification

General Tab

Name

A name for the Realize relationship

Documentation

Use to describe the Realize relationship.

Adding and Hiding Classes, and Filtering Class Relationships

The commands on the Query menu provide powerful facilities for controlling which model elements are represented by icons in the current diagram.

The options are as follows

- Add Classes - adds classes to the diagram by name.
- Expand Selected Classes - adds classes to the diagram based on their relationships to selected classes.
- Hide Selected Classes - removes selected classes from the diagram and optionally removes their clients or suppliers from the diagram.
- Filter Relationships - controls which kinds of relationships appear in the current diagram.

Creating Collaboration Diagrams 10

Contents

This chapter is organized as follows:

Creating Capsule Structure

Capsules are one of the primary modeling elements in Rose RealTime. Complete executable code implementations are generated by the toolset for capsules.

Capsule structure is defined through the Structure Diagram Editor.

There are three kinds of structural element that may be added to a capsule structure diagram:

- Capsule roles
- Ports
- Connectors

None of these elements are required. A capsule does not have to have any structural elements. To do anything useful, a capsule usually requires at least a port so that it can communicate with other capsules.

Creating a complete capsule structure definition may consist of any of the following basic steps:

- Adding a Capsule Role
- Creating a Port
- Connecting Ports on Capsule Roles Together

Using the Structure Editor

The structure editor is used to define the structure of a capsule class. That is, how instances of that capsule class are composed of other capsule class instances and protocol class instances (ports). The structure editor consists of three parts: the structure diagram area, the structure browser, and the Structure Diagram Toolbox.

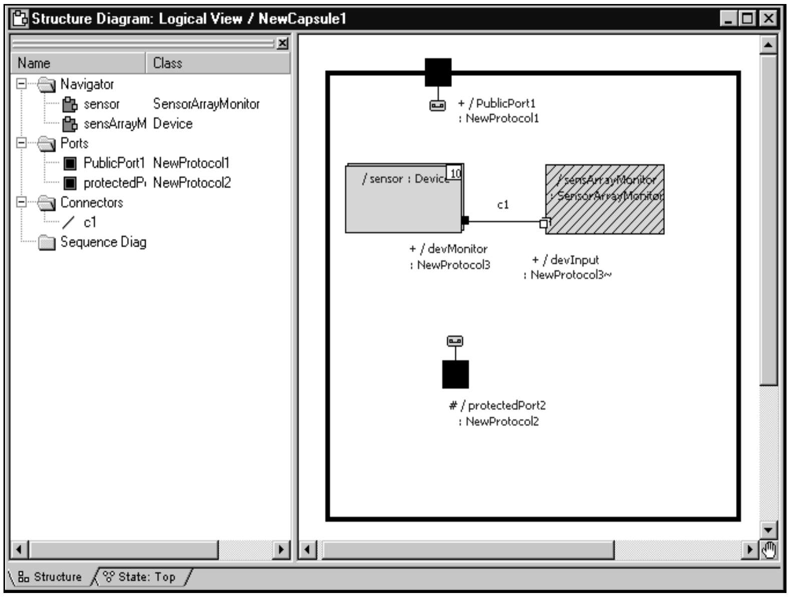
Structure elements, such as ports and capsule roles, can be created by dragging a protocol or capsule on to the structure diagram from any browser (usually from the model browser). Structure elements can also be added using the toolbox. The structure browser can be used to navigate to, and open editors and specification dialogs on contained structure elements.

Several standard diagram manipulations can be performed, including resizing, scaling, and filtering.

You can use the popup menu to navigate between structure and state diagrams.

Figure 34shows a sample structure editor. The window title bar shows the full name of the class. The left side of the window contains the structure browser. The right side contains the structure diagram area. The structure toolbox is not shown. It is usually anchored outside of the diagram window.

Figure 34 The Structure Editor



UML Options

You can use the popup menu to toggle the following UML options:

Base UML notation

Converts the structure diagram so that it uses base UML notation.

Show Classifier Name on Roles

Lets you turn off the classifier name portion of a role label.

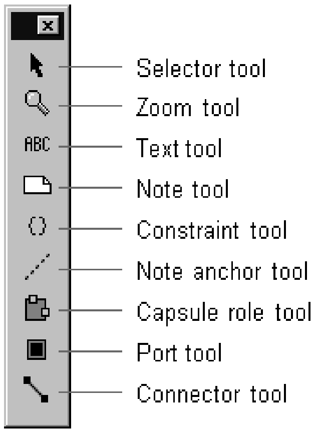
Show Protocol Name on Ports

Lets you turn off the classifier name portion on ports.

Structure Diagram Toolbox

The structure toolbox contains tools for adding elements to the structure diagram.

Figure 35 Structure toolbox



Selector tool

Use to select objects for moving, resizing, and so forth.

Zoom tool

Use to zoom in on or out from diagrams.

Text tool

Use to add text anywhere in the structure diagram.

Note tool

Use to annotate the diagram with textual notes. This is useful for marking up the diagram with explanations, review comments, and so forth. You can drag and drop a diagram or external document from the browser onto a note. Notice that the name of the diagram or external document is underlined. If you double-click on the note, the diagram or external document is opened. You can undo and redo this command.

Constraint tool

Use to add UML constraints to the diagram. A constraint can be anchored to a view element by using the anchor tool. Currently, constraints do not have any semantic meaning to the tool. There are RRTEI APIs to add or remove, and enumerate constraints in a diagram.

Note anchor tool

Use to anchor a note to a particular element on the diagram.

Capsule role tool

Use to add a capsule role to a capsule collaboration diagram. A pick-list is displayed allowing you to select the class of the capsule role from the list of capsule classes. The first entry in the pick-list menu is **Create a New Capsule** which creates a new capsule class with a default name such as 'NewCapsule1'.

Port tool

Use to add new ports to the capsule class structure. Ports can be placed either in the internal structure (inside the black interface boundary, which makes them protected, or on the structure interface, which makes them public.

A popup menu appears on the capsule role allowing you to select the protocol class of the port from the list of available protocol classes (that is, all protocol classes in the model). The first entry in the popup menu is **Create a New Protocol** which creates a new protocol class called 'NewProtocol1'.

Protected ports are automatically created as end ports. Public ports are created as end ports by default.

Connector tool

The connector tool is used to wire ports together. Usually connectors bind ports on different contained capsule roles together within the container capsule class. Connectors can also bind internal end ports of the container class to other ports. Interface end ports can only be bound within the context of a container class.

Only compatible ports can be connected together. Compatible ports are usually two ports of the same protocol class, one of which is conjugated.

Creating a Port

There are four different ways to add a port to a capsule:

- Drag and drop a protocol class name from the model browser onto the capsule structure diagram.
- Draw an aggregation between a capsule class and a protocol class on a class diagram.

- Use the port tool on the capsule structure diagram toolbox. Select the tool and click on the capsule boundary to add a public port. Click inside the boundary to add a protected port.
- From the Navigator area of a capsule diagram editor (either the collaboration diagram editor or the state editor), right-click on the Ports folder and select **Add New Port** from the popup menu.

If you use the port tool, a pick-list appears on the resulting port allowing you to select the protocol class to be used from a list of available protocol classes. The first entry in the pick-list is **Create a New Protocol** which creates a new protocol class called 'NewProtocol1'. If you choose to create a new protocol class it will be added to the same package as the container capsule class.

Creating a Non-Wired Port Using a System Protocol

The easiest way to create a non-wired port to access one of the system services (a Frame, Timing, Log or Exception port) is as follows:

1. Right-click on the **Ports** folder on the Navigator area of the capsule's state diagram editor.
2. Select **Add New Port** from the popup menu.
3. Select one of the system services from the list that appears.

Port Specification

The Port Specification provides control over information about ports on capsules.

The Port Specification contains two tabs: the General Tab and the Files Tab.

General Tab

Name

The port is referenced by a name. The default name provided when the port is first created is based on the protocol name. In addition to appearing on the structure diagrams, the port name is used by the behavior of the capsule containing the port. To send and receive messages, the capsule's behavior references the port name in detailed code, and in transition trigger events.

Stereotype

A stereotype represents the subclassification of an element. It represents a class within the UML metamodel itself, that is, a type of modeling element. Some stereotypes are already predefined, but you can also define your own to add new kinds of modeling types.

Stereotypes can be shown in the browser and on diagrams. The name of the stereotype may appear in angle brackets <<>>, depending on the settings found in either the Diagram or Browser tabs of the Options dialog located under the **Tools** menu. Refer to the Stereotype chapter for more information on stereotypes.

To show stereotypes on the diagrams, click **Options** from the shortcut menu and click **Stereotype Name** or **Stereotype Icon**. **Stereotype Name** displays the name in angle brackets (that is, <<stereotype>>). **Stereotype Icon** displays the graphical representation.

Protocol

Specifies the protocol class to be used for the port. The protocol class (together with the Conjugation check-box) determines the set of messages that can be sent through this port (the **out** set), and the set of messages that can be received (the **in** set). The field has a pull-down menu to select from the available protocol classes in the model. The pull-down list always includes the service protocols for communicating with the target services library.

The **Open** button opens the Protocol Specification for the selected protocol class.

Cardinality

Specifies the number of instances of the port that will appear at run-time. The port is implemented as a member variable of the containing capsule. The variable may be an array of ports, connected to multiple ports on the other end of the connector. In this case, the port name points to an array of port instances. The Cardinality specifies the size of the array. Not all port instances in the array are necessarily connected. Individual port instances are referenced by indexing into the array. See the message send syntax in the *Programmer's Guide* for details. The Cardinality can be specified with an integer value or as the name of a constant defined within the model.

Conjugated

A conjugated port is one in which the standard protocol class definition of in and out signals is reversed. That is, on a conjugated port the protocol class out signals become the port's in signals, and the protocol class in signals become the port's out signals. This enables two ports of the same protocol class to be connected together without having to define a separate reverse protocol. A connection can be made between two ports of the same protocol by conjugating one of the ports. This is the most common way of establishing communicating between two ports.

End Port

Toggle this check box to make the port an End Port, capable of sending and receiving messages. End Ports provide a connection between the behavior of the capsule containing the end port and the outside world. If this check-box is not checked, then the port is a *relay port. Relay ports cannot be protected, they must be public.*

In order to send messages, a capsule must have end ports. The end port's protocol defines the set of messages that can be sent.

In order to receive messages and process them within the capsule's behavior, the capsule must have end ports. The end port's protocol defines the set of messages that can be received.

Messages received on relay ports are not visible to the behavior of the capsule containing the port. Relay ports are intended to be connected to capsule roles contained within the capsule. Relay ports take messages from outside of the capsule and relay them through the capsule's encapsulation boundary to other capsules contained inside.

Wired

Toggle this check box to make the port a wired port. Wired ports are connected to other wired ports using connectors (via the Connector tool in the capsule structure diagram). Non-wired ports are connected to other non-wired ports by name.

The connection of wired ports is done automatically based on the system structure. Wired ports on fixed capsules are connected at initialization time. Wired ports on optional and plug-in capsules are connected dynamically when the capsule is instantiated or plugged-in.

The connection of non-wired ports may be done in two ways:

1. Automatically by name at the time the capsule is initialized (Automatic Registration).

   In this case, when the capsule is initialized, a non-wired protected port is connected to any non-wired public port of the same name. See Rules for Non-Wired Port Connection.

2. Dynamically by a name specified by the capsule's behavior (Application Registration).

In this case, the port is not connected at initialization time, it is connected when the capsule's behavior invokes a service function to register the port by a specified name. The same port may in fact be registered under different names at different points in the model execution.

This is determined by the registration method selected.

Protected

This check-box determines whether the port is visible outside of the capsule boundary. If the port is not protected, it is public. Public ports are part of the capsule interface and are visible to other capsules.

Notification

If this checkbox is selected, the port will receive **rtBound** and **rtUnbound** messages from the services library when ports get connected and unconnected.

Publish

Determines whether the port is visible (SSP) or invisible (SAP) from outside.

Registration

This selection is only enabled for non-wired ports. Non-wired ports are registered by name with a name service that performs the connection. Connections are made between protected non-wired ports (service clients) and a single public non-wired port (the service provider). There are two registration modes: **Automatic** and **Application.**

Files Tab

A list of referenced files is provided here. The files list popup menu allows you to insert and delete references to files or URLs.

You can link external files to the specification for documentation purposes.

Adding a Capsule Role

Capsule roles may be added to a structure editor by dragging a class name from a browser onto the structure diagram.

You can also use the Capsule role tool from the structure toolbox. A pick-list is displayed on the capsule role allowing you to select the class of the capsule role from the list of capsule classes. The first entry in the pick-list is **Create a New Capsule** which creates a new capsule class called 'NewCapsule1'. If you choose to create a new capsule class it is added to the same package as the container capsule class.

The class specifies the "type" for the role. In the case of optional or plug-in roles, instances of other classes may actually be incarnated or imported into the role at execution time if they are of compatible types (that is, they have the same interfaces and are subclasses of the specified capsule role class).

Capsule Role Specification

The Capsule Role Specification provides control over the properties of a capsule role in a capsule structure diagram. Capsule roles are references to capsule classes.

The Capsule Role Specification Dialog is a standard Specification Dialog, with additional fields controlling the properties of the capsule role.

General Tab

Name

The name of the capsule role within the container capsule structure. The capsule role name may be used in the detailed code of the container capsule.

Class

The Class field defines the Capsule Class to be used in instantiating this role. If the capsule role is an Optional role or a Plug-In role, then subclasses of the specified Class may also be instantiated into this role, but only if the substitutable flag is checked.

Cardinality

The Cardinality field defines the maximum number of capsule instances that can exist in this role at any given time. If the role is Fixed, then the number of instances of the role instantiated at run-time will be exactly the number defined in the Cardinality field. If the role is Optional, then up to <Cardinality> instances may be created at run-time. See Cardinality options.

Substitutable

This check box indicates whether subclasses of the specified capsule role's class can be instantiated into this role. This may happen in one of two ways:

1. If the capsule role is Optional, the container capsule may instantiate a subclass of the specified capsule class into the capsule role.
2. A subclass of the container capsule may override the class of the inherited capsule role.

Fixed

If the fixed check-box is checked, then a capsule of the specified class is automatically instantiated into the role in every instance of the container capsule at run-time. A number of instances equal to the specified cardinality will be created at initialization time.

Optional

If the optional check-box is checked, then the capsule role is instantiated under the program control of the container class. The container class must explicitly instantiate the capsule role within the detailed code of the container capsule state machine. This is done using the incarnate function of the Frame service.

Plug-In

If the Plug-In check-box is checked, then the capsule role is never directly instantiated, but rather an already existing instantiation from another capsule decomposition is imported into the role. That is, an existing capsule is dynamically “plugged in” to the specified role under the program control of the container class. The container class state machine must explicitly request the plug-in of a capsule at run-time within the detailed code. This is done using the import function of the Frame service.

Connecting Ports on Capsule Roles Together

To enable communication between capsules, you must connect together the ports on their interfaces.

You can only connect compatible ports together. For a port to be compatible, the out signals on each side must be a subset of the in signals on the other side. Usually, this is satisfied by connecting the base role and conjugate role of the same protocol together.

Connector Specification

The Connector Specification provides control over the properties of a connector in a capsule structure diagram. Connectors connect ports together to enable communication among capsules.

There are two tabs: General and Files.

General Tab

Name

The name of the connector. Connector names are not usually displayed on the structure diagram and are not significant in the generated code.

Delay

Specifies a communication delay across a connector. This field is for documentation purposes only. There is no validation or calculation of actual communication delays at run-time.

Cardinality

Specifies the number of connectors indicated by a connector line. When a connector is used to connect ports with cardinality > 1 or ports on capsule roles with cardinality > 1, the connector cardinality should match the cardinality of the port/capsule combination on either side of the connection.

Creating a Collaboration Diagram

To create a new collaboration diagram:

1 Select a package, class, capsule, or use case in the Logical View or Use Case View where you want to define the collaboration

2 Right-click on the element in the model browser.

3 Select **New > Collaboration Diagram**.

4 Enter the name for the collaboration diagram

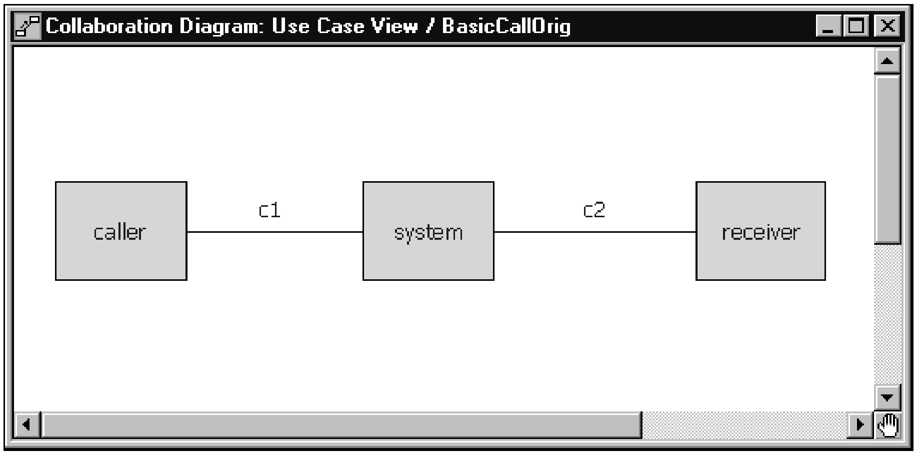
Using the Collaboration Diagram Editor

The collaboration diagram editor is used to create a diagram showing associations among object roles. An association between classifier roles is called an association role. A collaboration diagram represents a particular object configuration at run-time. The collaboration diagram consists of two parts: the diagram area and the Collaboration Diagram Toolbox. Multiple Collaboration diagrams can exist in the same model.

Elements of the collaboration diagram - such as classifier roles, capsule roles, and association roles - are added using the toolbox.

The window title bar shows the full name of the collaboration diagram.

Figure 36 Collaboration diagram editor



Relationship Between Collaborations and Sequences

The collaboration diagram editor shows the general communication pattern among a set of objects for a particular scenario at run-time. You can associate sequence diagrams with a collaboration diagram. The relationship is that a sequence diagram shows a particular execution of a given scenario. There may be many sequences showing different alternative paths for the same scenario. They should all have the same basic collaboration pattern, though.

In the example above, the scenario is a telephone call. There are three roles being played by objects at run-time. A caller represents the object initiating the call. The receiver represents the object receiving the call. The system represents the object that makes the connection between them. Several sequence diagrams could be derived from this collaboration. For example, one sequence diagram might show a completed call where the receiver answers. Another sequence might show a call that is not answered.

Opening a Sequence Diagram

To open a dialog listing all the Sequence diagrams associated with a particular Collaboration diagram, select **Open Sequence Diagrams** from the popup menu.

Sequence Overlays

You can also overlay Message Flow Arrows from a Sequence diagram on top of the Collaboration dialogs by selecting **Sequence Overlays...** from the popup menu. Only "Request" actions - Call and Send - are shown. Create and Destroy messages are not. Messages are only displayed when there is an existing Association Role or Connector to bind them to. Messages To or From the Environment are not displayed.

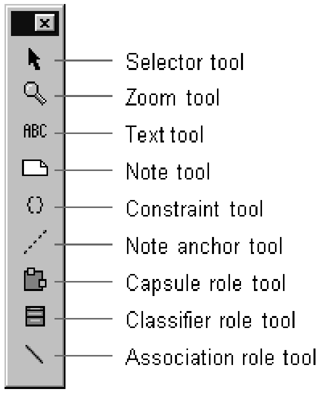
Code Generation

There is no code generated from the collaboration diagram. It is for communication purposes only. The capsule structure diagram is a specialized form of collaboration diagram with specific constraints that enable code to be generated to implement the communication patterns shown in the capsule structure.

Collaboration Diagram Toolbox

The collaboration diagram toolbox contains tools for adding elements to the collaboration diagram.

Figure 37 Collaboration diagram toolbox:



Selector tool

Use to select objects for moving, resizing, and so forth.

Zoom tool

Use to zoom in on a portion of the diagram. Click on the tool and then click on the part of the diagram you want to zoom in on.

Text tool

Use to add text anywhere in the structure diagram.

Note tool

Use to annotate the diagram with textual notes. This is useful for marking up the diagram with explanations, review comments, and so forth. You can drag and drop a diagram or external document from the browser onto a note. Notice that the name of the diagram or external document is underlined. If you double-click on the note, the diagram or external document is opened. You can undo and redo this command.

Constraint tool

Use to add UML constraints to the diagram. A constraint can be anchored to a view element by using the anchor tool. Currently, constraints do not have any semantic meaning to the tool. There are RRTEI APIs to add or remove, and enumerate constraints in a diagram.

Note anchor tool

Use to anchor a note to a particular element on the diagram.

Capsule Role tool

Use the Capsule Role tool to place a capsule_role on the collaboration diagram. When you place a capsule role, a pick-list is displayed allowing you to select from available capsule classes, create a new capsule class, or leave the class unspecified. The class specifies a type that must be satisfied by any instances in that role. In practice, this usually means that subclasses of the specified capsule class can fill the role.

This tool also appears on the capsule Structure Diagram Toolbox. The tool performs the same function in both diagrams.

Classifier Role tool

Use the Classifier Role tool to place a classifier role on the collaboration diagram. When you place a classifier role, a pick-list is displayed allowing you to select from available classifier classes, create a new class, or leave the class unspecified. The classifier specifies a type that must be satisfied by any instances in that role. In practice, this usually means that subclasses of the specified class can fill the role.

Association Role tool

Use the Association Role tool to draw a connection between two roles (capsule roles or classifier roles). An association between roles is a form of association with more explicit meaning than an association at the class level. It specifies that instances satisfying the types specified for these roles have some form of direct communication relating to the interaction specified for this collaboration.

Classifier Role Specification

The Classifier Role Specification provides control over the properties of a classifier role in a collaboration diagram.

The Classifier Role Specification is a standard Specification dialog, with additional fields controlling the properties of the classifier role.

There are two tabs: the General Tab and the Files tab.

General Tab

Name

The name of the classifier role within the collaboration.

Stereotype

A stereotype label for the association.

Classifier

Specifies a class to fill this role.

Multiplicity

The multiplicity field defines the maximum number of instances that can exist in this role at any given time.

Documentation

Use to describe this classifier role.

Files tab

A list of referenced files is provided here. The files list popup menu allows you to insert and delete references to files or URLs.

You can link external files to model elements for documentation purposes.

Association Role Specification

The Association Role Specification provides control over the properties of an association role in a collaboration diagram.

The Association Role Specification is a standard Specification dialog, with additional fields controlling the properties of the classifier role.

There are two tabs: the General Tab and the Files tab.

General Tab

Name

The name of the association role within the collaboration.

Stereotype

A stereotype label for the association.

Association

Specifies a class to fill this role.

Multiplicity

The multiplicity field defines the maximum number of instances that can exist in this role at any given time.

Documentation

Use to describe this association role.

Files Tab

A list of referenced files is provided here. The files list popup menu allows you to insert and delete references to files or URLs.

You can link external files to model elements for documentation purposes.

Creating State Diagrams 11

Contents

This chapter is organized as follows:

Creating Capsule State Machines

Capsules are one of the primary modeling elements in Rose RealTime. Complete executable code implementations are generated by the toolset for capsules.

Capsule behavior is defined through the State Diagram Editor.

There are three kinds of behavioral elements that may be added to a capsule behavior diagram:

- States
- Transitions
- Choice points

None of these elements are required. A capsule does not have to have any states or transitions. If the capsule has any interfaces (end ports) in its structure definition, then it must have a state machine to deal with events arriving on its interfaces.

Creating a complete capsule state diagram definition can consist of any of the following basic steps:

- Adding a State
- Adding a Choice Point

- Drawing Transitions Between States
- Defining State Transition Trigger Events
- Joining Transitions
- Creating Nested States

Using the State Diagram Editor

The state diagram editor is used to define the finite State machine for a class. The utility of the state diagram depends on the type of element it is specifying:

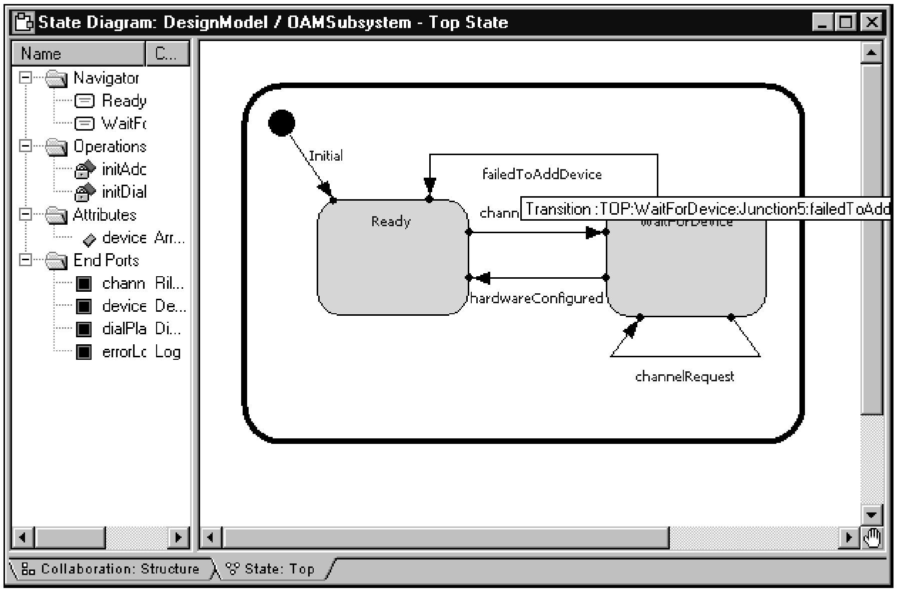
- For capsule classes, the state diagram will result in a complete code implementation generated for the class. The state diagram defines the majority of a capsule class implementation. The capsule class may also have operations defined on it, but the state diagram gives the capsule its asynchronous message processing capability.
- For protocols, the state diagram specifies the expected operation of any capsules that contain one of the protocol's roles. The protocol state diagram defines the allowable sequence of message inputs and outputs with respect to the protocol roles. There is no code generated for the protocol class behavior.
- For data classes, the state diagram captures the abstract behavior (often the abstract modes of operation) for the class. This does not result in any code being generated for the data class. The data class implementation is limited to the definitions of any attributes and operations specified through the Class Specification.

The state diagram consists of three parts: the diagram area, the navigator area, and the toolbox. Multiple State diagrams can exist in the same model.

Behavior elements, such as states and transitions, are added using the toolbox.

The window title bar shows the full name of the class.

Figure 38 State diagram editor



State Diagram Elements

The state editor window has tabs on the bottom to allow quick navigation to any nested states, and to the capsule structure editor. You can use the popup menu to navigate between diagrams, as well.

The state diagram allows you to create or edit the following elements:

- States
- State Transitions
- Choice Points
- Initial point and initial transition
- Junction points
- Final States

The state machine can be nested, allowing you to create hierarchical state machines. Hierarchical state machines maintain a state history. When a transition terminates on a hierarchical state, the history mechanism may be triggered to determine which substate becomes the active state.

Using the Navigator

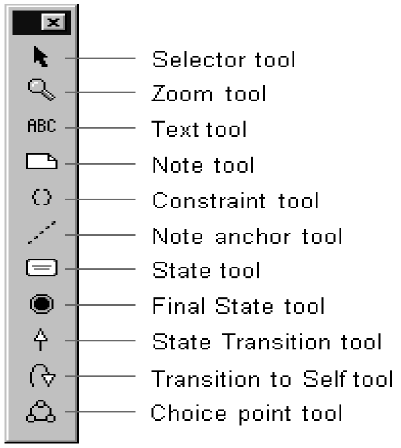
The Navigator area lists the states in hierarchical order, as well as operations, attributes, and ports (in state diagrams only).

Right-clicking on the items in the list provides a shortcut to many common operations, such as adding operations, attributes, and ports.

State Diagram Toolbox

The state diagram toolbox contains tools for adding elements to the state diagram. The toolbox is associated with the State Diagram Editor (see *Using the State Diagram Editor* on page 198).

Figure 39 State diagram toolbox



Selector tool

Use to select objects for moving, resizing, and so forth.

Zoom tool

Use to zoom in on a portion of the diagram. Click on the tool and then click on the part of the diagram you want to zoom in on.

Text tool

Use to add text anywhere in the structure diagram.

Note tool

Use to annotate the diagram with textual notes. This is useful for marking up the diagram with explanations, review comments, and so forth. You can drag and drop a diagram or external document from the browser onto a note. Notice that the name of the diagram or external document is underlined. If you double-click on the note, the diagram or external document is opened. You can undo and redo this command.

Constraint tool

Use to add UML constraints to the diagram. A constraint can be anchored to a view element by using the anchor tool. Currently, constraints do not have any semantic meaning to the tool. There are RRTEI APIs to add or remove, and enumerate constraints in a diagram.

Note anchor tool

Use to anchor a note to a particular element on the diagram.

State tool

Use to add a state to the diagram. Click on the diagram to place a new state at the selected location.

States are given default names, such as 's1', when initially drawn. To change the name, click on the state and hit the Backspace key to delete the default name, then type the new name.

Final State tool

Use to add a terminal state to the diagram. Click on the diagram to place a new final state at the selected location.

Transitions cannot be drawn initiating from a final state.

Capsule state diagrams cannot have a final state, so this tool is not displayed for capsules.

State Transition tool

Use to draw Transitions from one state to another, from a state to a branch, from a branch to a state, from a transition junction point on the superstate to a substate or to a transition exit point on the superstate, or from the initial point to an initial state.

Transition to Self tool

Use to draw a transition from a state back to itself. This can include self transitions on the outer state border, as well as on any substate.

Choice point tool

Used to add a branch point allowing a transition to branch to two alternate destination states.

State Specification

The state specification allows you to enter details about the state.

The state specification dialog contains the following tabs: General, Entry Actions, Exit Actions, Files.

Note: If this state is a top state, an initial point, or a final state on a data or protocol class, then it will not contain any **Entry Actions or Exit Actions tabs.**

General tab

Name

The name of the state. The state name appears on the state diagram, and will be part of the generated code for any capsule class. It will also be used in the verification of any sequence diagrams involving the capsule if the sequence diagram used as the specification contains state information.

Class

The class whose state machine this state is a part of.

Entry Actions / Exit Actions tabs

Code

A Code Editor used to enter the detail code that will be executed upon entry to or exit from the state.

This field may also be modified from the generated code and captured into the model using the Code Sync feature. For more information, see *Using Code Sync to Change Generated Code* on page 363.

Aggregating and Decomposing State Machines

You can create new superstates by aggregating several states, transitions and choice points. To aggregate several states into a new superstate, multiply select the states and choose **Parts > Aggregate**. A new state is created containing the selected states.

A superstate can be decomposed into its immediate substates by selecting **Parts > Decompose**.

Transition Specification

The transition specification is used to edit the properties of a state transition. There are up to four tabs: General, Triggers, Actions, and Files.

If the transition is an initial transition, or is not the originating segment of a joined transition, then the Triggers tab is not displayed.

General tab

Name

The name of the transition. If the transition is part of a capsule state diagram, the transition name will appear in the generated code for a capsule.

Internal

This check box indicates that a self-transition should not cause an exit from the state when triggered. The result is that when an internal transition is triggered, no exit or entry code is run.

Triggers Tab

Triggers List

The triggers list is used for Defining State Transition Trigger Events. The triggers list contains the list of individual trigger events. Each event consists of a port name, a signal or set of signals, and an optional guard condition. The Transition Events tab contains the list of events that can trigger the transition. The list is an ‘OR' list, meaning that the receipt of any one of the signals in the event list will cause the transition to fire. There is no ‘AND' definition of event triggers (since only one message is processed at a time).

To add new trigger events, right-click in the list area and select **Insert** from the popup menu. This brings up the Event Editor Dialog allowing you to select the port(s) and signal(s) that will act as trigger events.

Filter checkboxes provides options to display *Inherited* values, *Local* values and *Excluded* valuesT

Moving and Copying triggers

To move a trigger from one Specification sheet to another, drag and drop it. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

To copy a trigger from one Specification sheet to another, drag and drop it while holding down the Ctrl key. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

Actions Tab

Code

Contains a Code Editor for defining detailed action code.

This field may also be modified from the generated code and captured into the model using the Code Sync feature. For more information, see *Using Code Sync to Change Generated Code* on page 363.

For capsules, the transition action code will be output as part of the generated code, and the code will be executed when the transition is triggered at run-time.

Transition actions defined in state diagrams for protocols or regular (non-capsule) classes is not generated or executed. It is for information purposes only.

Files Tab

The Files tab allows for linking external files to the transition.

Choice Point Specification

The Choice Point Specification contains three tabs: General, Condition, and Files.

General Tab

Contains standard specification dialog items.

Condition Tab

Contains a Code Editor for entering the code that determines which branch of the transition will be taken. The code must return a true or false value (false is zero and true is non-zero).

This field may also be modified from the generated code and captured into the model using the Code Sync feature. For more information, see *Using Code Sync to Change Generated Code* on page 363.

Files Tab

A list of referenced files is provided here. The files list popup menu allows you to insert and delete references to files or URLs.

You can link external files to model elements for documentation purposes.

Initial State Specification

The Initial State Specification contains two tabs: General and Files.

General Tab

Name

The default name for the initial state is Initial and should not be changed.

Class

A non-editable field indicating the class whose state machine the initial state is part of.

Documentation

A description of the initial state.

Files Tab

A list of referenced files is provided here. The files list popup menu allows you to insert and delete references to files or URLs.

You can link external files to model elements for documentation purposes.

Junction Point Specification

The dialog shows information about the junction point. See the *Modeling Language Guide* on junction points and history for more information on these selections.

There are only two tabs: General and Files.

General Tab

Name

A name for the junction point. Most junction points are given automatically generated names.

Continuation

This selection specifies the semantics for how the state history will be used when there is no continuing transition. There are three options:

- Default - specifies that the default (initial) transition should be run.
- History - specifies that the state should return to shallow history.
- Deep History - specifies that the state should return to deep history, meaning that all substates also return to history. This is the behavior for all capsule state machines, so it is automatically selected.

Note: The default for capsule state machines is to always go to deep history, so deep history is automatically selected for capsule states, and the selections are grayed out.

Externally Visible

This check box indicates whether the junction point is visible on the outside of the state boundary.

Files Tab

A list of referenced files is provided here. The files list popup menu allows you to insert and delete references to files or URLs.

You can link external files to model elements for documentation purposes.

Event Editor Dialog

The event editor dialog is used to define triggering events for transitions in capsule state diagrams. The event editor is accessed from the Events tab of the transition specification dialog.

The event editor contains:

- a ports list
- a signals list
- a guard code area

The ports list contains a list of end ports in the capsule's collaboration diagram. Only end ports that have In signals are listed as they are the only ones capable of receiving messages.

The guard code may also be modified from the generated code and captured into the model using the Code Sync feature. For more information, see *Using Code Sync to Change Generated Code* on page 363.

To specify the trigger event:

1. Select a port check box to display the list of signals that can be received on that port.

   Multiple ports from the ports list can be selected, but when selecting multiple ports, the signals are only displayed if all ports share the same protocol.

   The signals list displays signals that can be received by the currently selected end port, as well as a default wild card selection indicated by a '*'.

2. Select one or more signals from the signal list.

   Multiple signals from the signals list can be selected. Any one of the selected signals will act as a trigger for the transition.

Adding a State

States can be added by clicking on the state tool in the state diagram toolbox, and then clicking on the state diagram where you want to add a state.

Alternatively, you can add states through the navigator area of the state diagram editor.

If you have state entry or exit actions to define, use the state specification dialog or Code window.

Adding a Choice Point

To draw a choice point, click on the choice point tool from the state diagram toolbox and then click on the diagram where you want the choice point added.

The choice point can be rotated by grabbing one of its handles and turning. The true and false branches can be flipped using the popup menu.

Once you have added a choice point, you should define the condition for the choice point.

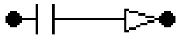
Drawing Transitions Between States

The transition tool is used to draw transitions.

Transitions are drawn originating from states, transition join points or choice points and terminating on those same elements.

To draw a transition, click on the transition tool, click on the originating element for the transition and drag the transition on to the terminating element.

When a new transition is drawn, the transition does not usually have a triggering event. Transitions with no trigger event are shown with a broken line:



Transitions containing code are shown with the arrowhead filled in black.

See *Drawing the Initial Transition* on page 209.

Specifying the Transition

Once you have drawn a transition, you can specify the transition details. The details of a transition include the trigger event(s), and the action code. These are specified through the Transition specification dialog.

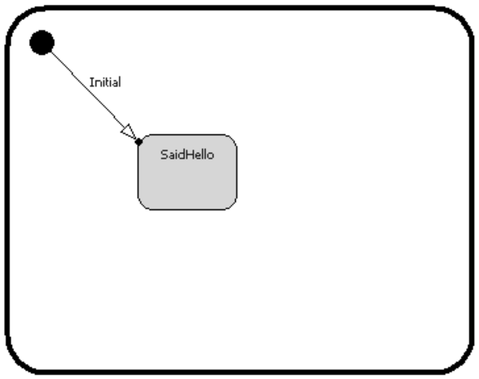
The trigger event and action code for capsule state machines result in generated code as part of the capsule implementation. No code is generated for the trigger event and action for other class state machines.

Drawing the Initial Transition

To draw an initial transition in a state diagram editor:

1. Click on the transition tool in the state diagram toolbox.
2. Click on the initial point in the diagram and drag the transition on top of the target state. The initial point is the black circle that appears in the top-left corner of the diagram.

Figure 40 Initial transition



The initial transition has a default name of 'Initial'. You can change the name by selecting the label and typing in it.

Defining State Transition Trigger Events

State Diagrams

To define a new state transition trigger event:

1. Open the **State Transition Specification** from the capsule State Diagram editor (double-click on the state transition).
2. Select the **Events** tab.
3. Right-click in the event list area.

4 Select **Insert** from the popup menu.

The Event Editor Dialog appears.

To define a new event in a capsule:

1 Click on the check box for the port and signal items to be included in the event.

2 The chosen items have a check mark next to them. Deselecting the check box removes items from the event definition.

Note: A state transition trigger event can have more than one signal selected on a port, and can have more than one port selected, though the signals list only shows the signals that are common in the protocols of the two ports in that case. To trigger a transition on signals on different ports, use multiple trigger events. A wild card trigger is available (*) in the signals list which triggers a transition if any of the valid input signals of the currently selected ports is encountered.

To define a new event in a protocol:

1 Click on the check box for the signal items to be included in the event.

2 The chosen items have a check mark next to them. Deselecting the check box removes items from the event definition.

Defining a new event in a data class

To define a new event in a data class, specify the name of the event.

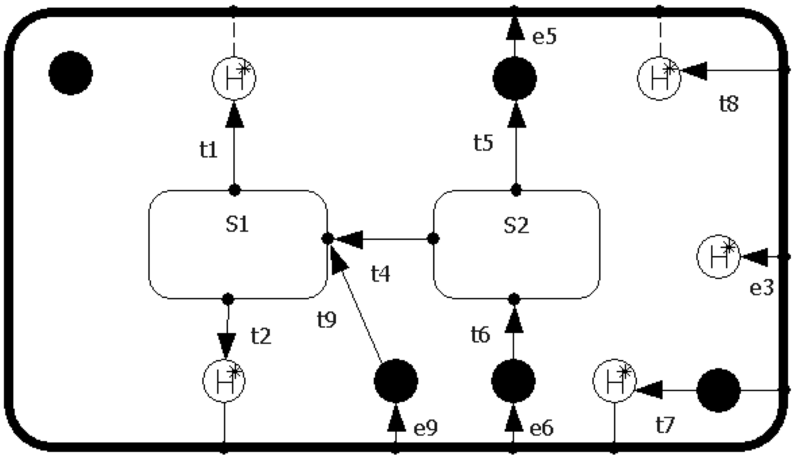
Joining Transitions

Transitions terminating on a superstate can be joined to transitions inside the state to terminate directly on a substate. Similarly, transitions inside a hierarchical state can be joined to transitions leaving the superstate. The points where a transition begins or ends are represented inside the state with join points. Join points may be dark circles or light circles.

To connect a new transition to an existing transition, select the transition tool and draw the transition starting from or terminating on a join point.

To join two existing transitions, select one of the transitions and move it so that the end point lands on the beginning point of the other transition, or so that the beginning point lands on the end point of the other transition.

Figure 41 Joined transitions



Creating Nested States

Nested states are created in one of two ways:

- During state creation, select the state icon from the diagram toolbox and place over a targeted superstate.
- After a state has been placed on a diagram, use the drag and drop technique to place it over a targeted superstate.

The border of the target superstate becomes bold as the nested state moves over it. Once the nested state is dropped on the superstate, the boundaries of the superstate may grow to accommodate the nested state. If the cursor is positioned over more than one state at the same time, the state at the deepest level of nesting is considered the target superstate. Multiple states can be selected and nested as a group.

When a hierarchical state is created using one of these methods, the target superstate becomes see-through. The **Edit Inside** menu item from the state's popup menu is toggled on.

Nesting is determined completely by cursor position. Once the cursor is moved outside the target state, no nesting occurs. The bold display of the target state's border serves as an indicator for nesting. States can overlap without nesting

Creating Sequence Diagrams 12

Contents

This chapter is organized as follows:

Creating a Sequence Diagram

Sequence diagrams can be created in both the Use Case View and the Logical View. A Sequence diagram shows a particular interaction scenario among roles or instances in the model. A Sequence diagram is created from a collaboration diagram, which shows a general interaction pattern among roles or instances. (This includes capsule structure diagrams.) That is, the collaboration diagram shows the general pattern of associations among roles or instances, which is often created first and usually evolved in parallel with the associated Sequence diagrams. The Sequence diagram shows a specific sequence of interactions among roles or instances for a particular scenario.

Sequences can also be associated with protocols. Protocols, which are currently always binary, do not show their collaboration because it is fixed.

Creating a New Diagram

There are four ways to create a new Sequence diagram: from the

- Model browser
- structure diagram browser
- structure or collaboration diagram
- trace window

From the Browser

To create a new Sequence diagram from the browser:

1. Select or create a collaboration diagram, capsule structure, protocol, package, use case, or class.
2. Right-click on the element in the model browser.
3. Select **New > Sequence Diagram** from the popup menu.
4. Enter the name of the Sequence diagram.

From the Structure Diagram Browser

To create a Sequence diagram from the structure diagram browser:

1. Right-click on the Sequence Diagrams folder in the Structure diagram browser.
2. Select Add New Sequence Diagram.

From the Collaboration or Structure Diagram

To create a new Sequence diagram from the collaboration or structure diagram:

1. Select or create a collaboration diagram, capsule structure or protocol.
2. Multi-select the model elements from the diagram to pre-populate the Sequence diagram.
3. Click in the diagram background and select **New > Sequence Diagram** from the popup menu.
4. Enter the name of the Sequence diagram.

Editing a Diagram

To edit a Sequence diagram:

1. Double-click on the diagram in the model browser.

   The Sequence Diagram editor appears.

   You can also select the **Open** menu item in the context menu for the Sequence diagram in the model browser.
2. Place capsules or class roles or instances in the diagram by dragging the class or capsule from the model browser, or by using the tools in the Sequence diagram toolbox.

3 Sequence diagrams are by default empty when created, except when created from protocols. Instances can be added to the Sequence diagram by dragging roles from the roles navigator within the structure browser. An instance representing the containing capsule class can also be added by dragging that class from the browser into the Sequence diagram.

   A Sequence diagram can also be pre-populated with instances by selecting the desired set of roles in the structure or collaboration diagram and then selecting the **Create Sequence Diagram** menu item from the background menu of the diagram.

   For structure diagrams, you can also optionally select the border if you want to show interactions between the capsule and its roles.

   Note: There are no borders to select in collaboration diagrams.

4 Draw messages among instances using the toolbox.

Adding Instances

The instances or roles in the Sequence diagram should generally be drawn from the instances or roles in the collaboration diagram. Collaboration and Sequence diagrams can show interactions among object instances or among roles. In most cases, they are more useful demonstrating interactions among roles, because a role demonstrates a part played in the scenario, which could be played by more than one instance.

Sequence diagrams are not automatically populated with instances. The instances must be added, either by dragging classes from the model browser or by using the instance tool from the toolbox.

Instances added using the instance tool are unspecified by default, which means that the tool does not know what actual design element the instance corresponds to. You can specify a role or create a new one using the drop-down model box that appears when you create a new instance. The Sequence diagram is not a complete specification until all instances are mapped to actual design elements. Use the Path field on the interaction instance specification dialog to specify which design instance the sequence instance maps to.

A Sequence diagram created under a protocol is pre-populated with two instances: base and conjugate. These instances cannot be removed and other instances cannot be added.

Defining Messages

Messages are created between instances or between instances and the environment on the diagram to show interaction. Messages can represent: asynchronous sends, synchronous sends, function calls, instantiations, destructions, FOC (Focus of Control) blocks, local states, local actions, and coregions. There is a separate message tool for each of these.

Specifying Message Details

Sequence diagrams act as design specifications. A complete Sequence diagram within a capsule structure can be verified by model execution. In order to verify a Sequence diagram, the sequence instances must be mapped to design instances and the send messages among capsule instances must be specified. The specification of the message includes identifying the source and destination ports, signal names, and possibly data types.

Cloning a Sequence Diagram

To clone a Sequence diagram:

1 In the browser, select the Sequence Diagram you want to clone.

2 Again in the browser, Control-drag it onto a Collaboration.

Note that you can select the same Collaboration that contains the original Sequence Diagram.

A new Sequence Diagram is created for you under this Collaboration.

Using the Sequence Diagram Editor

A Sequence diagram is a graphical view of a scenario that shows an object interaction in a time-based sequence. Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.

A Sequence diagram has two dimensions: vertical placement represents time and horizontal placement represents different objects.

Elements of the Sequence diagram, such as instances and messages are added using the toolbox.

The window title bar shows the full name of the Sequence diagram.

The following specification dialogs can be accessed from elements on the Sequence diagram:

- Instance Specification dialog
- Interaction Specification dialog
- Send Message Specification dialog
- Call Message Specification dialog
- Create Message Specification dialog
- Return Message Specification dialog
- Destroy Message Specification dialog
- Local State Specification dialog
- Coregion Specification dialog
- Local Action Specification dialog
- Reply Message Specification dialog

The popup menu for the Sequence diagram editor includes a validate command that opens the validation dialog. It also contains an **Auto-generate FOC** entry, which controls whether new send or call messages automatically get an FOC (Focus of Control) - and a return message, if appropriate - when they are created.

Note: If you are not interested in message activation, turn the **Auto-generate FOC** entry off. This simplifies the diagram display considerably.

Opening Collaboration Diagrams

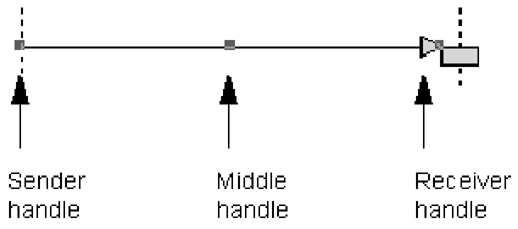
To open the Collaboration diagram associated with a particular Sequence diagram, select **Open Collaboration Diagram** from the popup menu for the background of the Sequence diagram.

Reorienting Messages

You can reorient messages to make semantic changes in the diagram, for example, to change the sender or receiver for a message.

Using the sender or receiver handles, you can change the sender or receiver. Using the Re-order handle (Middle handle), you can change the order of this message on the sender and receiver.

Figure 42 Message handles



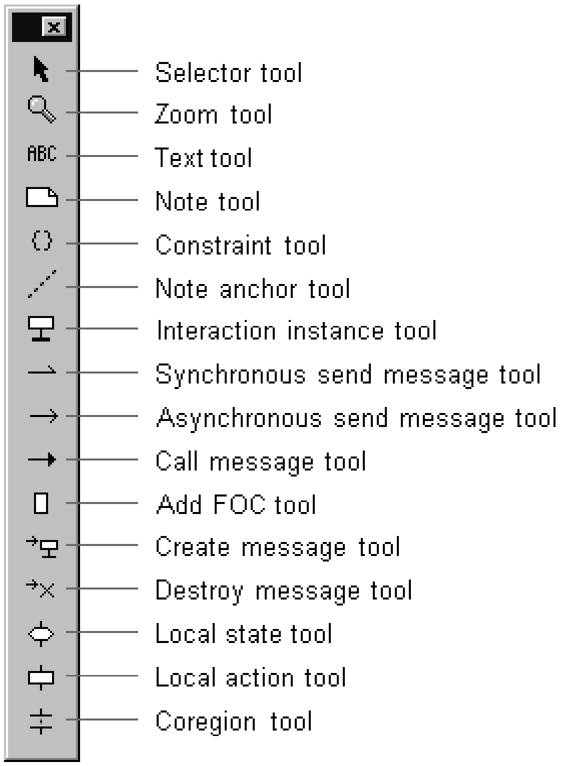
Moving Messages

You can move messages in a Sequence diagram to create more or less space between them. Message movement is restricted to avoid accidentally altering the semantics of the diagram. When moving a message, do not drag it using one of the handles.

Sequence Diagram Toolbox

The Sequence diagram toolbox contains tools for adding elements to the Sequence diagram.

Figure 43 Sequence diagram toolbox



Selector tool

Use to select objects for moving, resizing, and so forth.

Zoom tool

Use to zoom in on a portion of the diagram. Click on the zoom tool and then click on the area of the diagram to zoom in on.

Text tool

Use to add text anywhere in the diagram.

Note tool

Use to annotate the diagram with textual notes. This is useful for marking up the diagram with explanations, review comments, and so forth. You can drag and drop a diagram or external document from the browser onto a note. Notice that the name of the diagram or external document is underlined. If you double-click on the note, the diagram or external document is opened. You can undo and redo this command.

Constraint tool

Use to add UML constraints to the diagram. A constraint can be anchored to a view element by using the anchor tool. Currently, constraints do not have any semantic meaning to the tool.

Note anchor tool

Use to anchor a note or constraint to a particular element on the diagram.

Interaction instance tool

Use to add an object instance to the diagram. Instances are added along the horizontal axis at the top of the diagram.

Synchronous send message tool

Use to add a message between two instances. Synchronous send messages block the sender waiting for a return (like a function call). This corresponds to the 'invoke' operation.

Asynchronous send message tool

Use to add a message between two instances. Asynchronous messages do not block the sender.

Call message

Use to add a message between two instances. Call messages are like function calls, so the sender is blocked waiting for a return.

Add FOC

Use to add an FOC (Focus of Control) block to the selected message. Select the tool and click on the send or call message you want to add an FOC to.

Note: If the selected message can have a reply or return (that is, a synchronous send or call), it is automatically generated.

Create message tool

Use to indicate that one instance creates another instance dynamically. The create message indicates the moment of creation of the destination instance. The new instance opens at the end of the create message.

Destroy message tool

Use to indicate that one instance destroys (deletes) another instance dynamically. The destroy message indicates the moment of destruction of the destination instance.

Local state tool

Use to indicate a state change in one of the instances. Click on one of the instances in the diagram to place a new state at the selected location.

Local action tool

Use to indicate an action carried out by one of the instances. The action represents a significant activity or operation being performed by the instance at that time.

Coregion tool

Use to indicate a set of events/messages whose ordering is undefined. That is, although the messages appear in a particular order (as indicated by their vertical placement on the instance line), the actual run-time ordering may vary.

Interaction Instance Specification

The interaction instance specification has information about an instance on a Sequence diagram.

It contains two tabs: General and Files.

General Tab

Name

Specifies the name of this instance. Instances are unnamed by default. The name is displayed as part of the instance label on the diagram.

Path

Identifies the role path for an instance in a collaboration. The pull-down menu allows you to choose from the available roles in the immediate collaboration associated with the Sequence diagram.

For Sequence diagrams under structure diagrams, it is also possible to show interactions between the capsule and its roles. To do this, pick the capsule from the path pull-down menu.

Stereotype

Specifies the (optional) stereotype of this instance.

Documentation

Use the Documentation field to describe this instance.

Files Tab

The Files tab allows for linking external files.

Interaction Specification

The Interaction Specification is used to describe interactions on a Sequence diagram.

It has two tabs: General and Files

General Tab

Name

Specifies the name of the interaction.

Stereotype

Specifies the (optional) stereotype of this instance.

Documentation

Use the Documentation field to describe this interaction.

Files Tab

The Files tab allows for linking external files.

Local Action Specification

The Local Action Specification is used to describe actions in Sequence diagrams.

It contains General, Detail, and Files tabs.

General Tab

Name

A name for the local action, which is displayed on the Sequence diagram.

Stereotype

Specify a stereotype for the local action.

Detail Tab

Sender

Non-editable field with the name of the instance where the local action is defined.

Receiver

Not applicable to a local action.

Time

Capture the time of the action.

Effect

A textual description of the effect of the local action.

Local State Specification

The Local State Specification contains General, Detail, and File tabs.

General Tab

Name

A name for the local state. The name is displayed on the Sequence diagram.

Stereotype

Specify a stereotype for the local state.

Detail Tab

Sender

Non-editable field with the name of the instance where the local state is defined.

Receiver

Not applicable for the local state

Time

Capture the time of the state change.

Message Specification

There are several different kinds of messages, but all have similar controls in the Message Specification dialog.

The Message Specification dialog contains the following tabs: General, Detail, Port Detail (only for Send messages), and Files.

General Tab

Name

A name for the message. The name is displayed on the Sequence diagram.

Stereotype

Specify a stereotype for the message.

Documentation

Use this field to enter documentation on this element.

Detail Tab

Sender

Non-editable field with the name of the instance where the message originated.

Receiver

Non-editable field with the name of the instance where the message ends.

Time

Capture the time that the message was sent.

Data

A textual description of the message data.

Port Detail Tab

This tab is only significant for messages between capsule roles. The data on this tab can be filled in by selecting from the pull-down menus, which include data from the collaboration diagram.

If the fields on this tab are filled in for a Sequence diagram that acts as a behavior specification, then the data can be compared to the actual data captured from a run-time execution trace to verify the behavior at execution time against the specification.

From Port

The name of the port on the sender capsule.

To Port

The name of the port on the receiver capsule.

Signal

The name of the signal from the ports' protocol.

Delivered

Capture the time the message was delivered to the receiver.

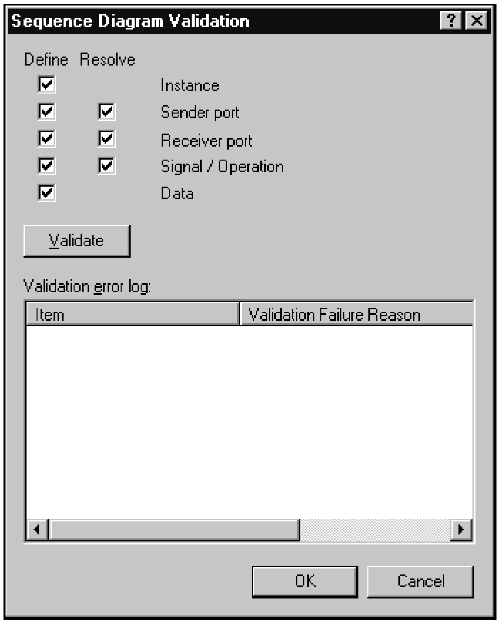
Priority

The priority at which the message is sent. (Applies only to an Asynchronous Send Message.)

Sequence Validation Dialog

A Sequence diagram can be used as a specification of the interaction among object roles and/or instances. Sequence diagrams are very useful as specifications of design intent to be checked against actual model execution results.

Figure 44 Sequence Diagram Validation dialog



The Validation Dialog allows you to check the Sequence diagram specification for missing elements. It provides control over what aspects of the Sequence diagram should be checked for completeness.

A list of check boxes to control what is checked during verification.

- Instance - checks that the path of each instance in the interaction is defined.
- Sender port - checks that the sender port names in the sequence are defined and, possibly, resolved to an existing port.
- Receiver port - checks that the receiver port names in the sequence are defined and, possibly, resolved to an existing port.
- Signal/Operation - checks that the signal names for send or operation names for a call are defined and, possibly, resolved to an existing signal.
- Data - checks that the data types of all messages in the sequence are defined.
- Validate button - performs the actual validation. Results of the validation appear in the error log.

Validation Error Log

Contains the results of the validation. Each item in the list indicates an undefined or unresolved sequence element.

Focus of Control

Focus of Control (FOC) is an advanced notational technique that enhances Sequence diagrams. This technique shows the period of time during which an object is performing an action, either directly or through an underlying procedure.

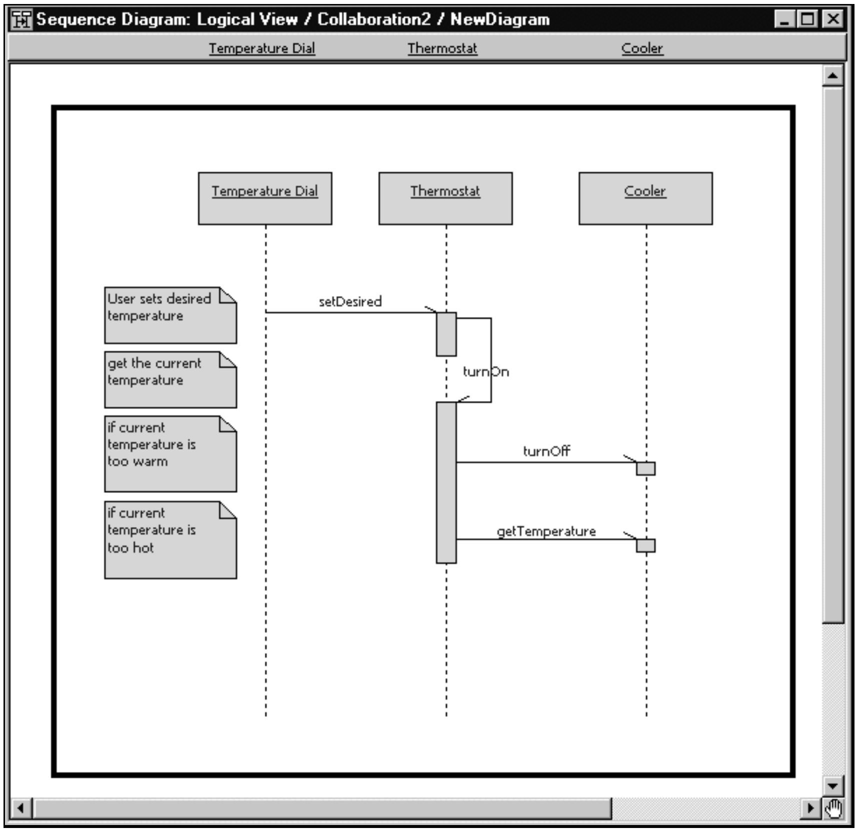
FOC is portrayed through narrow rectangles that adorn lifelines (the vertical lines descending from each object). The length of a FOC indicates the amount of time it takes for a message to be performed. When you move a message vertically, each dependent message moves vertically as well. Also, you can reorient a message vertically off the source FOC to make it detached and independent.

Activators

Messages that originate from an FOC are said to have been activated by the message that started that FOC.

A Sequence diagram with FOC notation and scripts follows:

Figure 45 Focus of Control Diagram Example



User sets desired temperature

get the current temperature

if current temperature is too warm

if current temperature is too hot

Coloring Focus of Control

To help distinguish a particular FOC from other items in a Sequence diagram, you can fill a FOC with a color.

To color a FOC:

1. Select the FOC you want to color.
2. Click Diagram Object Properties from the **Edit** menu and then click **Fill Color**.
3. Click on the color you want to make the selected FOC.
4. Click **OK**.

Defining Capsules and Classes

13

Contents

This chapter is organized as follows:

Creating a Class

Classes can be created in either the Logical View or the Use Case View of the Model browser.

To create a class:

1. Right-click on the Logical View package in the Model browser (or on the Use Case View package).
2. Select **New >Class** from the menu.

   A new class is created with a default name of 'NewClass1'.
3. Type over the name to change it.

Creating New Attributes

To create a new attribute on a class:

1 Right-click on the class in the model browser.

2 Select **New > Attribute** from the menu.

3 Double-click on the attribute to open the Attribute Specification Dialog to set the name, class or type, code generation properties (for example, virtual), and so forth.

Alternatively:

1 Open the Class Specification.

2 Select the **Attributes** tab.

3 Right-click and select **Insert** from the popup menu.

You can reorder attributes in the Specification dialog using drag-and-drop. You can undo and redo this action.

Creating New Operations

To create a new operation on a class:

1 Right-click on the class in the Model browser.

2 Select **New > Operation** from the menu.

3 Double-click on the operation to open the Operation Specification to set the name, parameters, return values, and so forth.

Alternatively:

1 Open the Class Specification.

2 Select the **Operations** tab.

3 Right-click and select **Insert** from the popup menu.

4 Double-click on the new operation and use the Operation Specification to specify the operation details.

You can reorder operations in the Specification dialog using drag-and-drop. You can undo and redo this action.

Class Specification

The class specification is used to edit the properties of a class. The dialog provides access to all member attributes and operations as well.

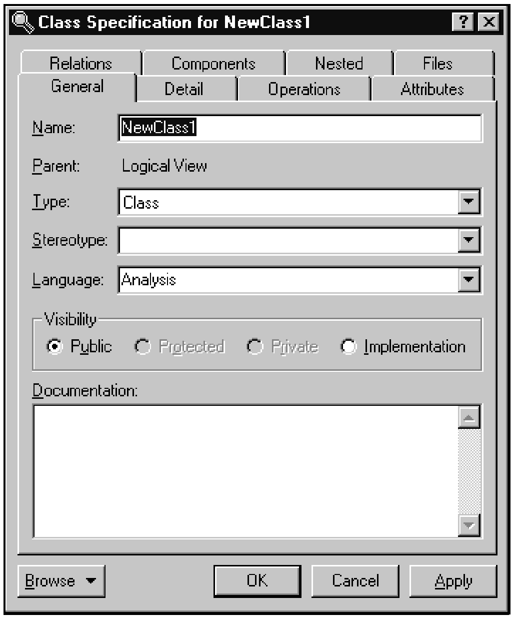
Class Specification Content

The class specification contains the following tabs:

- Class Specification - General tab
- Class Specification - Detail tab
- Class Specification - Operations tab
- Class Specification - Attributes tab
- Class Specification - Nested tab
- Class Specification - Components tab
- Class Specification - Relations tab
- Class Specification - Files tab

Class Specification - General tab

Figure 46 Class Specification - General tab



Name: NewClass1
Parent: Logical View
Type: Class
Stereotype:
Language: Analysis
Visibility: [✓ Public] Protected Private Implementation

Name

The name of the class you opened the Specification for.

Parent

The parent the class belongs to (its package, or class in the case of a nested class) is displayed in this static field.

Type

Your **Type** choices include:

- Class
- Parameterized class (no code generation available)
- Instantiated class (no code generation available)
- Utility class
- Parameterized utility class (no code generation available)
- Instantiated class utility (no code generation available)
- Metaclass (no code generation available)

Stereotype

A stereotype represents the subclassification of an element. It represents a class within the UML metamodel itself, that is, a type of modeling element. Some stereotypes are already predefined, but you can also define your own to add new kinds of modeling types.

Stereotypes can be shown in the browser and on diagrams. The name of the stereotype may appear in angle brackets <<>>, depending on the settings found in either the Diagram or Browser tabs of the Options dialog box located under the **Tools** menu. Refer to the Stereotype chapter for more information on stereotypes.

To show stereotypes on the diagrams, click **Options** from the shortcut menu and click Stereotype Name or Stereotype Icon. Stereotype Name displays the name in angle brackets (that is, <<stereotype>>). Stereotype Icon displays the graphical representation.

Language

Select the implementation language for the class from the available languages. The analysis selection indicates that no code will be generated for the class.

Visibility

The Visibility field specifies how a class and its elements are viewed outside of the defined package.

Select:	To Indicate:
Public	The element is visible outside of the enclosing package and you can import it to other portions of your model. Operations are accessible to all clients.
Protected	The element is accessible only to subclasses, friends, or the class itself.
Private	The element is accessible only to its friends or to the class itself.
Implementation	The element is visible only in the package in which it is defined. An operation is part of the implementation of the class.

The Visibility field can be set only in the specification. No special annotation is related to access control properties.

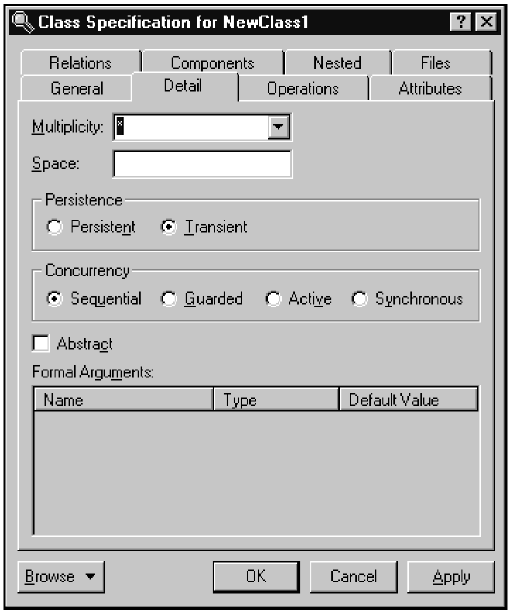
To change the visibility type for the class, click on the appropriate option in the visibility field. You can display the implementation visibility in the component compartment. You can display visibility in an icon through the shortcut menu.

Documentation

Use this field to enter documentation on this element.

Class Specification - Detail tab

Figure 47 Class Specification - Detail tab


Class Specification for NewClass1

Multiplicity: *

Space:

Persistence: Persistent [✓ Transient]

Concurrency: [✓ Sequential] Guarded Active Synchronous

Abstract: ☐

Formal Arguments:

Name	Type	Default Value

Multiplicity

The Multiplicity field specifies the number of expected instances of the class. In the case of relationships, this field indicates the number of links between each instance of the client class and the instance of the supplier. See Cardinality Options for more information.

Space

Use the Space field to specify the amount of storage required by objects of the class during execution.

Persistence

Persistence defines the lifetime of the instances of a class. A persistent element is expected to have a life span beyond that of the program or one that is shared with other threads of control or other processes.

The persistence of an element must be compatible with the persistence that you specified for its class. If a class persistence is set to Persistent, then the object persistence is either persistent, static or transient. If a class persistence is set to Transient, then the object persistence is either static or transient.

You can set the persistence only through the specification. This field is inactive for class utilities, parameterized class utilities, and instantiated class utilities.

To set the persistence, click on the applicable option in the Persistence field. You can display the persistence in the diagram by selecting **Show Persistence** from the popup menu.

Concurrency

This field denotes the semantics in the presence of multiple threads of control. The Concurrency field shows the concurrency for the elements of a class. The concurrency of an operation should be consistent with its class.

Type Description:

- Sequential (default) - The semantics of the class are guaranteed only in the presence of a single thread of control. Only one thread of control can be executing in the method at any one time.
- Guarded - The semantics of the class are guaranteed in the presence of multiple threads of control. A guarded class requires collaboration among client threads to achieve mutual exclusion.
- Active - The class has its own thread of control.
- Synchronous - The semantics of the class are guaranteed in the presence of multiple threads of control; mutual exclusion is supplied by the class.

The Concurrency field is inactive for class utilities, parameterized class utilities, and instantiated class utilities.

To change the concurrency, click on an applicable option button in the Concurrency field. You can display the concurrency in the class diagram by selecting **Show Concurrency** from the popup menu.

Abstract

The Abstract field identifies a class that serves as a base class. An abstract class defines operations and states that will be inherited by subclasses. This field corresponds to the abstract class adornment displayed inside the class icon.

To toggle the abstract adornment, click on its check box.

When you click Abstract, the abstract class adornment is displayed in the lower left corner of the class icon. You can change the abstract class adornment only through the specification.

The Abstract field is inactive for metaclasses, class utilities, parameterized class utilities, and instantiated class utilities.

Formal Arguments

In the Parameterized Class or Parameterized Class Utility specification, the formal, generic parameters declared by the class or class utility are listed.

In the Instantiated Class or Instantiated Class Utility specification, the actual arguments that match the generic parameters of the class being instantiated are listed.

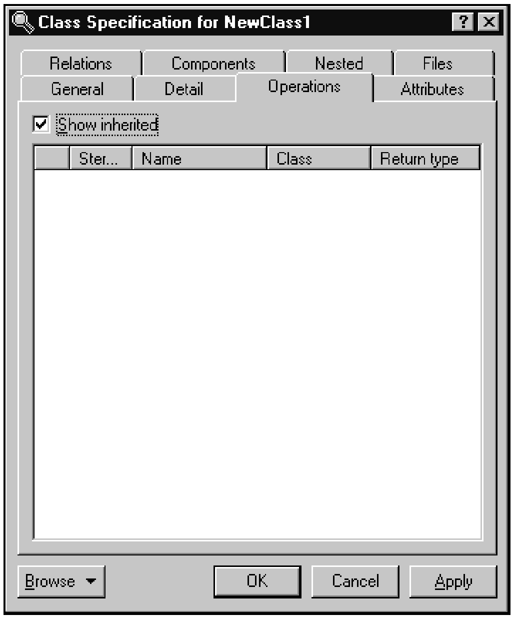
You can add, update, or delete parameters only through the Class specification. This field applies only to parameterized classes, parameterized class utilities, instantiated classes, and instantiated class utilities.

To define the parameters for a class, position the pointer within the Parameters field and click **Insert** from the shortcut menu or press the insert key.

Parameters are displayed on class diagrams.

Class Specification - Operations tab

Figure 48 Class Specification - Operations tab



	Ster...	Name	Class	Return type

Operations denote services provided by the class. Operations are methods for accessing and modifying Class fields or methods that implement characteristic behaviors of a class.

The Operations tab lists the operations that are members of this class. The actual definition of the operation is accessible from the Operation Specification.

The operations are listed with the following fields:

- Visibility Adornment (Unlabeled); the visibility of the operation is indicated with an icon. Following are the visibility options:
  - Public - the operation is accessible to all clients.
  - Protected - the operation is accessible only to subclasses, friends, or to the class itself.
  - Private - the operation is accessible only to the class itself or to its friends.
  - Implementation - the operation is accessible only by operations of this class.
- Stereotype - displays the name of the stereotype.
- Signature - displays the name of the operation.
- Class - identifies which class defines the operation.
- Return Type - identifies the type of value returned from the operation.

The **Operation** tab is active for all class types. In the class diagram, you can display operation names in the class compartment.

Show Inherited

Click this option to see operations inherited from other classes. If there is no check mark in this field, you can view only operations associated with the selected class.

Note: Rose RealTime allows you to directly modify any operation shown in the operations list by displaying the operations specification dialog. You should be careful when modifying base class operations for it may have implications on other elements in your model which reference or are subclassed from the base class.

Creating New Operations

To enter an operation in the Class Specification, select **Insert** from the popup menu. A new operation with a default name is added to the operations list.

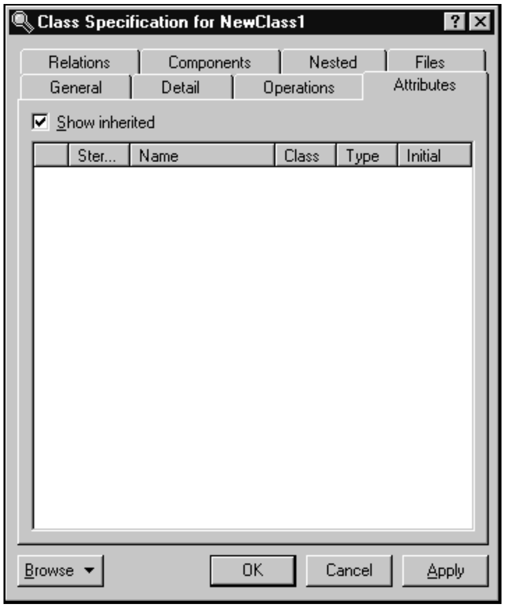
Moving and Copying Operations

To move an operation from one Specification sheet to another, drag and drop it. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

To copy an operation from one Specification sheet to another, drag and drop it while holding down the Ctrl key. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

Class Specification - Attributes tab

Figure 49 Class Specification - Attributes tab



The UML asserts that attributes are data values (string or integer) held by objects in a class. Thus, the Attributes tab lists attributes defined for the class. The attribute definition can be modified through the Attribute Specification Dialog.

Note: Attributes and relationships created using this technique are added to the model, but do not automatically appear in any diagrams. That is, adding an attribute affects the code generation for the class and a compilation dependency between the class of the container and the class of the attribute, but these relationships are not graphically visible in the model.

The descriptions for each field follow:

- Visibility Adornment (Unlabeled):
  - Public - the attribute is publicly visible, and is accessible to all clients.
  - Protected - the attribute may be accessed only by subclasses, friends, or by operations of this class.

  - Private - the attribute is accessible only by the class itself or by its friends.
  - Implementation - the attribute is accessible only by other operations in this class.
- Stereotype - displays the name of the stereotype.
- Name - displays the name of the attribute.
- Class - identifies where the attribute is defined.
- Type - this can be a class or a traditional type, such as int.
- Initial - displays the initial value of an object.

The Attribute tab is active for all class types.

Show Inherited

Click this option to see attributes inherited from other classes. If there is no check mark in this field, you can view only attributes associated with the selected class.

Note: Rose RealTime allows you to directly modify any attribute shown in the attributes list by displaying the attribute specification dialog. You should be careful when modifying base class attributes for it may have implications on other elements in your model which reference or are subclassed from the base class.

Creating New Attributes

You can add an attribute relationship by selecting **Insert** on the popup menu or by pressing the insert key. A new attribute with a default name is added.

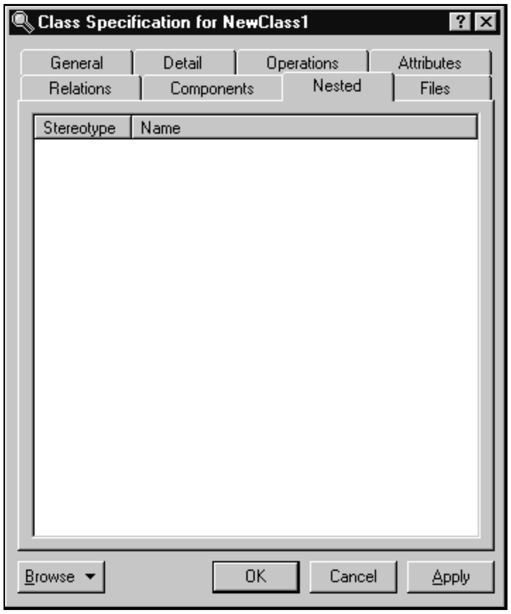
Moving and Copying Attributes

To move an attribute from one Specification sheet to another, drag and drop it. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

To copy an attribute from one Specification sheet to another, drag and drop it while holding down the Ctrl key. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

Class Specification - Nested tab

Figure 50 Class Specification - Nested tab



A nested class is a class that is enclosed within another class. Classes may contain instances of, inherit from, or use a nested class.

Enclosing classes are referred to as parent classes, and a class that lies underneath the parent class is called a nested class.

A nested class is typically used to implement functionality for the parent class. In many designs, a nested class is closely coupled to the parent class and is often not visible outside of the parent class. For example:

Think of your computer as a parent class and its power supply as a nested class. While the power supply is not visible outside the computer, the task it completes is crucial for the overall functionality of the computer.

Moving and Copying Nested Classes

To move a Nested class from one Specification sheet to another, drag and drop it. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

To copy a Nested class from one Specification sheet to another, drag and drop it while holding down the Ctrl key. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

To add a Nested Class from a Class Specification:

1 Create and name a class.

2 Display the Class Specification.

3 Click on the Nested tab.

4 Right-click to display the shortcut menu, then click **Insert**.

A nested class entry with a default class name is inserted.

To display a nested class:

1 On the **Query** menu, select **Add Classes**.

2 Select the nested class and place it in the Selected Classes list box.

You can undo and redo the addition of nested classes.

To delete a Nested Class from a Class Specification:

1 Select the nested class from the Nested tab in the Class Specification.

2 Right-click on the class to display the popup menu.

3 From the popup menu, select **Delete**.

Or, use the following steps to delete a nested class:

1 Select the name of the nested class from the Nested Classes list on the Nested Classes tab.

2 Press the Delete key.

If you delete a nested class that is also a parent to other nested classes, all the nested classes are deleted.

You can undo and redo the deletion of nested classes.

Note: When you attempt to delete a nested class from a Class Specification, a warning dialog appears to verify the deletion.

Relocating Nested Classes from the Browser to a Specification

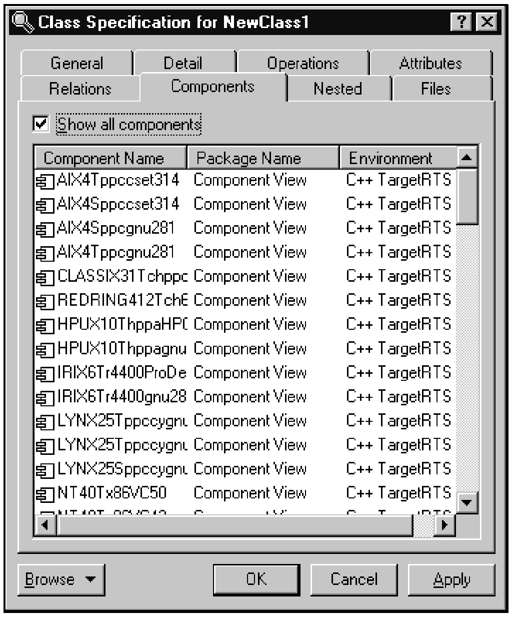
Classes and Nested Classes can be moved from the browser to the Class Specification Nested tab. If you move a class (NewClassA) from the browser and place it directly on top of a class (NewClassB) on the Nested tab, NewClassA becomes nested underneath NewClassB. However, only one level of class nesting appears on the Nested tab. You can view all levels of nesting in the browser.

Moving Nested Classes between Class Specifications

Nested classes can be dragged and dropped between Class Specification Nested tabs.

Class Specification - Components tab

Figure 51 Class Specification - Components tab



[✓] Show all components

Component Name	Package Name	Environment
AIX4Tppccset314	Component View	C++ TargetRTS
AIX4Sppccset314	Component View	C++ TargetRTS
AIX4Sppcgnu281	Component View	C++ TargetRTS
AIX4Tppcgnu281	Component View	C++ TargetRTS
CLASSIX31Tchppc	Component View	C++ TargetRTS
REDRING412TchE	Component View	C++ TargetRTS
HPUX10ThppaHP(	Component View	C++ TargetRTS
HPUX10Thppagnu	Component View	C++ TargetRTS
IRIX6Tr4400ProDe	Component View	C++ TargetRTS
IRIX6Tr4400gnu28	Component View	C++ TargetRTS
LYNX25Tppccygnu	Component View	C++ TargetRTS
LYNX25Tppccygnu	Component View	C++ TargetRTS
LYNX25Sppccygnu	Component View	C++ TargetRTS
NT40Tx86VC50	Component View	C++ TargetRTS

Components List

The components list displays a list of components to which this class has been assigned. Components can be inserted, deleted, and moved up and down in the list. Each component has a corresponding Component Specification for editing the component attributes.

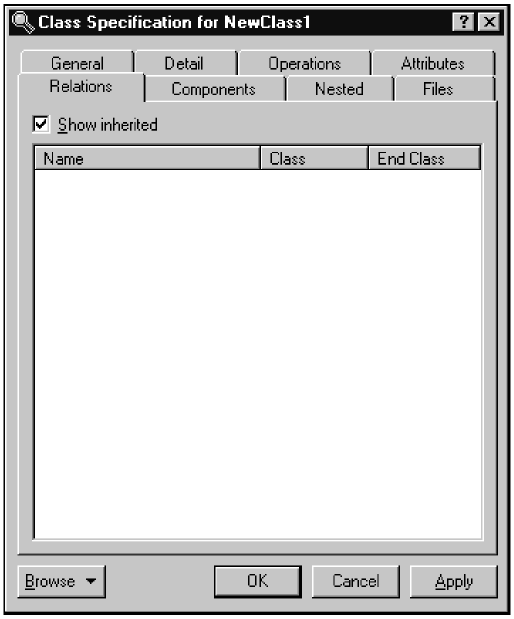
A check-box provides filtering control over which components are displayed:

Show all components displays the list of all components in the model.

Right-clicking on a component brings up the Components popup menu.

Class Specification - Relations tab

Figure 52 Class specification - Relations tab



Relations List

The relations list displays relations between the class and other model classes as specified in class diagrams. The relations list simply displays the relationships involving this class that appear on class diagrams in the model.

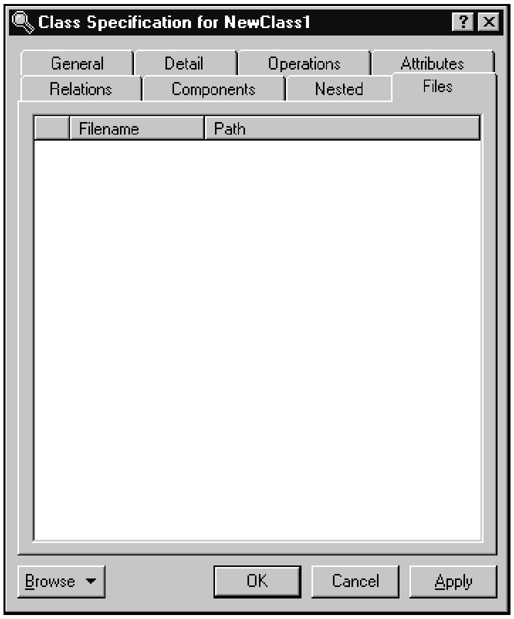
Each relation has a corresponding Association Specification for editing the relation attributes.

A check-box provides filtering control over which relations are displayed:

Show Inherited shows any relations inherited from a superclass protocol.

Class Specification - Files tab

Figure 53 Class Specification - Files tab



A list of referenced files is provided here. You can link external files to model elements for documentation purposes.

Attribute Specification Dialog

The Attribute Specification lets you display and modify the properties of a class or capsule attribute in the current model.

To display an Attribute Specification, select the entry on the Attribute tab of the Class or Capsule Specification and click **Specification** from the pop-up menu. Alternatively, double-clicking on the entry displays the Attribute Specification.

Specification Content

The Attribute Specification consists of the following tabs: General, Detail, Files.

General Tab

Name

A name for the attribute. This name will be the name for the generated attribute.

Stereotype

A stereotype value.

Class

The class the attribute belongs to is displayed in this non-editable field.

Visibility

- Public - the attribute is visible to any other classes.
- Protected - the attribute is visible only to subclasses and friend classes.
- Private - the attribute is not visible to any other classes, except designated friend classes.
- Implementation - the attribute is never visible to other classes.

Scope

- Class - there is a single instance of the attribute for all instances of the class (for example a static member in C++ terminology).
- Instance - each instance of the class will have a separate attribute instance.

Detail Tab

Type

Attribute types can either be classes or language-specific types. When the attribute is a data value, the type is defined as a language-specific type. You can enter the type in the Type field of the Class Attribute Specification. Rational Rose RealTime displays the type beside the attribute name in the class icon and updates the information in the model.

Initial Value

You can assign an initial value to your class attribute through this field. Click in the Initial Value field and enter the value.

Changeability

- Changeable - The attribute can be modified.
- Frozen - The attribute cannot be modified.
- Add-only - The attribute can only be updated in an additive way. This is not enforceable in most programming languages.

Derived

The Derived check box indicates whether the element was computed or implemented directly.

To define a element as derived, select the Derived check box. The element name is adorned by a "/" in front of the name.

If the derived box is checked, no code is generated for the attribute.

Operation Specification

You should complete one Operation Specification for each operation that is a member of a class.

If you change a class operations property by editing its specification, Rose RealTime updates all class diagrams containing icons representing that class.

To access the Operation Specification, select an entry on the Operation tab of the Class Specification and double-click the entry or click Insert from the popup menu. You can also bring the specification up through the popup menu.

Specification Content

The Operation Specification consists of the following tabs: General, Detail, Validations, Semantics, Files.

General Tab

Name

The name of the operation. The named operation will be generated as a member of the containing class.

Stereotype

Specifies a stereotype for the operation.

Class

A non-editable field that displays the class to which the operation belongs.

Visibility

- Public - Indicates that the operation is visible to other classes.
- Protected - Indicates that the operation is not part of the public interface of the class, but is visible to subclasses.
- Private - Indicates that the operation is not visible to other classes, including subclasses. May be visible to specific classes designated as friend classes.
- Implementation - Indicates the operation is not visible to any other classes, including subclasses and friends.

Options

- Polymorphic - Indicates that the operation should be inherited by all subclasses.
- Query - Indicates that the operation is read-only and does not modify the object's state.
- Abstract - Indicates that the operation is an abstract definition that should be overridden by specific implementations in subclasses.

Scope

- Instance - Indicates that the operation operates on individual class instances, usually because its calculations are based on the object state, or because it modifies the object state.
- Class - Indicates that the operation operates the same way regardless of the state of any individual object in the class.

Detail tab

Return Type

For operations that are functions, set this field to identify the class or type of the function's result. If show classes is set, the list box displays all the classes in the package. If Show Classes is not set, only the predefined set of return class types is displayed.

If you enter a class name and it does not exist in your model, the application does not create one.

Parameters

This field contains a list of the arguments of the operation. You may express these arguments in your selected implementation language.

The argument list can be rearranged with the click and drag technique. Select an argument from the list, drag it to the location, and release. The list reflects the new order.

Argument Specification Dialog:

To open the Argument Specification dialog, double click a parameter. The dialog has two tabs: General and Files.

The General tab contains fields for Name, Type, Default, and Documentation. It also provides the name of the owner of the parameter, that is, the operation that the parameter belongs to.

The Files tab provides a list of referenced files. You can link external files to model elements for documentation purposes.

You can **Undo** and **Redo** any changes from the **Edit** menu.

Moving and Copying Parameters:

To move a parameter from one specification sheet to another, drag and drop it. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

To copy a parameter from one specification sheet to another, drag and drop it while holding down the Ctrl key. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

Code

A code editor allowing you to enter the detailed implementation code for the operation.

Validation Tab

Protocol

This field lists a set of operations that a client can perform on an object and the legal orderings in which they might be invoked. The protocol of an operation has no semantic impact.

Qualifications

This field identifies language-specific features that qualify the method.

Exceptions

This field contains a list of the exceptions that can be raised by the operation. Enter the name of one or more classes identifying the exception.

Size

This field identifies the relative or absolute amount of storage consumed by the invocation of the operation.

Time

This field contains a statement about the relative or absolute time required to complete an operation. Use this field to budget time for the operation.

Concurrency

This field denotes the semantics in the presence of multiple threads of control. The Concurrency field shows the concurrency for the elements of a class. The concurrency of an operation should be consistent with its class.

Concurrency Field Options

Type	**Description**
Sequential (default)	The semantics of the operation are guaranteed only in the presence of a single thread of control. Only one thread of control can be executing in the method at any one time.
Guarded	The semantics of the operation are guaranteed in the presence of multiple threads of control. A guarded class requires collaboration among client threads to achieve mutual exclusion.
Synchronous	The semantics of the operation are guaranteed in the presence of multiple threads of control; mutual exclusion is supplied by the class.

You can set the concurrency of a class only through the Class Specification. The Concurrency field is inactive for class utilities, parameterized class utilities, and instantiated class utilities.

To change the concurrency, click on an applicable option in the Concurrency field. You can display the concurrency in the class diagram by clicking **Show Concurrency** from the shortcut menu.

Semantics Tab

Preconditions

Invariants that are assumed by the operation (the entry behavior of an operation) are listed.

Semantics

The action of the operation is shown in this area.

Postcondition

Invariants that are satisfied by the operation (the exit behavior of an operation) are listed in this area.

Interaction Diagram

Select an interaction diagram from the list box that illustrates the appropriate semantics. Selecting <New> brings up the New Interaction Diagram dialog, in which you can specify the diagram type and title.

Creating a Capsule Class

Capsule classes are created in the Logical View of the Model browser.

To create a new capsule class:

1 Right-click on the Logical View package (or another package of your choice) in the model browser.

2 Select the **New > Capsule** menu option. A new capsule class is created with a default name of 'NewCapsule1'.

3 Type over the name to change it.

You can also create new capsule classes using the capsule tool in the class diagram.

Each capsule has an associated structure diagram and state diagram.

The capsule class attributes, operations and other properties can be modified through the Capsule Specification. Open the specification dialog by double-clicking on the capsule in the model browser.

Capsule Diagrams

There are two diagrams associated with capsules:

State Diagram

The state diagram captures the high-level behavior of the capsule.

Structure Diagram

The structure diagram captures the interface and internal structure of the capsule in terms of its contained capsules and ports.

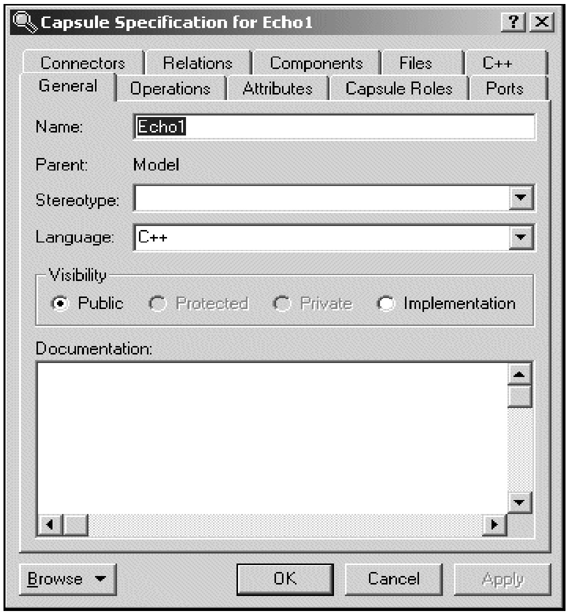
Undocking the Capsule Diagrams

These two diagrams can be docked together or viewed separately. To separate the diagrams into separate windows, grab one of the diagram tabs at the bottom of the window and drag it away to create a new window.

Capsule Specification

The capsule specification is used to edit the properties of a capsule.

Figure 54 Example specification dialog for a capsule



Name: Echo1

Parent: Model

Stereotype:

Language: C++

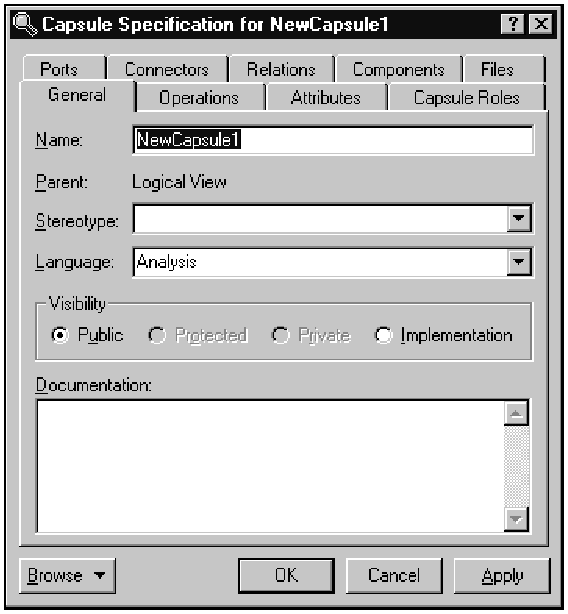
Visibility: [✓ Public] Protected Private Implementation

Documentation:

The capsule specification dialog contains the following tabs:

- Capsule Specification - General tab
- Capsule Specification - Operations tab
- Capsule Specification - Attributes tab
- Capsule Specification - Capsule Roles tab
- Capsule Specification - Ports tab
- Capsule Specification - Connectors tab
- Capsule Specification - Relations tab
- Capsule Specification - Components tab
- Capsule Specification - Files tab

Capsule Specification - General tab



Name: NewCapsule1

Parent: Logical View

Stereotype:

Language: Analysis

Visibility: [✓ Public] Protected Private Implementation

Figure 55 Capsule Specification - General tab

Name

The name of the capsule class. The capsule class name may be referenced in the detailed code of other capsule classes. One reason for this is for a container capsule to instantiate an optional capsule role (see the Frame service incarnate function).

Stereotype

A stereotype represents the subclassification of an element. It represents a class within the UML metamodel itself, that is, a type of modeling element. Some stereotypes are already predefined, but you can also define your own to add new kinds of modeling types.

Stereotypes can be shown in the browser and on diagrams. The name of the stereotype may appear in angle brackets <<>>, depending on the settings found in either the Diagram or Browser tabs of the Options dialog located under the **Tools** menu. Refer to the Stereotype chapter for more information on stereotypes.

To show stereotypes on the diagrams, click **Options** from the shortcut menu and click Stereotype Name or Stereotype Icon. Stereotype Name displays the name in angle brackets (that is, <<stereotype>>). Stereotype Icon displays the graphical representation.

Language

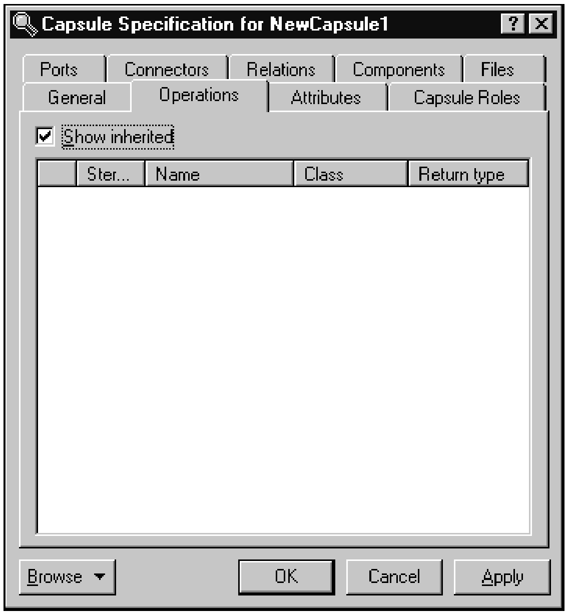
The language to be used for detailed coding and code generation.

Documentation

Use this field to enter documentation on this element.

Capsule Specification - Operations tab

Figure 56 Capsule Specification - Operations tab



Operations denote services provided by the class. Operations are methods for accessing and modifying Class fields or methods that implement characteristic behaviors of a class.

The Operations tab lists the operations that are members of this class. The actual definition of the operation is accessible from the Operation Specification.

The operations are listed with the following fields:

- Visibility Adornment (Unlabeled); the visibility of the operation is indicated with an icon. These are the visibility options:
  - Public - the operation is accessible to all clients.
  - Protected - the operation is accessible only to subclasses, friends, or to the class itself.
  - Private - the operation is accessible only to the class itself or to its friends.
  - Implementation - the operation is accessible only by the implementation of the package containing the class.
- Stereotype - displays the name of the stereotype.
- Signature - displays the name of the operation.
- Class - identifies which class defines the operation.
- Return Type - identifies the type of value returned from the operation.

The Operation tab is active for all class types. In the class diagram, you can display operation names in the class compartment.

Show Inherited

Click this option to see operations inherited from other classes. If there is no check mark in this field, you can view only operations associated with the selected class.

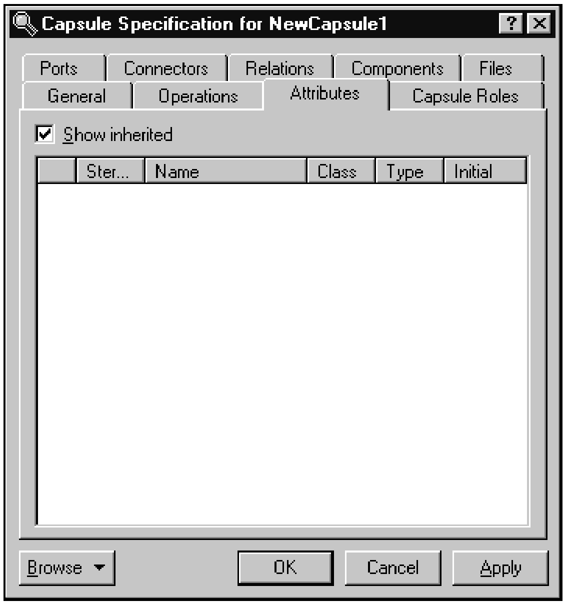
Note: Rose RealTime allows you to directly modify any operation shown in the operations list by displaying the operations specification dialog. You should be careful when modifying base class operations for it may have implications on other elements in your model which reference or are subclassed from the base class.

Creating New Operations

To enter an operation in the Class Specification, select **Insert** from the popup menu. A new operation with a default name is added to the operations list.

Capsule Specification - Attributes tab

Figure 57 Capsule Specification - Attributes tab



The UML asserts that attributes are data values (string or integer) held by objects in a class. Thus, the Attributes tab lists attributes defined for the class. The attribute definition can be modified through the Attribute Specification Dialog.

Note: Attributes and relationships created using this technique are added to the model, but do not automatically appear in any diagrams. That is, adding an attribute affects the code generation for the class and a compilation dependency between the class of the container and the class of the attribute, but these relationships are not graphically visible in the model.

The descriptions for each field follow:

- Visibility Adornment (Unlabeled):
  - Public - The attribute is publicly visible, and is accessible to all clients.
  - Protected - The attribute may be accessed only by subclasses, friends, or by operations of this class.
  - Private - The attribute is accessible only by the class itself or by its friends.
  - Implementation - the attribute is accessible only by operations in this class.
- Stereotype - displays the name of the stereotype.

- Name - displays the name of the attribute.
- Class - identifies where the attribute is defined.
- Type - this can be a class or a traditional type, such as int.
- Initial - displays the initial value of an object.

The Attribute tab is active for all class types.

Show Inherited

Click this option to see attributes inherited from other capsules. If there is no check mark in this field, you can view only attributes associated with the selected capsule.

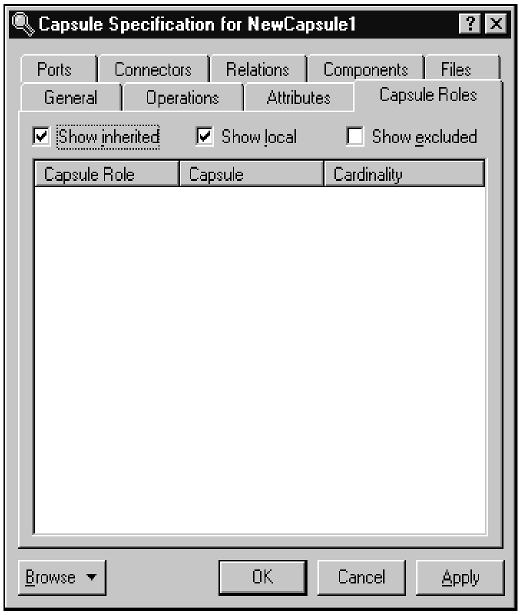
Note: Rose RealTime allows you to directly modify any attribute shown in the attributes list by displaying the attribute specification dialog. You should be careful when modifying base class attributes for it may have implications on other elements in your model which reference or are subclassed from the base class.

Creating new attributes

You can add an attribute relationship by selecting **Insert** on the popup menu or by pressing the insert key. A new attribute with a default name is added.

Capsule Specification - Capsule Roles tab

Figure 58 Capsule Specification - Capsule Roles tab



The capsule roles list displays all contained capsule roles within the immediate capsule decomposition. Capsule roles can be inserted, deleted, and moved up and down in the list. Each capsule role has a corresponding Capsule Role Specification for editing the capsule role attributes.

Inserting Capsule Roles through the Capsule Roles list is the same as adding capsule roles through the Structure diagram editor. When inserting a new capsule role, a pick-list appears allowing you to select the class for the capsule role. The new capsule role is given a default name, which can changed by double-clicking on it.

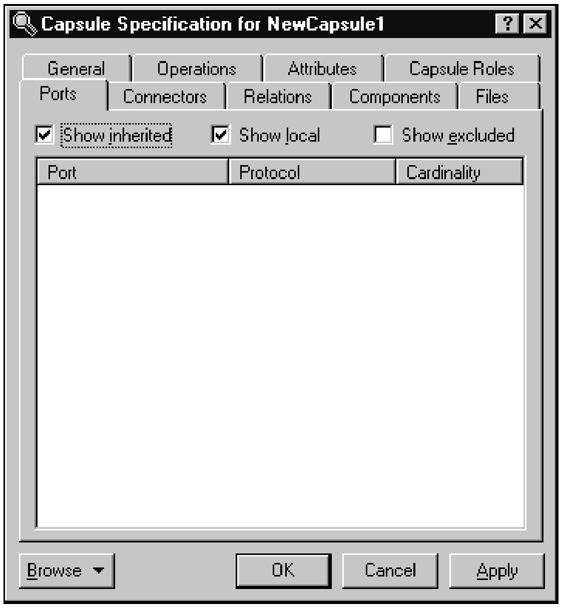
Three check-boxes provide filtering control over which capsule roles are displayed:

- **Inherited Values** - shows any elements inherited from a superclass protocol.
- **Local Values** - shows any elements defined within this capsule (not inherited).
- **Excluded Values** - shows elements defined in the superclass and deleted from the subclass.

Right-clicking on a capsule role brings up the Capsule Role popup menu.

Capsule Specification - Ports tab

Figure 59 Capsule Specification - Ports tab



Chosen settings: ☑ Show inherited, ☑ Show local, ☐ Show excluded

The ports list displays all contained ports within the immediate capsule decomposition. Ports can be inserted, deleted, and moved up and down in the list. Each port has a corresponding Port Specification for editing the port attributes.

Inserting ports through the list is the same as adding ports through the Structure diagram editor. When inserting a new port, a pick-list appears allowing you to select the protocol for the port. The new port is given a default name, which you can change by double-clicking on it.

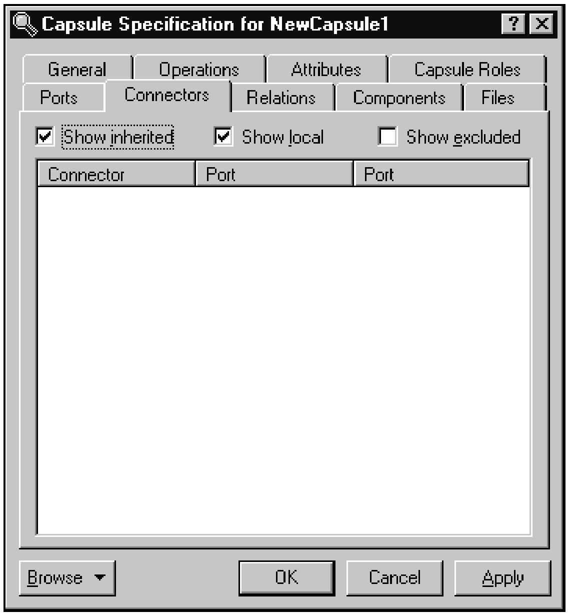
Three check-boxes provide filtering control over which ports are displayed:

- **Inherited Values** - shows any elements inherited from a superclass protocol.
- **Local Values** - shows any elements defined within this capsule (not inherited).
- **Excluded Values** - shows elements defined in the superclass and deleted from the subclass.

Right-clicking on a signal displays the Port popup menu.

Capsule Specification - Connectors tab

Figure 60 Capsule Specification - Connectors tab



Settings shown: Show inherited ☑, Show local ☑, Show excluded ☐.

The connectors list displays all connectors contained within the immediate capsule decomposition. Connectors can be deleted, and moved up and down in the list. Connectors cannot be inserted through this list: they can only be defined through the Capsule Collaboration Diagram Editor. Each connector has a corresponding Connector Specification for editing the connector attributes.

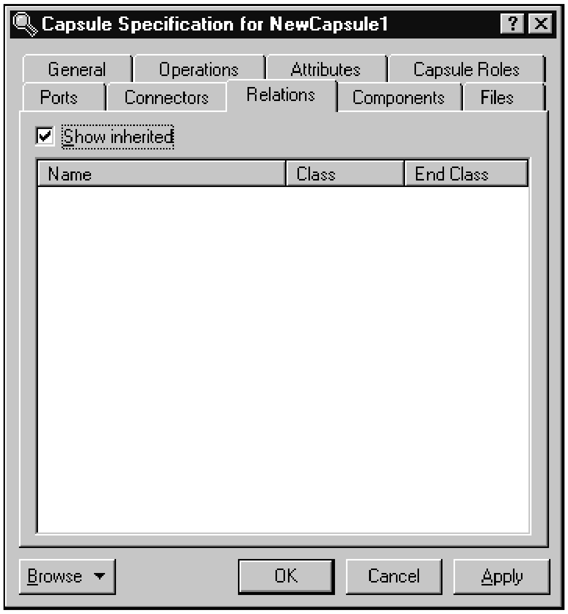
Three check-boxes provide filtering control over which connectors are displayed:

- **Inherited Values** - shows any elements inherited from a superclass protocol.
- **Local Values** - shows any elements defined within this capsule (not inherited).
- **Excluded Values** - shows elements defined in the superclass and deleted from the subclass.

Right-clicking on a signal brings up the Connector popup menu.

Capsule Specification - Relations tab

Figure 61 Capsule Specification - Relations tab



Relations List

The relations list displays relations between the class and other model classes as specified in class diagrams. The relations list displays the relationships involving this class that appear on class diagrams in the model.

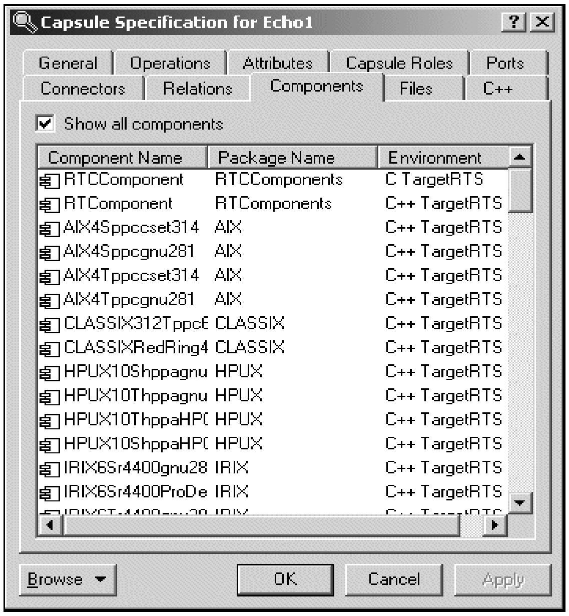
Each relation has a corresponding Association Specification for editing the relation attributes.

A check-box provides filtering control over which relations are displayed:

Show Inherited shows any relations inherited from a superclass protocol.

Capsule Specification - Components tab

Figure 62 Capsule Specification - Components tab



Capsule Specification for Echo1

[✓] Show all components

Component Name	Package Name	Environment
RTCComponent	RTCComponents	C TargetRTS
RTComponent	RTComponents	C++ TargetRTS
AIX4Sppccset314	AIX	C++ TargetRTS
AIX4Sppcgnu281	AIX	C++ TargetRTS
AIX4Tppccset314	AIX	C++ TargetRTS
AIX4Tppcgnu281	AIX	C++ TargetRTS
CLASSIX312TppcE	CLASSIX	C++ TargetRTS
CLASSIXRedRing4	CLASSIX	C++ TargetRTS
HPUX10Shppagnu	HPUX	C++ TargetRTS
HPUX10Thppagnu	HPUX	C++ TargetRTS
HPUX10ThppaHP(	HPUX	C++ TargetRTS
HPUX10ShppaHP(	HPUX	C++ TargetRTS
IRIX6Sr4400gnu28	IRIX	C++ TargetRTS
IRIX6Sr4400ProDe	IRIX	C++ TargetRTS

Components List

The components list displays a list of components to which this class has been assigned (a red check mark on the icon). Components can be inserted, deleted, and moved up and down in the list. Each component has a corresponding Component Specification for editing the component attributes.

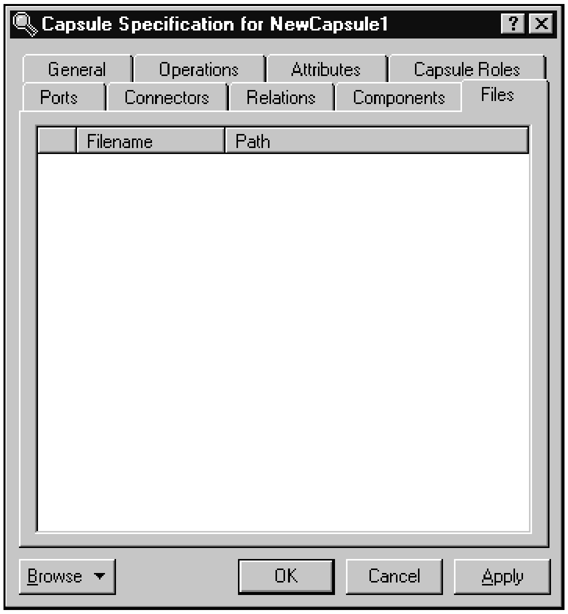
A check-box provides filtering control over which components are displayed:

Show all components displays all the components in the model.

Right-clicking on a component brings up the Components popup menu.

Capsule Specification - Files tab

Figure 63 Capsule Specification - Files tab



A list of referenced files is provided here. The files list popup menu allows you to insert and delete references to files or URLs.

You can link external files to model elements for documentation purposes.

Defining Protocols 14

Contents

This chapter is organized as follows:

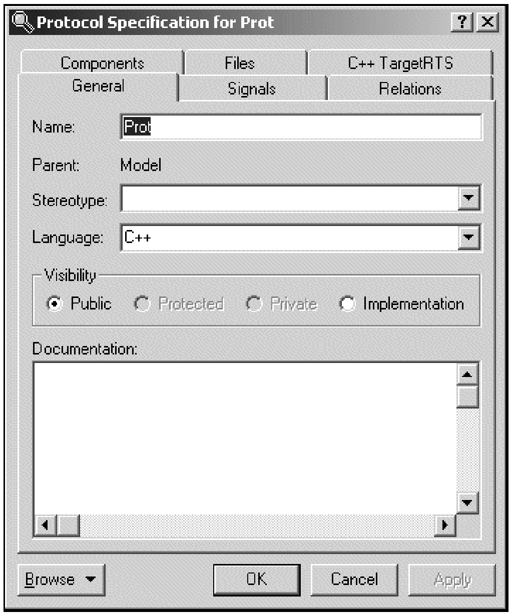
Protocol Specification

The protocol specification provides control over the definition of a protocol class. The dialog includes the following tabs:

- Protocol Specification - General tab
- Protocol Specification - Signals tab
- Protocol Specification - Relations tab
- Protocol Specification - Components tab
- Protocol Specification - Files tab

Protocol Specification - General tab

Figure 64 Protocol Specification - General tab


Protocol Specification for Prot

Name: Prot

Parent: Model

Stereotype:

Language: C++

Visibility: [✓ Public] Protected Private Implementation

Documentation:

Name

The name of the Protocol Class.

Language

Select the implementation language for the class from the available languages. The analysis selection indicates that no code will be generated for the class.

Stereotype

A stereotype represents the subclassification of an element. It represents a class within the UML metamodel itself, that is, a type of modeling element. Some stereotypes are already predefined, but you can also define your own to add new kinds of modeling types.

Stereotypes can be shown in the browser and on diagrams. The name of the stereotype may appear in angle brackets <<>>, depending on the settings found in either the Diagram or Browser tabs of the Options dialog located under the **Tools** menu. Refer to the Stereotype chapter for more information on stereotypes.

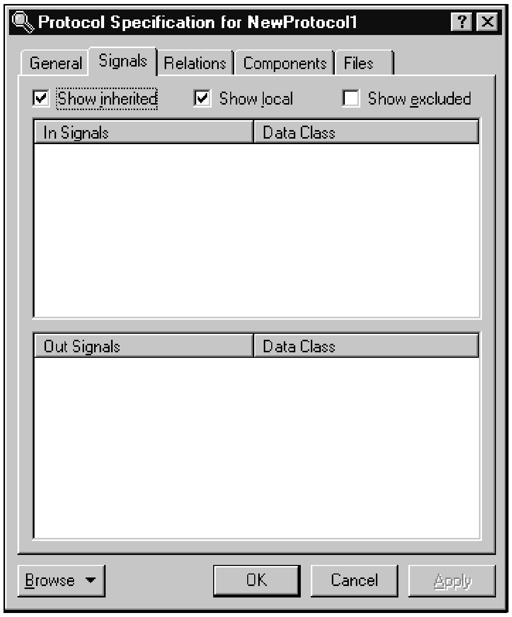
To show stereotypes on the diagrams, click **Options** from the shortcut menu and click Stereotype Name or Stereotype Icon. Stereotype Name displays the name in angle brackets (that is, <<stereotype>>). Stereotype Icon displays the graphical representation.

Documentation

Use this field to enter documentation on this element.

Protocol Specification - Signals tab

Figure 65 Protocol Specification - Signals tab



Show inherited: ✓ Show local: ✓ Show excluded: ☐

In Signals	Data Class

Out Signals	Data Class

This tab provides a list of signals that can be received (the **In** list) and sent (the **Out** list) by ports using this protocol.

In/Out Signal List

The signal list allows signals to be inserted, deleted, and moved up and down in the list. Each signal has a corresponding Signal Specification for editing the signal attributes.

Three check-boxes provide filtering control over which signals are displayed:

- Show inherited - shows any signals inherited from a superclass protocol.
- Show local - shows signals defined within this protocol (not inherited).
- Show excluded - shows signals defined in the superclass protocol and deleted from the subclass protocol.

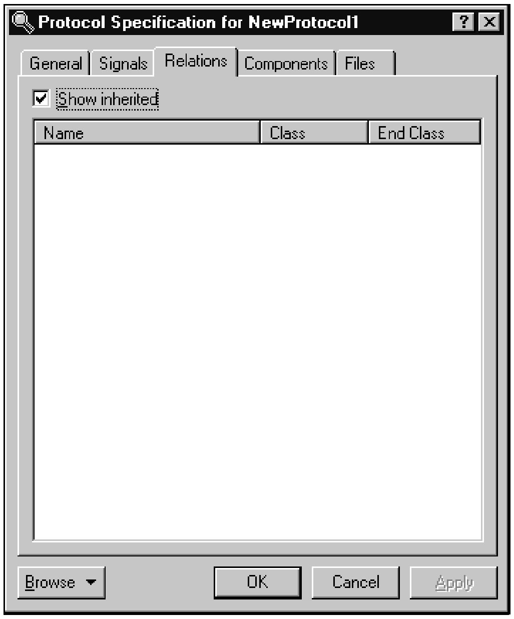
Right-clicking on a signal brings up the Signal popup menu, allowing you to insert new signals, delete signals, and promote/demote signals in the protocol class hierarchy. As well, you can select **Open Data Class Specification,** which brings up the Class Specification.

Copying signals

To copy a signal from one Specification sheet to another, drag and drop it. From the **Edit** menu of the main window, you can select **Undo** and **Redo**.

Protocol Specification - Relations tab

Figure 66 Protocol Specification - Relations tab



Relations List

The relations list displays relations between the protocol class and other model classes as specified in class diagrams. Relations can be inserted, deleted, and moved up and down in the list. Each relation has a corresponding Association Specification for editing the relation attributes.

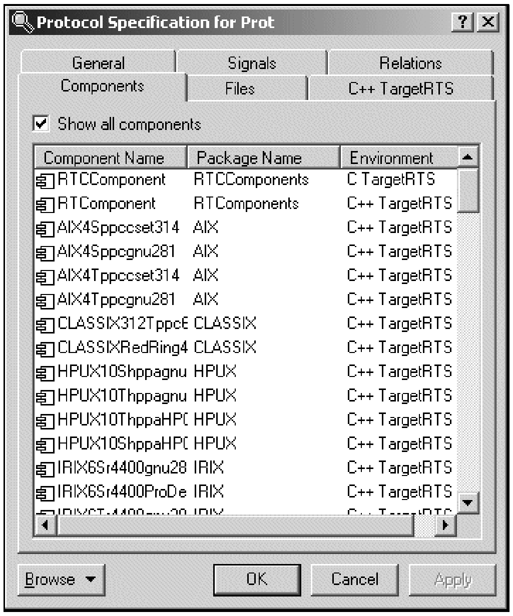
A check-box provides filtering control over which relations are displayed:

Show Inherited shows any relations inherited from a superclass protocol.

Right-clicking on a relation brings up the Relation popup menu.

Protocol Specification - Components tab

Figure 67 Protocol Specification - Components tab



☑ Show all components

Component Name	Package Name	Environment
RTCComponent	RTCComponents	C TargetRTS
RTComponent	RTComponents	C++ TargetRTS
AIX4Sppccset314	AIX	C++ TargetRTS
AIX4Sppcgnu281	AIX	C++ TargetRTS
AIX4Tppccset314	AIX	C++ TargetRTS
AIX4Tppcgnu281	AIX	C++ TargetRTS
CLASSIX312TppcE	CLASSIX	C++ TargetRTS
CLASSIXRedRing4	CLASSIX	C++ TargetRTS
HPUX10Shppagnu	HPUX	C++ TargetRTS
HPUX10Thppagnu	HPUX	C++ TargetRTS
HPUX10ThppaHPC	HPUX	C++ TargetRTS
HPUX10ShppaHPC	HPUX	C++ TargetRTS
IRIX6Sr4400gnu28	IRIX	C++ TargetRTS
IRIX6Sr4400ProDe	IRIX	C++ TargetRTS

Components List

The components list displays a list of components to which this class has been assigned (a red check mark on the icon). Components can be inserted, deleted, and moved up and down in the list. Each component has a corresponding Component Specification for editing the component attributes.

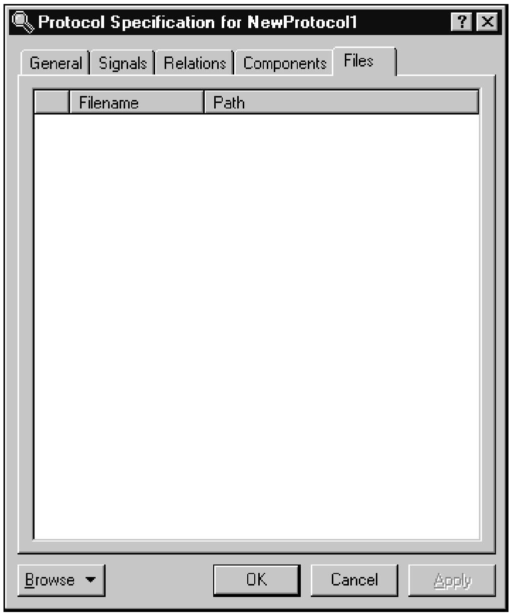
A check-box provides filtering control over which components are displayed:

Show all components displays the list of components in the model.

Right-clicking on a component brings up the Components popup menu.

Protocol Specification - Files tab

Figure 68 Protocol Specification - Files tab



A list of referenced files is provided here. You can link external files to model elements for documentation purposes.

Signal Specification

The dialog shows information about a signal in a protocol class. The signal specification is opened from the Protocol Specification - Signals tab.

There are two tabs: General and Files.

Signal Specification - General Tab

Name

Specifies a name for the signal. The name is referenced in detail code when a capsule sends a message through a port, and in the trigger event for transitions in the capsule state diagram (through the Event Editor Dialog).

Data Class

Specifies the class of the data object that is expected as a payload of the message. The data class field has a pull-down menu, which allows you to pick from the list of available data classes and types in the model.

Signal Specification - Files Tab

A list of referenced files is provided here. The files list popup menu allows you to insert and delete references to files or URLs.

You can link external files to model elements for documentation purposes.

Defining Packages 15

Contents

This chapter is organized as follows:

Introduction to Packages

Packages are organize model elements in larger models. Packages break up large models containing hundreds or thousands of elements into smaller, more manageable conceptual units. When properly designed, packages usually represent units of work for an individual or team, and units of reusability. That is, a package represents a set of highly related (highly cohesive) model elements. In most cases when looking to reuse portions of a model across software projects, entire packages would be reused rather than individual classes.

Packages also define the directory structure of a stored model. When a model is stored as controlled units, a subdirectory is created for each package, such that the representation of the model on disk mirrors the packaging hierarchy of the model in the tool.

Creating a Package

Packages can be created in the Use Case View, the Logical View or the Component View of the Model browser. Packages can contain other packages. In fact, the four main views in the model browser are themselves packages.

To create a package:

1 Right-click on the package in the model browser where you want the new package to be created.

2 Select **New >Package** from the menu.

  A new package will be created with a default name of 'NewPackage1'.

3 Type over the name to change it.

New model elements can be created within the package by clicking on the package and selecting the right-mouse button to access the popup menu.

Existing model elements can be moved across packages. See *Moving Model Elements* on page 279.

Packages and Class Diagrams

Packages can be displayed in class diagrams to show dependencies among packages. It is useful in large systems to construct top-level class diagrams that just show the packages, and allow users to drill down into the individual packages for more detailed class diagrams.

Package Specification

A Logical Package Specification enables you to display and modify the properties and relationships of a logical package in the current model.

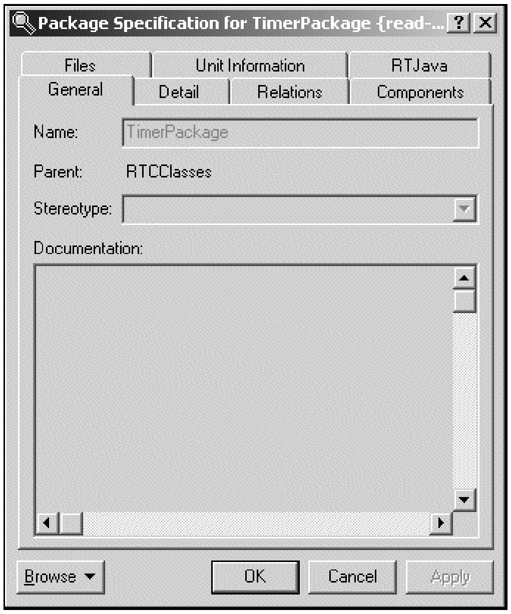
If you change a package's properties or relationships by editing its specification, the application updates all class diagrams containing icons representing that logical package. If you change a logical package's properties or relationships by editing a diagram containing its icon, the application updates the logical package's specification and any other diagrams containing its icon.

The package specification dialog provides control over the definition of a package. The dialog includes the following tabs:

- Package Specification - General tab
- Package Specification - Detail tab
- Package Specification - Relations tab
- Package Specification - Files tab

Package Specification - General tab

Figure 69 Package Specification—General tab


Package Specification for TimerPackage {read-...

Name: TimerPackage

Parent: RTCClasses

Stereotype:

Documentation:

Name

The name of the package.

Parent

Displays the name of the parent package. If this is one of the top-level view packages, the parent is the Model.

Stereotype

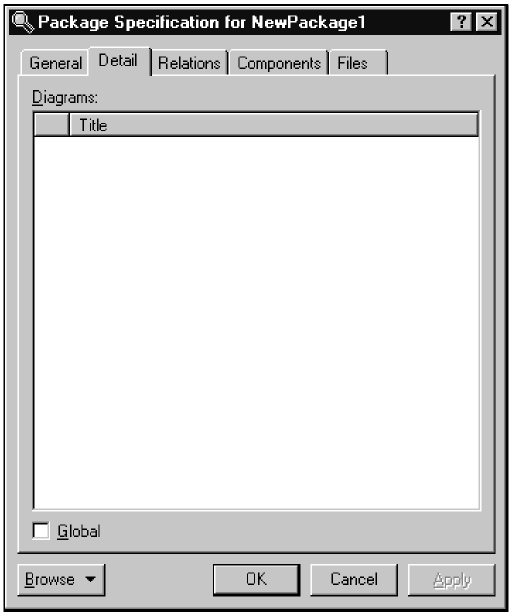
Displays the stereotype of the package. There are no pre-defined package stereotypes.

Documentation

Use this field to enter documentation on this element.

Package Specification - Detail tab

Figure 70 Package Specification - Detail tab



Global

The Global check-box indicates that all public classes in the logical package can be used by any other logical package.

To switch the global adornment, click on the Global check-box. When you set the global indicator, Rational Rose displays the word "global" in the lower left corner of the logical package icon.

You can change the global adornment only through the specification.

Diagrams

This field lists the diagrams contained in the package. When you add a diagram to the package, Rational Rose automatically updates this list.

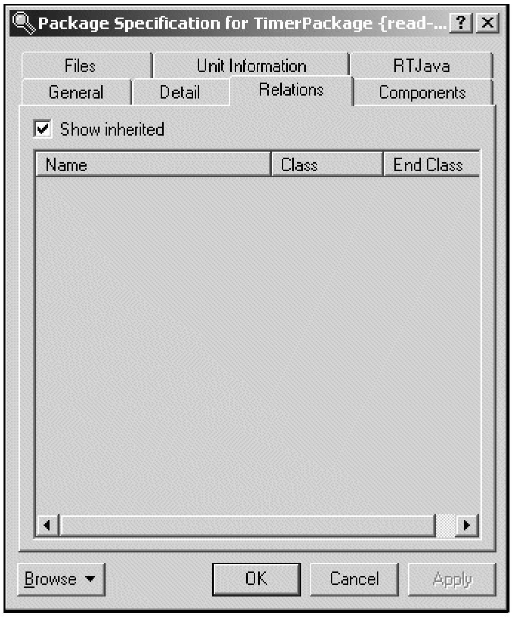
The first column contains the diagram's icon. The Title field is the title of the diagram you entered (and can be modified).

To add a new diagram, use the shortcut menu and select the appropriate insert diagram option.

The software displays the newly-created diagram.

Package Specification - Relations tab

Figure 71 Package Specification - Relations tab



Relations List

The relations list displays relations between the package and other model classes and packages as specified in class diagrams. Relations can be inserted, deleted, and moved up and down in the list. Each relation has a corresponding Association Specification for editing the relation attributes.

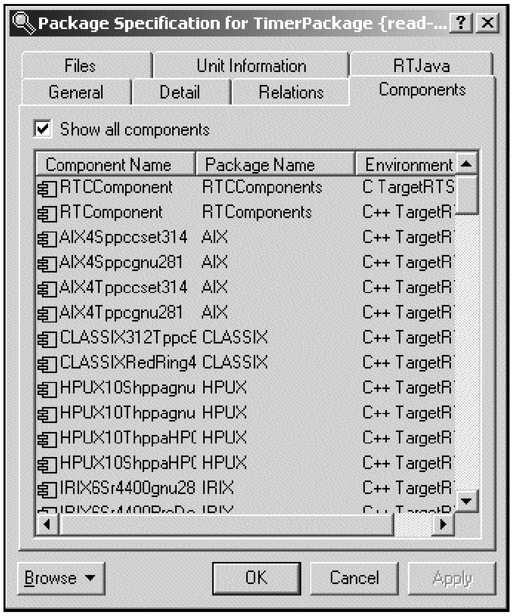
A check-box provides filtering control over which relations are displayed:

Show Inherited shows any elements inherited from a superpackage.

Right-clicking on a relation brings up the Relation popup menu.

Package Specification - Components tab

Figure 72 Package Specification - Components tab



Component Name	Package Name	Environment
RTCComponent	RTCComponents	C TargetRTS
RTComponent	RTComponents	C++ TargetR
AIX4Sppccset314	AIX	C++ TargetR
AIX4Sppcgnu281	AIX	C++ TargetR
AIX4Tppccset314	AIX	C++ TargetR
AIX4Tppcgnu281	AIX	C++ TargetR
CLASSIX312TppcE	CLASSIX	C++ TargetR
CLASSIXRedRing4	CLASSIX	C++ TargetR
HPUX10Shppagnu	HPUX	C++ TargetR
HPUX10Thppagnu	HPUX	C++ TargetR
HPUX10ThppaHPC	HPUX	C++ TargetR
HPUX10ShppaHPC	HPUX	C++ TargetR
IRIX6Sr4400gnu28	IRIX	C++ TargetR

Components List

The components list displays a list of components that reference this package (red checkmark). Components can be inserted, deleted, and moved up and down in the list. Each component has a corresponding Component Specification for editing the component attributes.

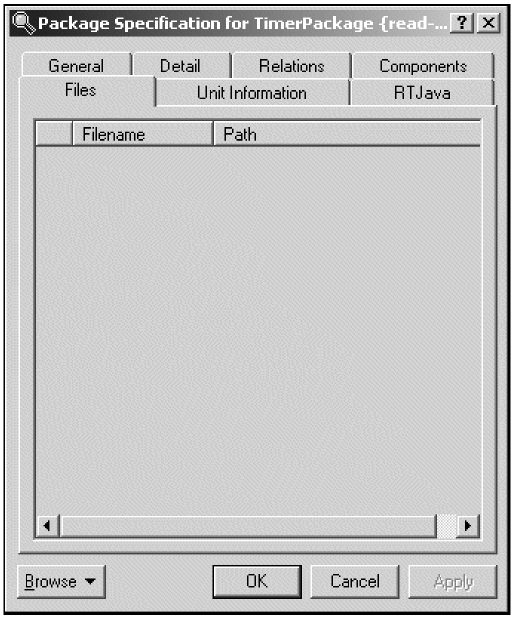
A check-box provides filtering control over which components are displayed:

Show all components displays the list of all components in the model.

Right-clicking on a component brings up the Components popup menu.

Package Specification - Files tab

Figure 73 Package Specification - Files tab



A list of referenced files is provided here. The files list popup menu allows you to Insert and Delete references to files or URLs.

You can link external files to model elements for documentation purposes.

Moving Model Elements

Classes and diagrams can be moved from one package into another package in the Model browser.

To move a class:

1. Click on the class in the model browser.
2. Drag it over the destination package.

To move a diagram:

1. Click on the diagram in the model browser.
2. Drag it over the destination package.

Impact of Moving Classes or Diagrams on Configuration Management

Because classes are stored in the CM system under the package directory, moving a class to another package causes a mismatch between the stored directory structure and the in-tool model packaging. The classes in the stored model directory are not automatically moved to new directories. The mismatch does not cause any problems for the tool (Rose RealTime keeps track of what the stored file name is for the element - see Unit Information tab), but it may cause confusion for users working directly with the CM tool.

Creating the Component and Deployment Views 16

Contents

This chapter is organized as follows:

Using the Component Diagram Editor

The component diagram editor is used to create a diagram showing the software as releasable units, together with their interfaces and inter-dependencies. Multiple Component diagrams can exist in the same model.

A component diagram shows the physical dependency relationships (mapping to a file system) between components - main programs, subprograms, packages, and tasks - and the arrangement of components into component packages.

Component diagrams are contained (owned) either at the top level of the model or by a package, which means that the diagram depicts the components and packages where the diagram is contained.

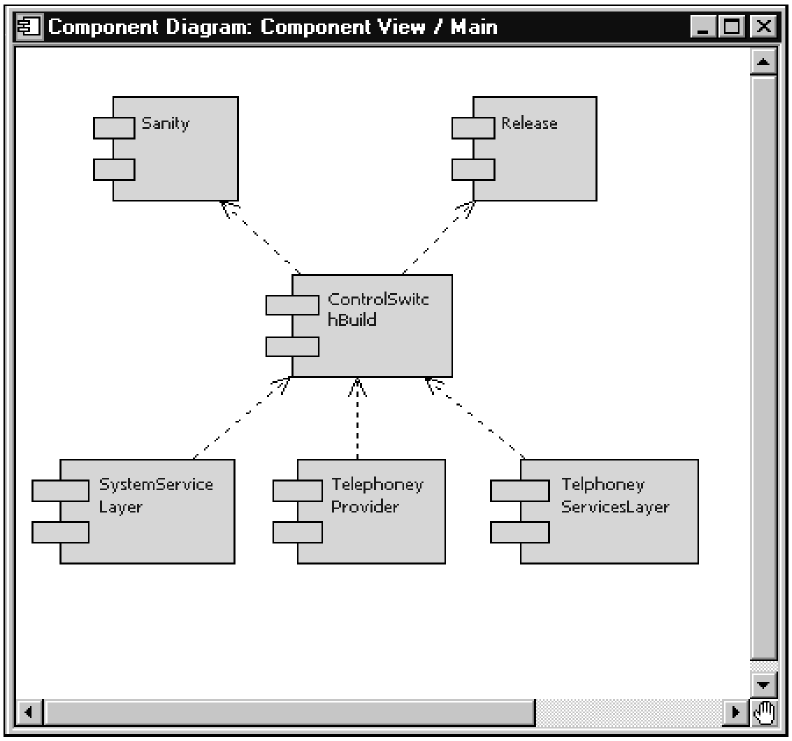
The component diagram consists of two parts:

- the diagram area
- the Component Diagram Toolbox

Elements of the component diagram, such as packages and components, are added using the toolbox or by dragging them from the browser. You can undo and redo moves from the **Edit** menu.

The window title bar shows the full name of the component diagram.

Figure 74 Component diagram



Component

Components can be added to the diagram using either the component tool from the toolbox, or by selecting a component from the Model browser and dragging and dropping it on to the diagram. Components may have dependency or aggregation relationships with other components.

The component details are specified through the Component Specification.

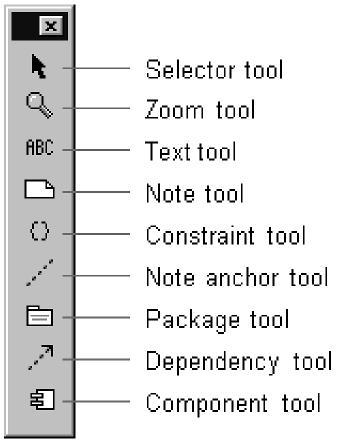
Dependency

A dependency indicates a client and supplier relationship. The client depends on the supplier to provide certain services. Use this relationship to indicate that the operations of the client invoke operations of the supplier.

Component Diagram Toolbox

The component toolbox contains tools for adding elements to the component diagram.

Figure 75 Component diagram toolbox



Selector tool

Use to select objects for moving, resizing, and so forth.

Zoom tool

Use to zoom in on a portion of the diagram. Click on the tool and then click on the part of the diagram you want to zoom in on.

Text tool

Use to add text anywhere in the structure diagram.

Note tool

Use to annotate the diagram with textual notes. This is useful for marking up the diagram with explanations, review comments, and so forth. You can drag and drop a diagram or external document from the browser onto a note. Notice that the name of the diagram or external document is underlined. If you double-click on the note, the diagram or external document is opened. You can undo and redo this command.

Constraint tool

Use to add UML constraints to the diagram. A constraint can be anchored to a view element by using the anchor tool. Currently, constraints do not have any semantic meaning to the tool. There are RRTEI APIs to add or remove, and enumerate constraints in a diagram.

Note anchor tool

Use to anchor a note to a particular element on the diagram.

Package tool

Use the Package Tool to add a package to the diagram. The package is given a default name such as 'NewPackage1'.

Dependency tool

Use to indicate a dependency between packages or between components. A dependency indicates that some element in one package depends on (uses) some element in another package.

Component tool

Use to add a component to the diagram. The component is given a default name such as 'NewComponent1'. See “Building Basics” on page 295 for more information on creating and building components.

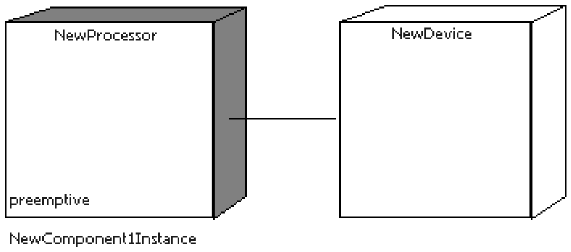
Using the Deployment Diagram Editor

The deployment diagram editor is used to create a diagram showing system deployment across processing nodes. The deployment diagram shows the allocation of processes to processors in the physical design of a system. A deployment diagram may represent all or part of the process architecture of a system. Multiple deployment diagrams can exist in the same model. The deployment diagram consists of two parts:

The diagram area, and the toolbox.

The window title bar shows the full name of the deployment diagram.

Figure 76 Deployment diagram editor



Deployment Diagram Elements

A deployment diagram shows the hardware configuration of the system under construction, and the distribution of software across that configuration.

There are four types of elements that can be placed on the diagram:

- Two types of hardware node: processors and devices
- Connections between the hardware nodes
- Software component instances deployed on the hardware nodes

Processors

A processor is a hardware component capable of executing programs. You can further define a processor by identifying its processes and specifying the type of process scheduling it uses.

The Processor Specification Dialog dialog provides details on processor attributes.

Devices

A device is a hardware component with no computing power. Each device must have a name. Device names can be generic, such as "modem" or "terminal."

The device specification dialog provides details on device attributes.

Connections

A connection represents some type of hardware coupling between two nodes. The hardware coupling can be direct, such as an RS232 cable, or indirect, such as satellite-to-ground communication. Connections are usually bi-directional.

The Connector Specification provides details on connection attributes.

Components

Components can be placed on processors for control over the distribution of the software for execution. The result is a component instance that can be specified through the Processor Specification Dialog dialog.

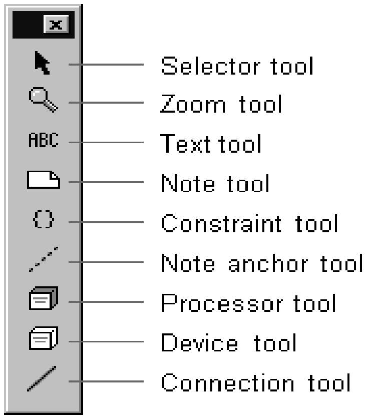
Packages

Use to add a package to the diagram. The package is given a default name such as 'NewPackage1'.

Deployment Diagram Toolbox

The deployment diagram toolbox contains tools for adding elements to the deployment diagram.

Figure 77 Deployment diagram toolbox



Selector tool

Use to select objects for moving, resizing, and so forth.

Zoom tool

Use to zoom in on a portion of the diagram. Click on the tool and then click on the part of the diagram you want to zoom in on.

Text tool

Use to add text anywhere in the structure diagram.

Note tool

Use to annotate the diagram with textual notes. This is useful for marking up the diagram with explanations, review comments, and so forth. You can drag and drop a diagram or external document from the browser onto a note. Notice that the name of the diagram or external document is underlined. If you double-click on the note, the diagram or external document is opened. You can undo and redo this command.

Constraint tool

Use to add UML constraints to the diagram. A constraint can be anchored to a view element by using the anchor tool. Currently, constraints do not have any semantic meaning to the tool. There are RRTEI APIs to add or remove, and enumerate constraints in a diagram.

Note anchor tool

Use to anchor a note to a particular element on the diagram.

Processor tool

Use to add a processor node to the diagram. Click on the diagram to place a new processor at the selected location.

Processors are given default names, such as 'processor1', when initially drawn. To change the name, click on the device and hit the backspace key to delete the default name, then type the new name.

Device tool

Use to add a device node to the diagram. Click on the diagram to place a new device at the selected location. Devices are given default names, such as 'device1', when initially drawn.

Connection tool

Use to add a connection between two nodes on the diagram. Click on the first node on the diagram and drag the connection to the second node.

Importing and Exporting 17

Contents

This chapter is organized as follows:

Importing a File

Several different kinds of files can be imported into Rose RealTime. These type of files are:

- .rtptl (RRT petal file)
- .ptl (Rose petal file)
- .cat (Rose package file)
- .sub (Rose component package file)

To import an element into a Rose RealTime model:

1. Select **File > Import**
2. In the resulting dialog, select the type of file you want to import.
3. Select the file and click **Open**.

Exporting a File

Packages, classes, components, and use cases can be exported into .rtptl files. This allows these types of files to be imported into other models.

To export an element in a model:

1 Select the element to be exported in the model browser.

2 Right-click and chose **File > Export** from the popup menu.

Note: Diagrams cannot be exported by themselves, as they belong to the package they are defined in.

Note: Do not export Services Library shared packages. For more information, see Exporting Controlled Element From Model To File in the *Guide to Team Development - Rational Rose RealTime*

Naming Guidelines 18

Contents

This chapter is organized as follows:

Introduction to Naming Guidelines

Rose RealTime does not support name spaces, so there are a number of names that will be part of the global name space. Be careful not to use the same names for any elements that may conflict. Also, make sure you avoid using reserved names, such as any names from the Rose RealTime Services Library, Language-reserved words (for example, C++), names of common operating system functions or data structures. Spaces in names should also be avoided because some targets do not handle them.

Assigning Names

Each unique model element must have a unique name, and each relationship can be labeled with a word or phrase that denotes the semantics or purpose of the relationship. You can type the name in the diagram or in the Name field in the specification.

- If you type the name in the diagram, your entry is displayed in the Name field.
- If you type the name in the specification, the software displays the new name in the element icon and updates the information in the model.

You can rename an element using one of the following methods:

- Change its name in the diagram.
- Change its name in the specification.
- Change its name in the browser.

For more information about renaming, see the topic *Renaming a Model Element*.

Special Case Notes

Special considerations for naming include the following:

- **class attribute** - each attribute must be unique within a class.
- **operation** - omit the function parenthesis when typing the operation name. The software automatically displays the parenthesis when you display the operation in the class compartment.
- **connection** - the name field is optional.
- **state** - state icons that have the same name are assumed to represent the same state if they appear in the same context; otherwise, each state icon is assumed to represent a distinct state. State icons that appear in different state diagrams represent distinct states, even if they have identical names.
- **use case actors, classes, capsules, protocols** - these elements must all have names that are unique. Class names are part of the generated code name space, and must not conflict with each other or with other items in the global name space, for example, global functions, signal names.

Building and Executing Models 19

Contents

This chapter is organized as follows:

Building and Running Models

The mapping from design - that is, classes and capsules - to source code and executables is not an easy task. It is during this phase of the software development process that the majority of errors are introduced into a system, especially when it is done manually. There is always a risk that the implementation will diverge from the original design, and in most cases that is exactly what happens. However, since the UML has well-defined semantics, Rose RealTime can automatically generate a model or design into a lower-level language and then compile it into an executable. With automatic total source code generation of your design, the model becomes the system.

Is Rose RealTime a Compiler?

The answer is yes and no. Rose RealTime compiles models into a high-level language representation. It generates source code, or complete implementations, of models while the generated source code is compiled and linked into machine language using an external compiler and linker. The result is an executable that can be run and observed via the Rose RealTime toolset.

Real-Time Services (Services Library)

Behavior in a model is specified using a State machine, and communication patterns are specified with capsule structure. When a model is built, these abstractions must be converted to implementation. Normally, you would have to implement your own state machine, inter-process communication, concurrency control, thread management, timing, and debugging capabilities. However, Rose RealTime provides a set of pre-compiled Services Libraries for different platforms, which provides this functionality for you. In summary the facilities provided by the RealTime Services Library are:

- The mechanisms that support the implementation of concurrent communicating state machines
- Thread management and concurrency control
- Timing
- Inter-thread and inter-process communication
- Observability and debugging of a running model

Before you Start

To allow models to be built and executed on a variety of platforms with different tools (for example, on Windows NT with Visual C++ 6.0 or on Solaris with gcc 2.8.1), Rose RealTime allows build settings to be fully configurable. However, even though complicated build and execution configurations can be setup, there are also default settings that can be used to build and execute less complicated models.

To learn more about building and executing models select a basic or more advanced topic from the list below:

Building

- Building Basics - helps you build your first model
- Creating a Component - using component aggregation

Executing

- Execution Basics - helps you run and observe your first model
- Loading and Running Component Instances on Embedded Targets - target loading, restarting, and resetting
- Overview of Observability Options - watches, traces, sequence diagrams, behavior breakpoints, logging output, source code break points
- Running from Outside the Toolset - running a model without immediate observation, attaching to a running model

Building Basics

Before trying to build a model, it is important to understand the role of components for modeling the physical aspects of a system. The physical elements of a model refer specifically to source code and executables.

A component is always created with a default configuration for your host machine. This includes a default compiler, compiler flags, linker, and so forth. In many cases these settings are sufficient for building simple sets of classes and capsules that do not require integration with external source files, or libraries.

This section leads you through the steps of building a simple model that does not require integration with external files (everything is defined with the toolset). This will help you understand the build workflow without getting into specialized configuration options. After you understand the basic build workflow refer to the Component Wizard for more information on configuring components with advanced build setting.

Top-level Capsule

Basically any capsule can be built and run. The capsule that you choose to build is called the top-level capsule. It represents the highest scope of the executable that you want to create. All classes and capsules referenced (contained or in a dependency relationship) with the selected top-level capsule, directly or indirectly, will also be compiled.

Note: Since any set of capsule and classes can be compiled, you are not required to compile the whole model all the time. The capsule you decide to build may form only a subset of the whole system. This allows for easier unit testing.

1 Create a component.

2 Build the component.

3 Review the build results.

Assigning an Active Component

If you find yourself building and running the same component and component instances often you should configure an active component. When a component is configured as being active the toolbar build icons and menu items become available for easy access to common build and run commands. In addition you can configure which component instances (executables) should be automatically run when the run button is pressed.

In the browser, select **Set As Active** from the popup menu

or

1 From the **Build** menu select the **Settings** item to open the Build Settings dialog.

2 From the **Active Component** combo box, choose a component that will be become the active component.

3 For information on the other configurable build options shown, see Build Settings Dialog.

4 Click **OK**.

   Notice that the build toolbar icons are now enabled and so are items under the Build Menu.

Creating a Component

In order to build an executable of a model you must first create a component that will be used to manage the build configuration parameters. There are a couple of different ways of creating a component and assigning a top-level capsule.

You can create the component first, then assign the top-level capsule to it later.

To create a component:

1 Select the Component View folder, right-click and from the popup menu choose **New > Component**.

  A new component with the default settings for your platform is created.

2 Double-click on the default Component diagram, usually called Main, to open it.

3 Drag and drop the new component you just created onto the Component diagram.

4 Then drag and drop the top-level capsule onto the new component that was added to the component diagram.

  Note: You can also assign a capsule or class to a component by dragging and dropping the capsule, class, or protocol from the model browser onto the component in the model browser.

5 Open the components specification, switch to the References tab and set the top-level capsule.

Alternatively, you can use the Component Wizard to help configure a component. To run the Component Wizard, select **Build > Component Wizard**.

Starting a Build

When a component is built, there are actually quite a number of things that happen. First the capsules referenced by the component are verified, then the model files are written to disk, an external program is called to generate the source code from the model files, the external compiler is invoked to compile, and lastly the linker is invoked to create the final executable version of the component.

Each phase of the build process produces output that is used by the next phase, with the final result being an executable.

To build a component from the browser:

1 Select the component from the model browser.

2 Right-click and select **Build** from the popup menu.

  Note: If you are working on a UNIX-based platform, and are planning to run the component with Purify, select **Build > Link with Purify**. For information on running a component with Purify, see *Running a Component Instance with Purify* on page 315.

3 After the elements have been saved to disk the build dialog appears and shows the build progress.

The build results will be shown. You should review to see if there are any errors or warnings.

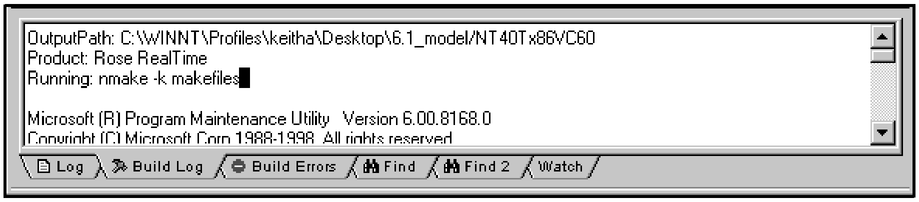
Building a Component from the Build Menu or Toolbar

Instead of directly building a component from the browser, you can build the active component directly from the **Build** menu or by selecting one of the active component toolbar buttons to verify, generate, or build the active component.

Reviewing the Build Results

You can view the results of your build by selecting **View > Output** and clicking the Build Log tab.

Figure 78 Build Log tab



```
OutputPath: C:\WINNT\Profiles\keitha\Desktop\6.1_model/NT40Tx86VC60
Product: Rose RealTime
Running: nmake -k makefiles

Microsoft (R) Program Maintenance Utility   Version 6.00.8168.0
Copyright (C) Microsoft Corp 1988-1998. All rights reserved.
```

Review any errors shown in the Build Errors tab, and correct before trying to rebuild. You can jump to the error location in your model by double-clicking on any error shown in the bottom part of the results window. As well, you should be familiar with some of the most common build errors (Understanding Build Errors). They are described briefly and should be used in conjunction with your compiler and linker documentation.

The Build Log Tab contains stdout and stderr of all phases of generation, compilation, and link. The Build Errors Tab contains a parsed version of the output stream.

Build Menu

Build

Opens the Build dialog from which you can choose the Build Level.

Quick Build

Builds the component incrementally.

Rebuild

Forces a complete build of a component. All classes references by the component will be verified, regenerated, compiled, and linked.

Clean

Removes all files from the output directory.

Code Sync

Invokes the mechanism to capture external changes made to the generated code back into the model. For more information, see “Using Code Sync to Change Generated Code” on page 363.

Stop Build

Stops the build (or the Code Sync) in progress.

Run

Loads the component instances specified in the Build Settings Dialog. The component must be successfully built before it can run.

If the Attach Target observability flag was set on the Component Instance Specification dialog, and a Target observability Port number filled in, then the execution interface is displayed allowing you to control the execution of the model.

Start (F5)

Starts the execution of the component instances. If the component instances are in the reset state, then execution begins with all fixed capsules being initialized (initial transitions fired). If the component instances are in the stop state, then execution resumes.

Stop (Shift+F5)

Stops the execution of the component instances at the current point of execution and remembers the state of all capsules. Execution is stopped as soon as each currently running transition is finished. The stop button does not halt execution in the middle of a transition action.

Step (F10)

Steps through the next deliverable message. Pressing the step button while in the stopped state causes the next message of the highest available priority to be delivered, and any associated transitions are executed. Execution stops again as soon as the last transition segment for that message has finished executing.

Restart (Ctrl+Shift+F5)

Resets the component instances, resetting all fixed and destroying all dynamic capsule instances. The running component instance is terminated and a new one is run.

Load

Loads the components instances specified in the Build Settings dialog. The component must be successfully built before it can run. The Load command spawns an external process in which the model executable runs. You will likely see an external command window appear.

The Attach Target observability flag must be set on the Component Instance Specification dialog, and a Target Observability Port number filled in for the model to be loaded within the tool.

The execution interface is displayed allowing you to control the execution of the model. See Execution basics for more information on the execution tools.

Reload

Kills the existing model process and runs the model again. The execution interface stays open.

Shutdown

Kills the existing model process and closes the execution interface.

Settings...

Displays the Build Settings Dialog. You must use this dialog to specify the active component before you can build the component.

Add Class Dependencies...

Runs a script that checks for any missing dependencies between model elements and add them. The script checks dependencies found in attributes or operations. It does not check for code-level dependencies.

Component Wizard...

Activates the Component Wizard to help you through the steps of creating and deploying a component.

Build Settings Dialog

The Build Settings dialog is used to select an active component for building and component instances for running. The build settings are not saved as part of a model. They are saved with the workspace.

Active Component

Used to select an active component. The combo box contains all components in your model.

Active Component Instances List

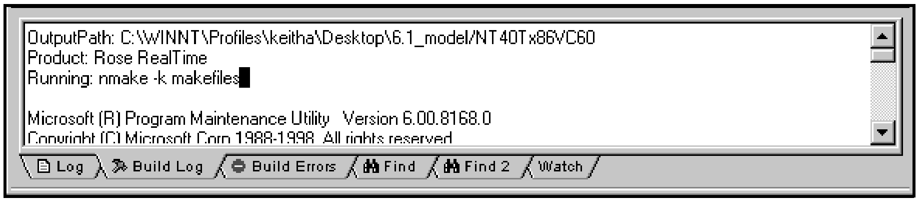
This list is populated with all the component instances in the model. Component instances that are selected in this list are automatically run when the active component is run. You can select and de-select component instances by clicking in the checkbox on the left-hand side of each component instance name. The order in which the component instances are run is determined by the load order setting in the Component instance specification.

Build Log Tab

The Build Log tab stores the contents of the compilation and code generation log. Select **View > Output** and click the Build Log tab to open it. Compilation or code generation messages are posted to the Build Log tab regardless of whether it is visible.

You can save the contents of the Build log tab to a file. You can also choose to automatically save messages to a file as they are posted.

Figure 79 Build Log tab


```
OutputPath: C:\WINNT\Profiles\keitha\Desktop\6.1_model/NT40Tx86VC60
Product: Rose RealTime
Running: nmake -k makefiles

Microsoft (R) Program Maintenance Utility   Version 6.00.8168.0
Copyright (C) Microsoft Corp 1988-1998. All rights reserved.
```

The Build Log tab contains the raw output stream from the build. You can examine the contents of this window to get a context on any error message displayed in the build messages list.

Build Errors Tab

The Build Errors tab contains a parsed version of the output stream. It is important to review the Build Log tab because some errors cannot be parsed by the error parser.

The Build Errors tab contains a Location column that gives the class/code segment name pair. The Context column provides the context of the problem. The Message column gives a description of the problem. These messages are taken directly from the compiler error stream and therefore reflect the accuracy of the compiler that you are using. Further, errors within your code segments may lead to errors being reported in system-generated files.

Double clicking on an error or warning in the Build Error tab brings you to the location in the model of the problem that caused the error or warning. See Common Build Errors (*Understanding Build Errors* on page 307) for a short summary of common generic build errors.

Unknown Compiler Message Stream

It is possible that the compiler being used reports errors in ways that are not understood by Rose RealTime. There are no standards for error reporting by compilers and linkers. Hence, the error parser is often targeted for a particular compiler and linker. If you are using an unsupported compiler, Rose RealTime will probably not be able to understand the error output from the parser, and may inaccurately report errors. You have to rely on the raw output stream to see the direct output of the compiler, rather than going by the errors reported by the Build Errors tab.

See the *Porting Guide*.

Component Specification

A Component Specification displays and modifies the properties and relationships of each component in the current model. The same specification is used for all component kinds.

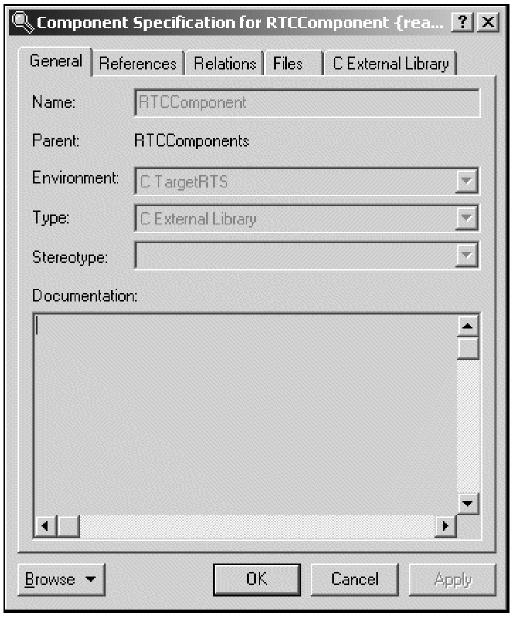
Specification Content

The Component Specification consists of the following tabs:

- Component Specification - General tab
- Component Specification - References Tab
- Component Specification - Relations Tab
- Component Specification - Files tab

Component Specification - General tab

Figure 80 Component Specification - General tab


Component Specification for RTCComponent {rea...

Name: RTCComponent

Parent: RTCComponents

Environment: C TargetRTS

Type: C External Library

Stereotype:

Documentation:

Name

The component name is referenced during the build process.

Parent

Specifies the parent component package.

Environment

Specifies the run-time system and code generator used in the build.

Type

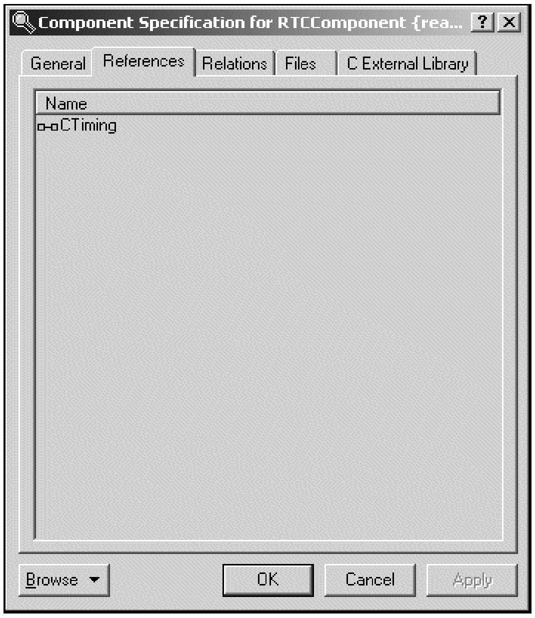
Specifies what is being built, for example, an executable or a library.

Stereotype

A component stereotype represents the subclassification of an element. The most common type of components are already predefined as stereotypes, including Main Program, Package Body, Package Specification, Subprogram Body, Subprogram Specification, Task Body and Task Specification. You can also define and add your own kinds of stereotypes.

Component Specification - References Tab

Figure 81 Component Specification - References tab


Component Specification for RTCComponent {rea...

Name
CTiming

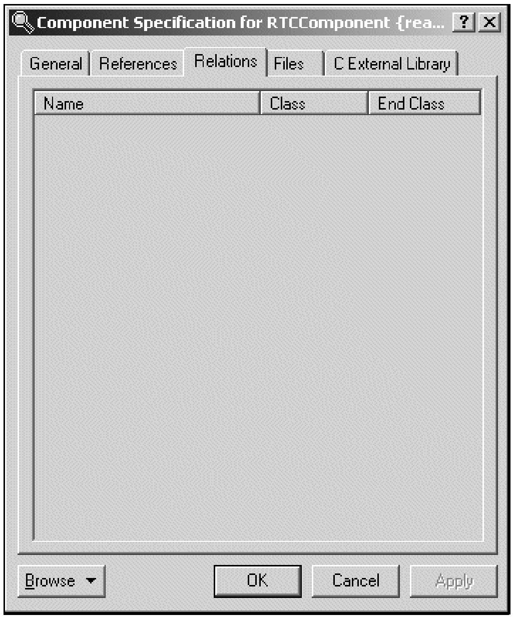
References List

The references list displays the list of packages (includes all elements in the package), classes, capsules and protocols to be compiled with this component.

If during the dependency check elements that are not in this list are found to be needed for the build, a dialog appears asking you to add them.

Component Specification - Relations Tab

Figure 82 Component Specification - Relations tab

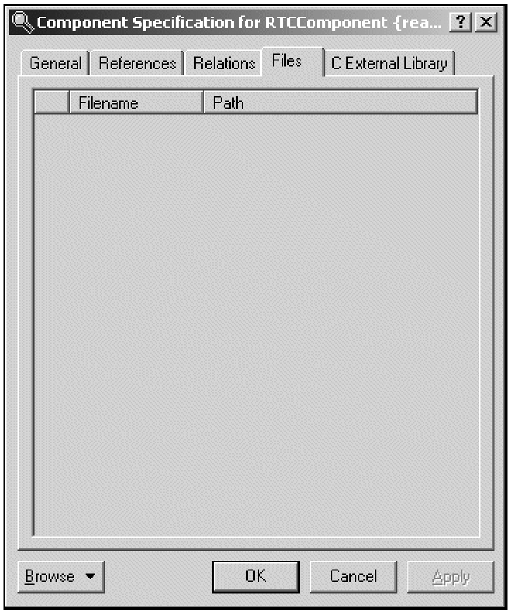


Relations List

The relations list displays aggregation relations between the component and other components in component diagrams.

Component Specification - Files tab

Figure 83 Component Specification - Files tab



A list of referenced files is provided here. The files list pop-up menu allows you to insert and delete references to files or URLs.

You can link external files to model elements for documentation purposes.

Component Dependencies

You can break up the system you are building into multiple components. Model the build dependencies using the component dependencies.

See the *Guide to Team Development - Rational Rose RealTime* and language-specific guides for more information.

Common Build Errors 20

Contents

This chapter is organized as follows:

Understanding Build Errors

Often, the compilation details returned in the Build Results window gives you a clearer picture of what the error is, but you must understand what the compilation details are reporting. The compilation details are the direct results returned by the compiler. They contain the names and line numbers from the actual generated code, so you have to analyze the error message to determine where the error occurred. It is often very useful to refer to the compilers documentation to understand the meaning of certain reported errors or warnings.

Below is a small list of generic and common build symptoms with possible causes:

Unknown command, command not found, the name specified is not recognized

- Is your compiler installed correctly?
- Is your make program configured and installed correctly?
- Are you linking with the correct Services Libraries?
- Are the Rose RealTime environment variables set?

Redefinition of basic types or multiple declarations for X

- Do you have any name conflicts?

Unresolved symbol or undeclared identifier

- Have you configured the necessary inclusions, libraries, or object files?
- Are you missing dependencies between classes in your model?
- Does the Capsule role have the same name as the Capsule?

Missing Class Dependencies

Missing dependencies are a common source of compilation errors. You need to identify which capsules and classes depend on other classes in your model. That way when you compile a capsule or class, it will find the definition of the class you depend upon. Also if that class's interface changes, the build process will automatically rebuild all the capsules and classes that depend upon it.

To resolve these types of errors add the correct dependencies between classes using the **Build > Add Class Dependencies** Wizard or by manually creating a dependency relationship between classes.

Capsule Role Name Same as Capsule Name

An error of this type is generated when a capsule role instance has the same name as a capsule.

To resolve the problem, give the capsule role a different name than the capsule class. A good rule in situations like these is to always start capsule class name with an uppercase letter, and capsule roles with lowercase letters.

Linking Wrong Services Library Set

If you find that the output stream has many undefined messages, you may be accessing an inappropriate Services Library set.

Code generated with this release of Rose RealTime does not work with the Service Libraries of previous releases (and vice versa).

The compiler must match the library set being used since most compilers do name-mangling on variables. For example, if your compiler target is NT and your compiler is MSVC++ 6.0, use the target NT40 and the x86-VisualC++-6.0 library entry.

Compiler Not Installed Correctly

If the CC environment variable is either undefined or the default compiler and linker defined in **libset.mk** cannot be found, or CC is defined to something that either cannot be found or is not a compiler, you will sometimes see the following in the raw output field of the Build Results window if your make command is not found:

```
The name specified is not recognized as an internal or external
command, operable program or batch file.
```

Compile a Simple Hello World Program

To ensure that your compiler and linker are installed correctly, write and build a small test program from outside of Rose RealTime. Ensure that it compiles and runs successfully.

Check Environment Variables

You should be able to invoke your compiler and linker from outside of Rose RealTime on the command line. If you cannot you should verify your PATH environment variable and ensure that the directory that contains the tools for your platform is in the path.

Review Your Compiler Flag Settings

You should review your compiler settings. Have you overridden the default compiler, or have you added flags to the component specification compiler tab?

System Does Not Understand the Make Command

Your OS does not understand make or the make is being used is in some interesting way different from what Rose RealTime expects. You will sometimes see the following in the raw output field of the Build Results window if your make command is not found:

```
The name specified is not recognized as an internal or external
command, operable program or batch file.
```

Check Environment Variables

You should be able to invoke **make**, **gmake**, or **nmake** from outside of Rose RealTime, for example, on the command line. If you cannot you should verify your PATH environment variable and ensure that the directory that contains the make utility for your platform is in the path.

Ensure that Component has Correct Make Types Configured

Also, you should ensure that the make name and types defined in the component specification compilation make and generation make tabs represent the correct type of make installed on your system.

Name Conflicts

Odd compile errors can easily be caused by name conflicts, such as naming a capsule role the same as a signal name. You must be aware of the name scoping of various entities in your programming language to ensure that no conflicts occur.

In Rose RealTime, most named entities have capsule-level name scope. For example, within a capsule class, the following are named entities, and any duplication among the names of these entities may cause problems:

- capsule roles
- attributes
- ports
- operations

As well, symbols declared as extern, as in included .h files, are generally part of the global name space, and must not conflict with any names of entities in your model. There are several names that are reserved for the OS/Compiler, such as 'return' and 'exit'. The list of known reserved words is listed in the section below

Some name conflicts are more insidious in that the conflicting names are actually 'compile-time' compatible, and slip by the compiler, resulting in a run-time error that may be difficult to track down. Typically, this means that the elements actually have a common superclass, or, if the error occurs in a function which takes a void * parameter, it is because the entity that was passed as a parameter was not the expected one.

Missing Header Files, Object Files, and Libraries

Most models make calls to external code libraries, even if it is just the basic system calls (such as printf, scanf, cin, cout, and so forth). The include files that define these calls must be specified prior to compilation, so that the compiler can resolve these references. Likewise, the libraries or object modules that contain the actual compiled definitions of these external classes and functions must be specified so that the linker can resolve the symbol references.

You will likely see the following type of error message if you have not included the correct header files:

```
'print_this' : undeclared identifier
```

You will likely see the following type of error message if you have not specified a library or object files that should be linked into your model:

```
unresolved external symbol "int __cdecl print_this(void)"
fatal error XXXXXX: 1 unresolved externals
```

To resolve these types of errors add the correct files or search directories to the component specification dialog under the inclusions or libraries tabs.

Compile Fails on Valid C++ Models with VC++ 5.0 or VC++ 6.0

The $INCLUDE and $LIB environment variables may not be properly set. Ensure that your compiler binaries are on the path and that the $INCLUDE and $LIB environment variables are set (for example, they could be set for the user who installed VC++, but not set for another user). Set the environment variables. Refer to the VC++ documentation for further details.

Error loading Capsule (“could not spawn process”)

If the executable (`capsule1.exe`) is stored on an NFS server then the NFS client must be configured to have execute permission set.

Error Linking Capsule (“error from nmake”)

If the executable (capsule1.exe) is stored on an NFS server then the NFS client must be configured to have execute permission set.

Windows NT Compilation Command Line Limits

If you encounter a compilation error message that complains about the command line being too long, the cause may be that the length of your compile or linker has exceeded a limit.

Windows NT compilation has command line limits in two areas: source compilation and linking. Both limits have been explored for the Visual C++ 5.0, VRTX PPC Microtec 1.4 and Tornado 1.0.1 PPC Cygnus 2.7.2 compilers.

Source File Compilation

The variables in source compilation are the update name, the $ROSERT_HOME path, compilation options, the local working directory and include directories. The only compiler that has a measurable limit is VRTX. The command line limit is 768 characters.

A workaround for the problem is to reduce the number of include directories by combining include files. Other solutions are to shorten paths and names for the variables listed in the previous paragraph.

Linking

The variables in linking are the update name, the $ROSERT_HOME path, the link options, the number and name length of libraries, the library search paths and the local working directory. The link limits are shown below:

- Visual C++ 5.0: more than 20875 characters
- VRTX PPC Microtec 1.4: 4147 characters
- Tornado 1.0.1 PPC Cygnus 2.7.2: 4150 characters
- HPUX 10.20: 16384 characters (make: couldn't load shell.stop)

A workaround for the problem is to shorten paths and names for the variables listed in the previous paragraph.

Running and Debugging 21

Contents

This chapter is organized as follows:

This chapter describes running and debugging Rose RealTime models.

Execution Basics

After a component has been built successfully, you can run the resulting executable. If you have Purify installed, you can run the executable with Purify, to customize error detection for each component in your program. After the component has been built, see *Running a Component Instance with Purify* on page 315. If you do not have Purify installed, see *Running a Component Instance without Purify* on page 317.

Rose RealTime provides an execution environment that can be used to execute and observe component instances on a processor (a type of node).

While a component instance runs, you can control and observe its execution. This functionality is very powerful: it allows a component instance to be observed at the modeling language level, rather than at the source code level.

Tasks

1. Creating a component instance
2. Running a component instance with Purify
3. Running a component instance without Purify
4. Observing a running component instance

Creating a Component Instance

Before running a component that has been built, you must first assign an instance of the component to a processor.

Tasks

1. Select the **Deployment View** folder, right-click and from the popup menu click **New > Processor**.
2. A new processor with the default settings for your platform is created.
3. Double-click on the Deployment diagram to open it.
4. Drag and drop the processor onto the Deployment diagram.
5. Then drag and drop a component from the **Component View** model browser on to the processor that was just added to the Deployment diagram.

   Note: You can also create a component instance by dragging and dropping a component from the model browser onto a processor in the model browser or to the **Processor Specification - Detail** tab.

6 Open the processor's specification dialog, and change to the Details tab. Under the Component Instances list you should see the new component instance that was created.

7 From the **Model View** browser you can also see the list of component instances associated with their respective processor.

Running a Component Instance with Purify

If you do not have Purify installed on your system, see *Running a Component Instance without Purify* on page 317.

After the component is built and a component instance has been created, the instance can then be run and observed. Purify detects errors in your own code as well as the components your software uses.

The **Run with Purify** item is only visible if you have Purify installed.

The processor must have the same operating system as the toolset, otherwise the **Run with Purify** item will be grayed out. For example, a component instance with a Unix processor must be running on a Unix operating system.

If you are using a UNIX operating system, ensure that you linked with Purify during the build.

Tasks

If you have configured an active component, then once the build is complete, you can use the execute icon from the toolbar (press F5), or select **Build > Run with Purify** from the main menu to automatically run all the component instances selected in the Build Settings dialog.

You can also run any component instance by selecting the component instance from the model browser, right-clicking and selecting **Run with Purify** from the popup menu.

After you select **Run with Purify**, you will be prompted to select **Yes** if you haven't got a build. After you answer the prompt, it may take a minute or so before the toolset finishes running the executable, especially for a large model.

While a component instance runs with Purify, follow these steps to set up execution control from the toolset:

1. A console window appears and you must ensure that the following is displayed (for Windows NT users).

   Note: A console window only appears on host-based targets. Other tools are required to see console windows on targets.

   If the observability line below is not shown in the console, ensure that the observability check box and observability port have been configured in the Component Instance specification.

```
Purify for Windows NT,

Copyright (C) 1993-2000 Rational Software

All rights reserved.

Version 2001.03.00 Early Access; Build: 3142;

WinNT 4.0 1381 Service Pack 5 Uniprocessor free

Instrumenting:

   Compile.EXE 241726 bytes


Purify: while processing file
z:\versions\models\myfiles\build\Compile.EXE:

Note: Instrumentation repeating with 6 additional entry points.

Rational Rose RealTime C/C++ Target Run Time System
Release 6.20.B.03 (+c)
Copyright (c) 1993-2000 Rational Software
rosert: observability listening at tcp port 8978
```

2. Bring control back to the toolset by clicking on any part of the toolset. You will notice a new tab called RTS has been added on you model browser. The browser contained in this new window is called the RTS Browser. It is used to control the execution of a running component instance. You can run and control multiple component instances from within Rose RealTime, for each running instance there is a separate RTS Browser tab.

3. Click on the new tab to show the RTS Browser.

4. The execution control buttons are at the top of the RTS browser. Press the Start button to start the execution of the loaded component instance. Everything printed from your model to stdout and stderr will be shown in the console window that appeared when the component instance was loaded.

5 After you exit the RTS browser, the Purify window appears with the Purify results. For information on how to interpret the results, see “Interpreting the Purify Log Reports” on page 297. For information on how to save the Purify results to a file, see “Running from outside the toolset” on page 318.

6 When you are finished running the component instance with Purify, press the shutdown button. The component instance is killed and control is returned to Rose RealTime.

Note: You can also control the execution of a component instance by using the entries in the Build section of the main menu, or in the popup menu of a component instance.

Interpreting the Purify Log Reports

The Purify output is displayed in a tree control listing all exceptions in order of occurrence. When running on a UNIX platform, each exception report consists of a message. When running on a Windows platform, each exception report consists of a message preceded by an icon, to indicate the severity.

- Messages preceded by a blue circle containing the letter **i** are for information only.
- Messages preceded by a red circle containing the letter **i** indicate that there is a user error.
- Messages preceded by a yellow triangle containing an exclamation mark (!) are warning messages. They usually indicate memory leaks.

If the message text is bold, it indicates that there is something in the model you can see; usually a user error, such as a memory leak.

If several levels of message text are bold, you can scope down to the actual message which points to the line of code changed by the user. You can double click on the bold messages to see the section in the code that caused the message.

Running a Component Instance without Purify

After the component is built and a component instance has been created, the instance can then be run and observed. There are two basic ways of running component instances. They are both described below.

Tasks

If you have configured an active component, then once the build is complete you can use the execute icon from the toolbar (press F5), or select **Build > Run** from the main menu to automatically run all the component instances selected in the Build Settings dialog.

You can also run any component instance by selecting the component instance from the model browser, right-clicking and selecting **Run** from the popup menu. If the **Run** item is grayed out, it is probably because the target control scripts configuration is pointing to the wrong directory in the Processor specification.

While a component instance runs, follow these steps to setup execution control from the toolset:

1 A console window appears and you must ensure that the following is displayed.

Note: A console window only appears on host-based targets. Other tools are required to see console windows on targets.

If the observability line highlighted below is not shown in the console, ensure that the observability check box and observability port have been configured in the Component Instance specification.

```
Rational Rose RealTime C/C++ Target Run Time System

Release 6.20.C.00 (+c)

Copyright (c) 1993-2001 Rational Software

rosert: observability listening not enabled
```

2 Bring control back to the toolset by clicking on any part of the toolset. You will notice a new tab called RTS has been added on you model browser. The browser contained in this new window is called the RTS Browser. It is used to control the execution of a running component instance. You can run and control multiple component instances from within Rose RealTime, for each running instance there is a separate RTS Browser tab.

3 Click on the new tab to show the RTS Browser.

4 The execution control buttons are at the top of the RTS Browser. Press the Start button to start the execution of the loaded component instance. Everything printed from your model to stdout and stderr will be shown in the console window that appeared when the component instance was loaded.

5 When you are finished running the component instance, press the shutdown button. The component instance is killed and control is returned to Rose RealTime.

Note: You can also control the execution of a component instance by using the entries in the Build section of the main menu, or in the popup menu of a component instance.

Observing a Running Component Instance

A very powerful feature of Rose RealTime is the ability to observe a running component instance at the model level. This kind of high-level debugging is not what most developers are used to. More conventionally, developers converted design models to source code. When it was compiled and run, the only way to trace the execution was at the source code level. The design model representation was of no use.

In Rose RealTime you can see the triggered transitions, active states in the state monitors, and watch the dynamic structure animated in the structure monitor. In addition, you can use probes to trace the messages being passed in the system.

Tasks

Observe a running capsule instance by opening monitors and message traces:

1. Once you have followed the steps to run your component instance, change to the RTS browser tab and press the Start button.
2. Expand the top-level capsule folder and select a leaf capsule instance. Non-leaf capsules instances represent the class of the instances.
3. Right-click on a capsule instance, and from the popup menu select **Open State Monitor**.

   A monitor window appears, and you should be able to see the state machine of this capsule instance. The current state is highlighted in black. In addition the last transition fired is drawn in black.
4. Select the Probes tool from the monitor toolbox. Place a probe onto a state by moving the probe cursor over the state then clicking the left mouse button to apply the probe to the state.
   Select the probe that you have just applied to a state, and from the popup menu choose **Open Trace Window**.
5. The opened trace window shows all messages that occur in this state. Follow similar steps for adding probes to ports, and junction points.
6. Notice that any new probe that is added to a monitor is also added to the Probes folder in the RTS Browser. You can perform common operations on probes by using the popup menu from the Probes folder.

Rose RealTime Execution Interface

The execution control of component instances is separated into two main functions: the target control of the component instance and the observability of a component instance. The target control interface provides an interface for automating the tasks related to running, loading, and terminating component instances. The observability interface provides the ability for the Rose RealTime toolset to connect to a running component instance and provides a visual view of the running instance.

Target Control Programs

In order to allow control of component instances on different platforms, easy customization, and support for other targets, the target control utilities are implemented as a set of external executables and scripts that are invoked from the toolset to perform the various target control tasks.

These scripts and executables for target control are located in the following directory:

```
$Target_scripts = $ROSERT_HOME/bin/tc/"host"
```

Below the tc directory (tc for target control) is a list of the hosts on which Rose RealTime can run. And within each of these directories is a list of platforms for which there are control utilities. For example, in the `$ROSERT_HOME/tc/win32` directory there are other directories, for example, win32, tornado, and tornada2. This shows that for a toolset running on a Windows platform the toolset can control component instances for Windows and Tornado platforms, meaning that they can be run, loaded, terminated automatically by Rose RealTime.

The Processor specification dialog must be told in which directory to look for the control utilities for the platform that the processor represents. The control options on the component instance menu (run, load ...) are enabled or disabled depending on the control utilities that are found in the directory specified for that processor. For each utility program that the toolset finds in the target control directories, the appropriate menu item is enabled, indicating that the toolset supports the control function for that platform.

Note: You can always manually run, load, etc., a component instance from outside the toolset.

Overriding Target Control

The Operation mode field on a Component Instance specification dialog specifies whether the controls utilities should be used or the component instance will be loaded manually. In the latter case, most of the component instance control menu items are disabled.

Observability Interface

Once a component instance is running (it must be listening to a specified tcp/ip port using the -obslisten command line parameter) the observability interface can connect to the running component instance, and control and animate its execution.

You can observe (connect to) any component instance that was started with observability enabled (listening to the tcp/ip port specified in the component instance spec dialog) even though it was not started with the target control utilities. Use the **Attach Target** option in the component instance to observe a running component instance.

Overview of Observability Options

After you become familiar with building and running within the toolset, you can start debugging your models. There are several options that are available from within execution environment. You should read the details about each option to become more familiar with the following options.

Observability option	**Explanation**
Watches	Use watches to inspect and modify the values of capsule attributes.
Traces	Use traces to see the messages that are being sent within the system.
Injecting messages	Inject test messages into a model to unit test capsules.
Probe break points	Use probe breakpoints to stop a running model when a specified event is received.
Sequence diagrams	Create and save sequence diagrams of message traces between capsule instances.
Command line debugger	Debug a model without the use of the toolset.
Source code debugging	Debug detail code problems using a source code debugger.

Component Instance Menu

The component instance menu provides commands that are used to control the execution of a component instance.

Load

Loads or downloads a component instance to a target platform. The load does not start the execution of the loaded component instance. Use **Run** once it is loaded. This is only used with target platforms that require loading of modules before they are run. For platforms that do not require loading of modules, this menu item is disabled.

Unload

Use only with target platforms that require loading of modules before they are run. For platforms that do not require loading of modules, this menu item is disabled.

Run

Starts the execution of the component instance. If observability is configured to attach at start-up the RTS browser appears. When observability is attached at start-up the component instance is paused, or does not start processing messages, until the start button is pressed on the RTS Browser.

Run with Purify

Starts the execution of the component instance and tests for different aspects, including memory leaks. This menu item appears only if Purify is installed.

Shutdown

Kills the running component instance, closing the RTS Browser if necessary.

Restart

Kills the running component instance and runs another instance. If the instance is running on an target board, the component is reloaded before a new instance is run.

Reload

Used only with target platforms that require loading of modules before they are run. For platforms that do not require loading of modules, this menu item is disabled. This unloads then loads the component without resetting the target board.

Attach Target

Enabled only if a component instance has been run without observability at startup. You can attach observability to a running process at any time, if that the process was started with observability enabled. This menu item can only be used with **Detach Target**.

Detach Target

Detaches observability, meaning that the toolset no longer communicates with the running component instance. This menu item can only be used with **Attach Target**.

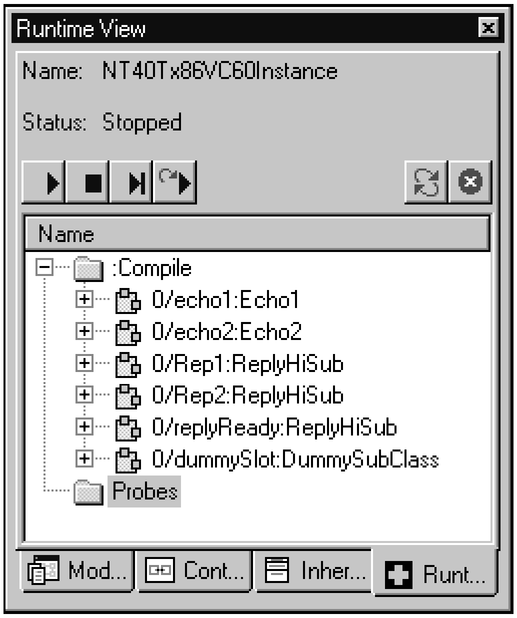
Attach Console

Attaches a console window to the executing target model to interact with the command line model debugger, and so forth.

RTS Browser

The RTS Browser appears as an additional tab on the model browser. It provides an execution interface for controlling the running instance. There is always one RTS Browser for each component instance that is running with observability. The browser is composed of three main parts: an execution control and information pane, a capsule instance browser, and a probes browser.

Figure 84 RTS Browser



Execution Control and Information Pane

This area shows the name of the component instance, the execution status, and the execution buttons.

- **Start** - runs the component instance, allowing all messages to be delivered. The button can be pressed after the component instance has been initialized or is stopped.
- **Stop** - pauses the execution of the component instance. Execution is only paused after the currently executing transition is finished. The button does not halt execution in the middle of a transition. The stop button can be pressed when the model is running.
- **Step** - allows one message to be delivered in the component instance. Pressing step while the component instance is running allows the current executing transition to finish, then delivers the next message, then pauses or stops. The step button can also be pressed when a component instance is stopped or paused.
- **Restart** - causes the current component instance to terminate and starts a new component instance. This button can be pressed at any time; however, it is disabled when you are in manual mode. (Reloads when target is loadable.)

- **Refresh** - updates the status of the capsule instances and probes shown in the browser tree.
- **Shutdown** - terminates the current component instance and effectively stops the execution environment. All execution monitor windows, watches, traces, and any other execution environment windows are closed.

Capsule Instance Folder

This list shows all the capsule instances. All the capsule instances are located in the folder named after the top-level capsule. Instances that have not been created yet, for example, optional or plug-in instances, are shown in the browser but with a red 'X' in front of the capsule instance name.

By default only the capsule instances are shown in the folder. The name shown contains the replication index of the instance within the capsule role, the capsule role name into which the instance was created, and the capsule class name of the instance. For example:

```
0/echo1:Echo1
```

You can also view the capsule roles—the roles into which the capsule instances are created—by right-clicking in the RTS Browser main window and selecting **Filter > Show Roles**. Capsule roles are shown in the list with the name and replication factor of the role, and with the capsule instances created in that role as subentries in the tree browser. For example:

```
o/echo1:Echo1
0/echo2:Echo2
0/Rep1:ReplyHiSub
0/Rep2:ReplyHiSub
0/replyReady:ReplyHiSub
0/dummySlot:DummySubClass
```

Probes Folder

The probes list lists the probes. The popup menu on probe entries in the list allows quick access to the common operations that can be performed with probes.

Monitors

Monitors are read-only views of capsule instances state machine and structure (capsule collaboration) during the execution of the instance. None of the structure or state elements can be modified or moved from the monitor view. The monitor view shows the state and structure components displayed in a lighter shade to emphasize the read-only nature of their parts. Multiple monitor diagrams can exist in the same model.

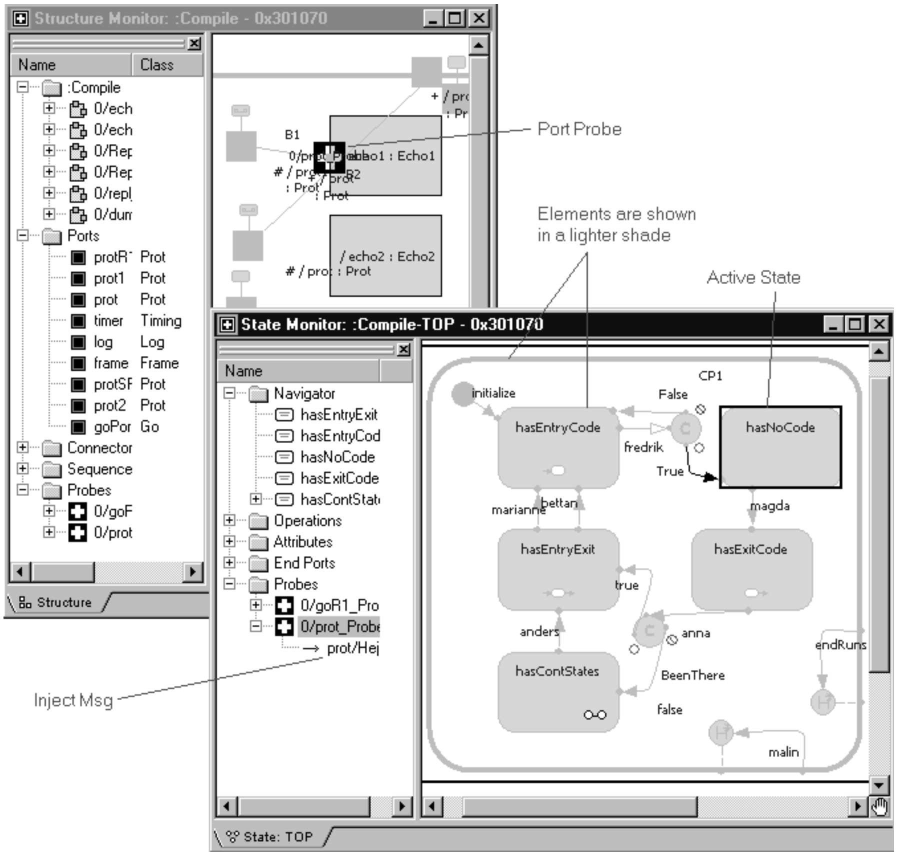
You can right-click on an element and select **Open Specification...** to open the element's specification. As well, on Composites states, transitions, and choice points, you can select **Show Source Code Location**, which opens a dialog that indicates the location of the code for the action associated with the element.

Animation

To allow observation of state and structure, monitors provide visual clues during execution. The structure monitor shows changes in the dynamic structure by showing optional capsule roles that have not been created with the traditional shading. Once they are created, they are shown as fixed capsule roles. Also, current cardinalities of capsule instances are shown. In order to find specific instances of a replicated capsule role as shown in the structure monitor, you can use the cardinality browser tool.

In the State, or state diagram monitor, the current state is highlighted. In addition, if the state diagram shows hierarchical states, the last active state remains highlighted. When a transition is taken, the state monitor highlights the transition.

Figure 85 Capsule state and structure monitors and browsers



Opening a Monitor

Select a capsule instance from the RTS browser, and from its popup menu, select either **Open Structure Monitor** or **Open State Monitor**.

Probes

From within a monitor, you can place probes on ports, junction points, and states by using the Probes tool.

Trace Windows

Trace windows are used to log messages sent or triggering events in a running system. Trace windows show lists of messages, including local state, incarnate and destroy, and import and deport messages. Each row in the list corresponds to one message. Rows can be divided into multiple columns, where each column is used to display different details regarding the message in that row.

You can trace without having the Trace window open. Tracing is turned on by opening a Trace window. However, after tracing is started, closing the window does not stop the tracing. Messages are buffered internally in the probe. Tracing is only stopped by deleting the probe.

There are three different types of trace windows. Each type of trace window shows sets of messages captured at different scopes in the system.

Type of trace	**Scope of message capture**	**Default columns shown**	**Opened by...**
Capsule instance trace	Shows messages exchanged between capsule instances	Time, capsule instances, message signal, optional data	Selecting capsule instances in the RTS browser and choosing **Open Trace Window** from the popup menu.
Port trace	Shows messages coming in or out of a specific port	Time, direction (I/O), priority, signal, data	Creating a probe on a port, then selecting **Open Trace** from the Probes popup menu**.**
State trace	Shows messages that trigger and event in the state machine	Time, port, priority, signal, data	Creating a probe on a junction point or state, then selecting **Open Trace** from the Probes popup menu.

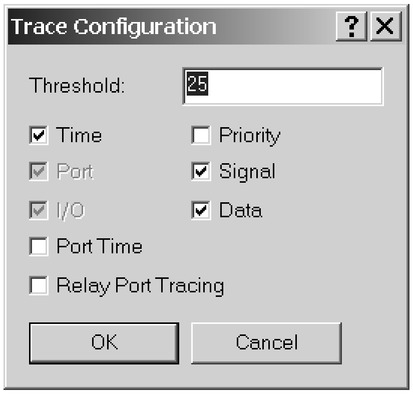
Deleting Messages

Any message can be deleted from the trace by right-clicking on a message in the list then selecting **Delete** from the popup menu.

Trace Configuration

You can configure the information displayed in the trace window by right-clicking in the trace window and selecting **Configure...** from the popup menu. The Trace Configuration dialog appears.

Figure 86 Trace Configuration dialog


Threshold: 25

- [✓] Time
- [ ] Priority
- [✓] Port
- [✓] Signal
- [✓] I/O
- [✓] Data
- [ ] Port Time
- [ ] Relay Port Tracing

You can enable Relay Port Tracing by selecting the **Relay Port Tracing** check box. If this check box is selected, all messages between capsule instances, including instances being received through relay ports, appear in a trace. For messages relayed to sub-capsules within a capsule, the trace shows through which ports these messages were passed.

To observe the message passing, convert the trace to a sequence diagram.

Using Different Types of Traces

Typically when a message fails to flow through a set of capsules as expected, it is important to see where the message flow was first in error. To debug these kinds of errors, first use Capsule instance traces to look at the messages originating and terminating from the capsules in the message flow. If the messages are incorrect and the fault origination cannot be identified, place Probes on specific ports in a composite capsule. Based on whether the messages are still faulty, you can narrow down the cause of the error by further subdivision. Once the faulty capsule has been identified, it is valuable to place traces and message breakpoints on the state machine.

Opening a Sequence Diagram

Selecting this popup menu item opens a dialog that lets you choose the capsule where the generated sequence diagram is saved.

Probes

Probes are used to monitor messages passing through ports and events that trigger transitions in a running capsule instance. They are attached to states, junction points, or ports by using the probe tool, which is available when viewing a state diagram or structure monitor of a capsule instance.

Probes can be placed on instances that have not yet been created, for example, even before the component instance is running. Probes are associated with component instances and are stored with them, such that they do not have to be redefined each time the component instance is run. A component instance's probes are listed in the RTS browser, inside the probes folder.

Probe type	**Can be created on...**	**Description**
port probe	ports, replicated ports	Port probes allow tracing messages passing through the port, or in the case of replicated ports, messages passing through all instances of the port. They also allow you to inject messages to a port.
state probe	junction points, states	State probes placed on junction points allow tracing of events that trigger the associated transition. Probes on states trace all messages that occur in that state. State probes do not allow message injection. They can, however, be used as state break points to stop the execution of the system when a particular probe has been reached.

Use the Probe Specification dialog to configure a probe. You can also use a probe's context menu to quickly open the probe trace window, the Inject window, and activate or deactivate a probe.

Placing Probes on Replicated Ports

If a port is replicated you can place a probe on all instances of the ports by closing the '*' from the port instance browser. This results in a probe that monitors and injects on all instances of the port. You can also place a probe on a particular instance of a port by selecting a particular instance number from the port Instance browser, then placing the probe on the port.

Inject Window

On a port probe specification sheet the Probe Specification—Detail tab allows you to define messages and send them in or out of the port on which the probe is attached.

Inject messages also appear under the owning port probe in both state and structure monitor diagram browsers. The inject message context menu lets you inject, modify, or delete an inject message. Double-clicking the inject message injects the message.

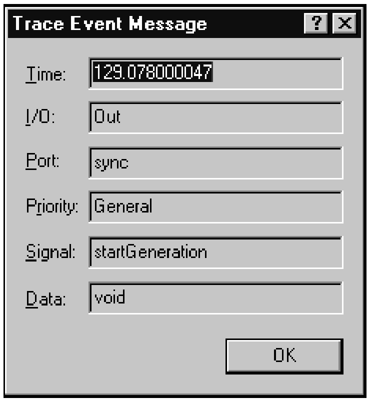
Capsule Instance Trace

A capsule trace window is a type of message trace that shows capsule instances with messages listed in separate columns for recording message flow between instances. The left column displays the time at which the event occurred, the subsequent columns display the source and destination ports, the signal name, optional data, and the capsule instances.

Trace Event Message Dialog

You can right-click on a capsule trace window and select **Open Specification** to open the Trace Event Message dialog (see Figure 78), which contains information about an event message.

Figure 87 Trace Event Message dialog



Time:	129.078000047
I/O:	Out
Port:	sync
Priority:	General
Signal:	startGeneration
Data:	void

Creating a Sequence Diagram from a Message Trace

You can take a snapshot of a message trace at any time and create a Sequence Diagram. Each interaction in the resulting Sequence Diagram is labeled with the signal name. Message lines can cross one another indicating message overtaking. Since the Sequence Diagram is a snapshot of the trace, it is not updated dynamically.

To create a run-time sequence diagram:

1. Open a capsule instance trace.
2. Right mouse click in the message trace window and select **Open Sequence Diagram**.

   A sequence diagram is created from the message trace.

   Note: Only the messages shown in the trace window will appear in the sequence diagram; therefore, if you want to create a sequence diagram with less messages you can pause the running component instance, delete messages from the trace, then create the sequence diagram.
3. You can select a saved sequence diagram and select **Open Trace** to reopen another capsule instance trace.

Dragging Capsule Instances into a Trace

Additional capsule instances can be added to a trace window by dragging and dropping them from the RTS Browser to the trace window. This is useful for configuring the order of instances already in the window, as well as for adding optional instances that were not created at the time the trace was started.

Message Trace Configuration Dialog

This dialog configures a Trace window.

Threshold Field

An integer value used to specify the maximum number of events displayed in the trace window before discarding on a first-in first-out basis. The default threshold is 25.

Note: Messages are also buffered in the running component instance. The larger the threshold, the more memory is allocated in the running component instance. This can be set in the Probe specification threshold.

Column Check Boxes

These correspond to the list columns in the trace window. You can specify which columns are displayed. Each type of trace has its own default columns that are shown. See the trace window help for details on the default columns for the different types of traces.

Execution Watch Tab

Capsule instance attributes can be inspected at run-time and modified from the Watch tab of the Output window. The watch tab has two columns: the name of the attribute and its value.

To add an attribute instance or variable to the watch window, open a state monitor and drag-and-drop the attribute from the Attributes folder into the watch window.

You can also edit the value of a variable by selecting the Value field then entering another value for the variable.

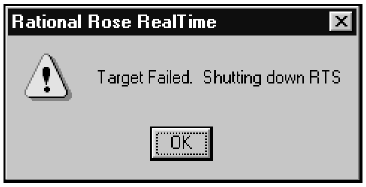
Refreshing the Watch Values

The watch values are refreshed when a message is received by the state of the capsule instance. If the state monitor from where the watch was created is closed, the watch value stops being updated. If the state monitor is closed, you can manually force an update of a watch value by right-clicking on the watch item and selecting **Refresh** from the popup menu.

Run-time Exception While Running a Component Instance

A running component instance can crash suddenly with a run-time exception that could be due to either design errors (sending an inappropriate signal through a port, for example) or coding errors (illegal memory references, for example). Rose RealTime will detect that the process is no longer running and display an information dialog warning that the RTS system will be shutdown.

If Purify is installed on your system, and if a component instance running with Purify crashes, the results appear on the Purify output window.


Rational Rose RealTime

Target Failed. Shutting down RTS

Rose RealTime can help you resolve design errors. For example, problems with state machine logic can be found with a state monitor and message sequencing problems can be found with traces. However, when your model contains detail level coding errors that cause exceptions, the best tool for resolving these problems is your source level debugger. You can add source breakpoints from within a state monitor to automatically launch a source code debugger to help you resolve detail level coding errors.

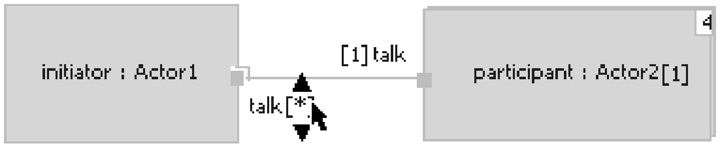
Instance Browser

The instance browser tool is useful for selecting and examining a particular instance of a replicated port or replicated capsule role in a structure monitor.

To use the instance browser:

1 Open a structure monitor that shows either replicated ports or replicated capsule roles.

2 Move the cursor over the cardinality field. The cardinality field is shown at the end of a capsule role name or port name in square brackets. Notice that two black arrows appear, one above the cardinality field and another below.



3 Select the top or bottom arrow to select a particular instance. For ports you can select the '*' entry in the instances list to select all port instances.

Note: As you change the cardinality you may notice the capsules border and shading change to reflect the state of the instance you are viewing.

Source Code Debugging

In addition to the observability debugging tools, you can also use the native debugging facilities at your disposal. Occasionally you have to step through your code to really find out what is happening. Rose RealTime can be configured to automatically start up an external source code debugger when a breakpoint is reached.

Using the breakpoint tool from the state monitor toolbox, you can place source code break points on any element in your state monitor that contains detail level code, including

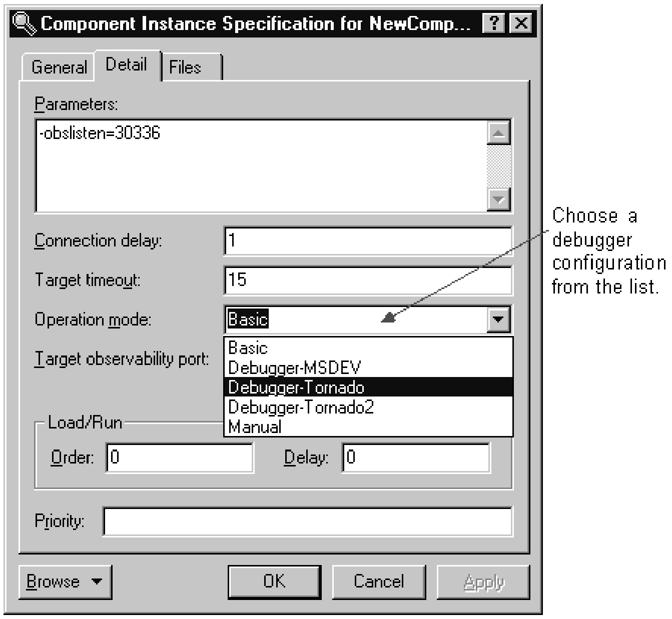
- Transitions
- State entry/exit actions
- Branches

Note: Actions map directly to an operation in the source code so that when the external source debugger hits a breakpoint the breakpoint will always be at the beginning of the operation.

To set source code breakpoints when running a component instance, follow these steps:

1. Rose RealTime must know that a component instance is to be loaded with the source debugger. The component instances specification (Details tab) controls the selection of the source debugger to use, which is one of the options that is available for target control.

The appropriate debugger must be chosen from the **Operation mode** field in the component instance specification dialog.


Component Instance Specification for NewComp...

General | Detail | Files

Parameters:
-obslisten=30336

Connection delay: 1

Target timeout: 15

Operation mode: Basic

Target observability port:

Load/Run

Order: 0 Delay: 0

Priority:

2 Load the source debugger and component instance by choosing **Load** from the component instance right-click menu or the load button from the toolbar.

At this point the source debugger should be loaded and initialized. The component instance has not run yet, hence the RTS Browser is not visible.

Note: Do not forget to configure the component to generate debugging information when compiled. Refer to your compiler and linker documentation for the specific flags that should be used to include debug info into an executable. If the component instance is loaded into the source debugger without debug symbols the source debugger will usually inform you of this.

3 Run the component instance (see *Running a Component Instance with Purify* on page 315 or *Running a Component Instance without Purify* on page 317).

The RTS browser appears.

4 Start the component instance, and use the breakpoint tool to add breakpoints on transitions, states (you are prompted for entry or exit breakpoint), and choice points.

5 When the breakpoint is hit the debugger pops to the front and displays the source code corresponding to the breakpoint. You can now use the debugger and Rose RealTime to debug your running component instance. Remember, however, that once a breakpoint is hit, you must use the debugger to continue execution of the component instance.

After the source debugger has been loaded, it remains loaded until the Unload command on the component instance is chosen. This means that the source debugger can remain open while the component instance is run and restarted multiple times.

Running from Outside the Toolset

Binary files or executables built from Rose RealTime do not necessarily have to be run from within the toolset. In some cases it is necessary to run, or even download to another machine (usually a RTOS), the executable manually. For example, this is useful if you are using a target for which the target control scripts and programs are not available.

Purify

You can run Purify from outside the toolset and import the Purify results into the Rose RealTime.

1 When using Purify outside the toolset, run the results.

2 Save the results as plain text.

3 Import the results into Rose RealTime by going to the Purify pane and selecting **Import** from the context menu.

If the purify output matches a line of code in the model, then the corresponding line of code in Purify appears bold.

Observability Command Line Parameter

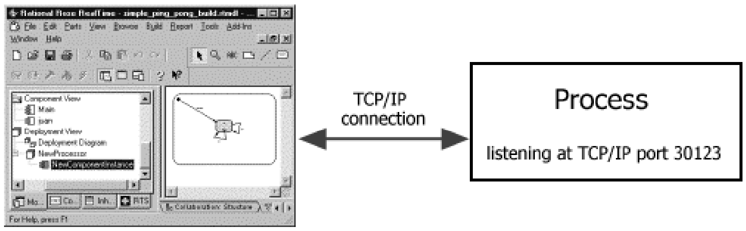
Although the executable is run from outside the toolset, it can still be observed using the observability interface provided by the toolset. If you ensure that the observability command line parameter is passed to the executable before it is run, the toolset can connect to the running model at any time, as well as disconnect. For example, this is how you would start your component instance from outside the toolset:

```
%myProgram -obslisten=30123
```

You can add the following if you want to start running the model immediately:

```
-URTS_DEBUG=go
```

From within the toolset, ensure that in the component instance specification the Target observability port is set to 30123. Now, whenever is required, you can use the **Attach Target** option in the component instance popup menu to attach to the running instance.



Component Instance Menu

To connect to a running process, select a component instance for the correct type of instance that has been run manually. Set the Observability Port in the Component Instance specification dialog to the same value that was specified as a command line argument. Then use the **Attach Target** menu option from the component instance popup menu to connect to the running instance.

Using the Command Line

The result of a successful build of a component is an executable module. You can execute this module directly from the command line if the target environment is the workstation itself; otherwise, you have to download it to the target platform.

You can also start the model, or component instance, automatically using the target control capability.

Command Line Arguments

There are only two Services Library predefined command line parameters that can be used.

```
-obslisten=<tcpip_port>
```

This parameter is to instruct your component instance to listen at the specified tcp/ip port for observability connections from the toolset.

For example:

```
%myProgram -obslisten=67887 -URTS_DEBUG=<a valid debugger command>
```

Use this to pass commands to the Services Library command line debugger, which runs automatically when the component instance is started. For example, the following will quit the debugger automatically and allow the process to run freely.

```
%myProgram -URTS_DEBUG=quit
```

Note: Remember that command line parameters with spaces need quotes.

```
-oblisten -URTS_DEBUG=go
```

Allows you to start running the target and make TO connections at a later time.

Application-Specific Command Line Arguments

You can supply additional command line arguments for use by your component instance model, as you would for any other application. If the component instance is run from the toolset, you can specify command line arguments in the Component Instance specification dialog. The arguments are passed on the command line after the name of the executable, for example:

```
%myTopActor foo 99
```

See accessing command line arguments from within a model for more information.

Loading and Running Component Instances on Embedded Targets

The requirements for running a process on a host platform and on an embedded platform are somewhat different. For clarification, the term host platform refers to the platform on which Rose RealTime is running. Embedded platform refers to a platform that is not running the toolset. For example, before anything can be run on an embedded target, it must first be loaded or downloaded to the target. This step is not required when simply running on a host platform. It is also common to restart the target board, meaning that a soft reboot is performed.

Utility Scripts

For the reasons mentioned above, the execution options are different when running on a host platform or on a target platform In order to support loading, resetting, restarting, and running of component instances on several different target platforms, a set of scripts and executables are invoked from the toolset. Rose RealTime comes with a set of supported target control utilities.

Component Instance Specification

The Component Instance specification dialog contains settings that control the way in which the component instance are run or loaded.

Specification Contents

The Component Instance specification dialog contains the following tabs: **General**, **Detail**, **Purify** (if installed) and **Files**.

Component Instance Specification - General tab

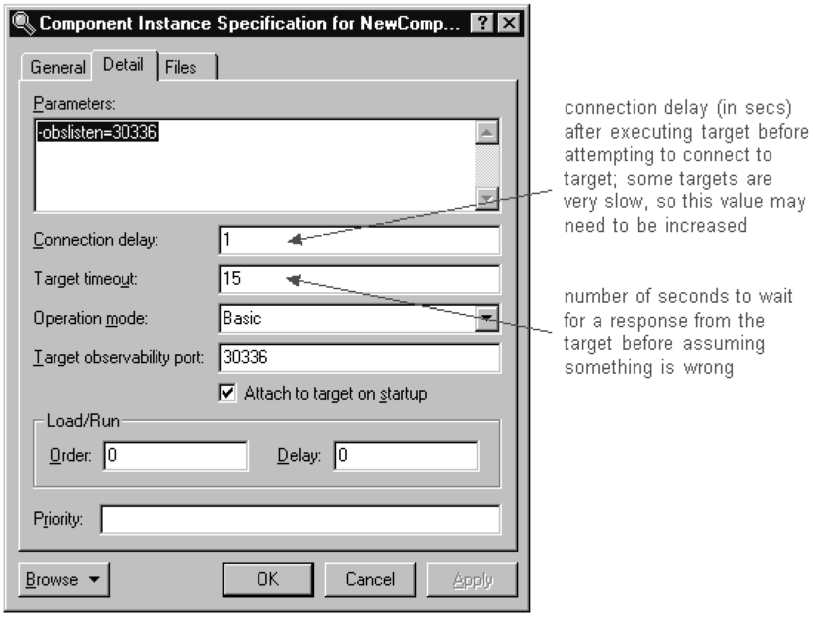
Name

The name of the component instance.

Note: This is not the name of the actual executable that was created from the build.

Component Instance Specification - Detail tab

Figure 88 Component Instance Specification - Detail tab


Component Instance Specification for NewComp...

General | Detail | Files

Parameters:
-obslisten=30336

Connection delay: 1

Target timeout: 15

Operation mode: Basic

Target observability port: 30336

[✓] Attach to target on startup

Load/Run
Order: 0 Delay: 0

Priority:

connection delay (in secs) after executing target before attempting to connect to target; some targets are very slow, so this value may need to be increased

number of seconds to wait for a response from the target before assuming something is wrong

Parameters

Text in this field represents command line arguments that are passed on the command line when the component instance is loaded. The content of this field is passed as is.

Operation Mode

The operation mode specifies the target control configuration for the component instance. The Basic option configures the component instance to use the target control utilities to load and run the component instance. The Manual option instructs the toolset not to attempt to load the component instance.

In addition, if you are setting source level break points Probes, you will have to select the debugger that will be loaded by the target control scripts for your platform. Basic mode is implied when one of the debugger options is not selected.

The options are:

- **Basic** - Use the target control utilities to automatically load and run the component instance.
- **Debugger MSDEV** - Use the Microsoft Visual Studio debugger to load and run the component instance, as well as for setting, clearing, and displaying breakpoints. For more information, see *To configure MSDEV Debugger mode:* on page 351.
- **Debugger Tornado** - Use the target control utilities to load and run the component instance with. Use the Tornado debugger for setting, clearing, and displaying breakpoints. For more information, see *To configure Tornado for Debugger mode:* on page 351.
- **Debugger Tornado 2**- Use the target control utilities to load and run the component instance with. Use the Tornado 2 debugger for setting, clearing, and displaying breakpoints. For more information, see *To configure Tornado for Debugger mode:* on page 351.
- **Debugger xxgdb**[1] - Use the GNU xxgdb debugger to load and run the component instance, as well as for setting, clearing, and displaying breakpoints. (UNIX only). For more information, see *To configure xxgdb Debugger mode:* on page 353.
- **Manual** - The toolset does not attempt to load the executable. The user must manually load the executable.
- **EMVT**- Use EMVT (Embedded Microsoft Visual Tools) to load and run the WIndows CE component instance. For additional information on using the Windows CE option, see *To configure a conponent instance for Windows CE, follow these tasks:* on page 347.

Overview of Observability Options

Attach Target Observability on Start-up

Check this item if you would like the toolset to automatically observe a component instance when it is run by the target control scripts. You can always connect the toolset to the process at some later time.

Target Observability Port

Specify a TCP/IP port number to use for connecting the toolset's execution environment to the target executable. The port number must not already be in use by another process.

Load/Run

- **Order** - An integer value representing the relative order in which this component instance is loaded, or run, in relation to other component instances listed and selections in the Build Settings dialog. Lower numbers are run first.
- **Delay** - An integer value representing the number of seconds to delay before the component instance is loaded or run. This is useful when simultaneously running multiple component instances specified in the Build Settings dialog. If you want to

---

1. The **xxgdb** integration works differently from the MSDEV and Tornado. Follow these steps:

   1.Build the desired component with the appropriate debug options, for example, **-g**.

   2.In the component instance specification, select Debugger-xxgdb for Operation Mode.

   3.Start TO.

   4.Open the desired State Monitors. You may need to “step” to get access to them.

   5.Set breakpoints on the appropriate elements within the desired State Monitor.

   6.Restart Target Observability.

   Breakpoints are now enabled.

   7.Run the model.

   8.Remember to continue in the debugger when you hit a breakpoint. The toolset gets no indication that a breakpoint was hit.

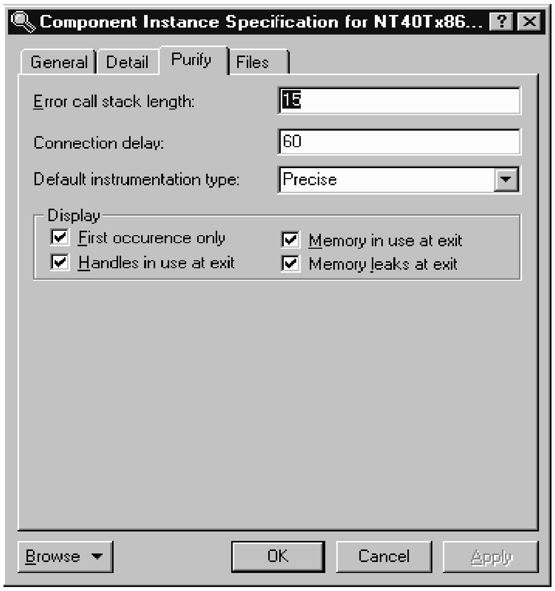
   9.If you remove breakpoints, they will not take effect until you restart the model again.

Ensure that one component instance has time to start correctly before running the other - for example, if they need to communicate - you can specify a run delay for the second component.

Overview of Observability Options

Component Instance Specification - Purify tab

Figure 89 Component Instance Specification - Purify tab


Component Instance Specification for NT40Tx86...

General | Detail | Purify | Files

Error call stack length: 15

Connection delay: 60

Default instrumentation type: Precise

Display

☑ First occurence only ☑ Memory in use at exit

☑ Handles in use at exit ☑ Memory leaks at exit

Error Call Stack Length

- The maximum number of call stack levels which you want Purify to record for error locations in the program.
- This setting affects whether two errors are considered identical (those with the same message type and error location call stack) and displayed as one message with a count of repeated occurrences, or considered different and displayed as separate messages.
- Purify uses the error location call stack to determine whether a message is a unique or repeat occurrence. Specifying a larger number gives Purify more call stack levels to compare and increases the chances that Purify will display a message as a unique occurrence.

Connection Delay

- An integer value representing the number of seconds to delay before attempting to connect to the target. This allows Purify time to instrument the executable as necessary. For a large module, you will need to adjust the connection delay to be more than the default of 60 seconds.

Default Instrumentation Type

- The level of error checking and coverage monitoring on a per module basis. Select one of the following:
    - precise (default) - provides full run-time error detection and precisely pinpoints problems in any component in the program.
    - minimal - provides quick instrumentation for modules whose errors are of less interest.
    - exclude - excludes DLL's which may cause your program to malfunction when `SetWindowsHook()` is called.

Display

- **First occurrence only** - displays only the first occurrence of a message with a count of repeated, identical occurrences, for all Purify sessions
- **Handles in use at exit** - displays the handles that are in use when you exit a program, for all Purify sessions
- **Memory in use at exit** - displays allocated blocks of memory, to which there are still pointers, at exit. This allows you to fix large amounts of memory in use in long running programs, to avoid out-of-memory problems
- **Memory leaks at exit** -displays memory leaks (allocated blocks of memory to which there are no pointers) found when you exit a program, for all Purify sessions

See the Purify documentation for more information and details on Purify, and for descriptions of possible error messages.

Processor Specification Dialog

This dialog allows configuration of the type of processor that this element represents, in addition to the processes (component instances) that will run on the processor.

Specification Contents

The processor specification dialog contains the following tabs: General, Detail, Files.

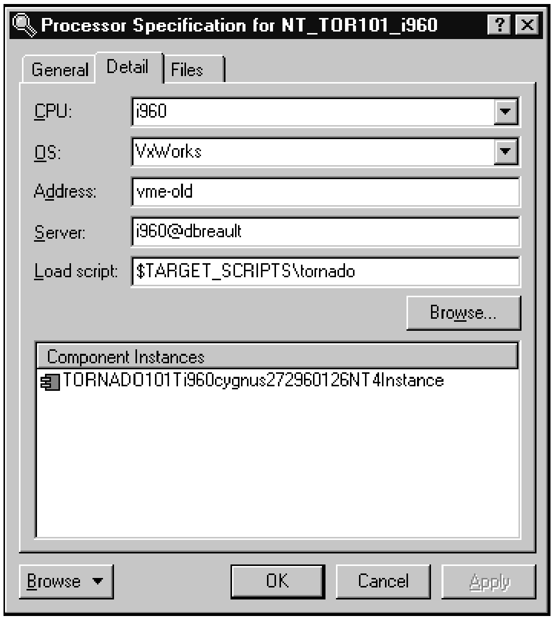
Processor specification - General Tab

Name

A name for the processor. The name appears on the deployment diagram, but the name is not used for execution purposes. The actual target id is specified using the address field on the Detail tab.

Processor Specification - Detail tab

Figure 90 Processor Specification - Detail tab


Processor Specification for NT_TOR101_i960

CPU: i960

OS: VxWorks

Address: vme-old

Server: i960@dbreault

Load script: $TARGET_SCRIPTS\tornado

Component Instances

TORNADO101Ti960cygnus272960126NT4Instance

CPU

Name of the type of central processing unit for this processor element.

OS

Name of the operating system running on this processor.

Address

Network address for the processor. This field can contain a hostname or an IP address. For example jhostl or 145.34.5.6.

Note: For systems not connected to a network, you must use 127.0.0.1 in this field.

Server

In some environments there is a server that handles loading and executing of a component instance for the target RTOS. This is the name or the address of this server.

Load script

Path to the target control utility directory that contains the scripts and programs that are responsible for loading and unloading processes on that processor. If this field does not point to a valid script directory you will not be able to execute component instances from within the toolset.

Component Instances

This is the list of component instances that will run on this processor. You can add a component instance to this list by dragging and dropping a component instance from the model browser to this list. Dropping a component instance on a processor results in the creation of a process. You can also right click and select **Insert**. The Create Component Instance dialog appears in which you can select a component to create an instance from and give it a name. See the Processor specification dialog for process details.

Browse

When you click the **Browse** button, the **Select Directory** dialog appears from which you can locate the Target Scripts directories.

Using Windows CE

To allow control of component instances for the Windows CE platform, the target control utilities are implemented as a set of external executables and scripts that are invoked from the toolset to perform the various target control tasks.

These scripts and executables for target control are located in the following directory:

```
$Target_scripts = $ROSERT_HOME\bin\tc\win32\wince
```

For a toolset running on a Windows platform, the toolset can control component instances for a Windows CE target platform; a component instance can be run, loaded, and terminated automatically by Rose RealTime.

To configure a conponent instance for Windows CE, follow these tasks:

- *To specify the Windows CE target control configuration for the component instance:* on page 347
- *To configure the Windows CE component instance:* on page 349
- *To run and load the Windows CE component instance:* on page 350
- *To unload the Windows CE component instance:* on page 350

To specify the Windows CE target control configuration for the component instance:

1 Establish an ActiveSync connection between your Desktop and the Windows CE device.

For information on establishing an ActiveSync connection, see your Windows CE documentation.

2 Configure for your Windows CE envirement.

Microsoft Embedded Tools includes batch files to configure your environment for different processors. For example in /EVC/WCE300/bin, there is a batch file called WCESH3.bat that sets up an environment for an **sh3** target. Batch files for other targets are available in the same directory.

Note: Ensure that the environment variables are configured for your target processor for your specific CPU.

The operation mode specifies the target control configuration for the component instance. For Windows CE, you can specify either **Basic** or **Debugger** modes. The Basic option configures the component instance to use the target control utilities to load and run the component instance. If you want to set source level breakpoint, you can specify a debugger that is loaded by the target control scripts for your Windows CE platform.

For instructions on setting Debugger mode, see *To configure Windows CE for Debugger mode:* on page 352.

3 **Optional:** To use **Basic** mode, prior to starting Rational Rose RealTime, type the following at the Command Prompt:

Set RRT_WINCE_TARGET_DIR=\<*directory_name*>\

where <*directory_name*> is the name of the location of download the model executable and the TCKill agent for your target. If the directory name is not set, the model executable and the TCKill agent are downloaded to the root directory on the target.

4 Start Rational Rose RealTime.

5 Open an existing model, or create a new model.

6 In the **Model View** tab in the browser, right-click on **Deployment View** and click **New > Processor**.

   The **Processor Specification** dialog must be told in which directory to look for the control utilities for the Windows CE platform. The control options on the component instance menu (such as Run and Load) are enabled or disabled depending on the control utilities found in the directory specified for that processor.

7 In the **CPU** box, select the appropriate CPU for your **target** processor.

8 In the **OS** box, select **Windows-CE**.

9 In the **Address** box, specify the network address for the processor.

   This field can contain a hostname or an IP address. For example jhostl or 145.34.5.6.

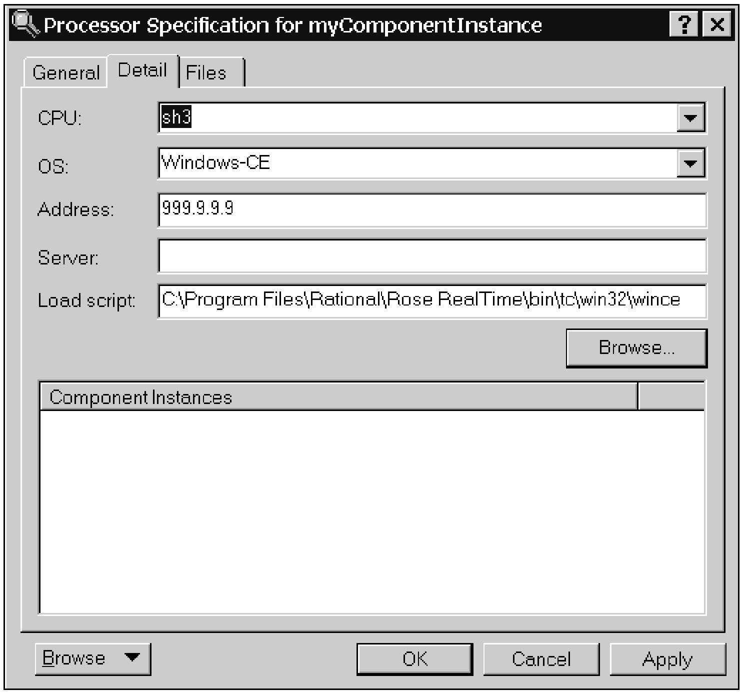
10 Leave the **Server** box blank.

11 In the **Load Script** box, type the following:

```
C:\Program Files\Rational\Rose RealTime\bin\tc\win32\wince
```

   Note: The path to the target control utility directory that contains the scripts and programs responsible for loading and unloading processes on that processor. You must specify the fully-qualified path. If this field does not contain a valid script directory, you cannot execute component instances from within the toolset.

Your **Processor Specification** dialog will look similar to following:


Processor Specification for myComponentInstance

CPU: sh3

OS: Windows-CE

Address: 999.9.9.9

Server:

Load script: C:\Program Files\Rational\Rose RealTime\bin\tc\win32\wince

12 Click **OK**.

To configure the Windows CE component instance:

1 In the **Model View** tab in the browser, drag a component from the **Component View** folder to your Windows CE processor to create a new component instance.

2 Select the new component instance.

3 Right-click and click **Open specification**.

4 Click the **Detail** tab.

5 In the **Connection delay** box, specify an interger, (the time in seconds) that specifies how long the toolset waits before listening for a connection from the target.

6 In the **Target timeout** box, specify an interger, (the time in seconds) that specifies how long the toolset listens for the connection from the model running on the target.

7 In the **Operation mode** box, select **Basic**.

8 Click **OK**.

To run and load the Windows CE component instance:

1 In the **Model View** tab in the the browser, select the new component instance from the **Deployment View** folder.

2 Right-click and click **Load**.

The component instance is loaded onto the Windows CE target.

The component must be successfully built before it can run. If the **Attach Target observability** option was set on the **Component Instance Specification** dialog and a **Target observability Port number** specified, the execution interface displays to allow you to control the execution of the model.

3 Right-click the component instance and click **Run**.

On your Windows CE device, the model runs, but it is controlled from the toolset on the Desktop. You can now step through your model and observe its progress in the State Machine.

To unload the Windows CE component instance:

Because the component instance was loaded onto the Windows CE target, it must be unloaded later.

1 In the **Model View** tab in the the browser, select the new component instance from the **Deployment View** folder.

2 Right-click and click **Unload**.

Note: For **Basic** mode, the executable on the Windows CE target device is deleted; the TCKill agent remains on the target. If you wish to remove the **TCKill** agent, on the Windows CE target, you must manually delete the **TCKill** agent for your target.

Using Debugger Modes

You can specify any of the following debugger modes:

- MSDEV - (Microsoft Visual Studio - Windows only)
- Tornado
- Tornado 2
- EMVT - (Microsoft Embedded Visual Tools - Windows only)
- xxgdb - (GNU - UNIX only)

To configure MSDEV Debugger mode:

To set source level breakpoint probes, you must select a debugger that will be loaded by the target control scripts for your platform.

Note: Basic mode (the default) is implied when one of the debugger options is not selected.

1 In the **Model View** tab in the browser, select the processor from the **Deployment View** folder.

2 Right-click and click **Open Specification**.

3 Click the **Detail** tab.

4 In the **Operation mode** box, select **Debugger-MSDEV**.

5 Click **OK**.

6 In the **Model View** tab in the browser, select the component instance.

7 Right-click and click **Load**.

8 Righ-click and click **Run**. If prompted to build the component instance, click **Yes**.

Now, you can set breakpoints and debug your model.

To configure Tornado for Debugger mode:

To set source level breakpoint probes, you must select a debugger that will be loaded by the target control scripts for your platform.

Note: Basic mode (the default) is implied when one of the debugger options is not selected.

If you set the operation mode to **Debugger-Tornado** or **Debugger-Tornado2**, you can set break points and debug your model.

Note: Before starting Rose RealTime, you must configure the Tornado environment.

1 In the **Model View** tab in the browser, select the processor from the **Deployment View** folder.

2 Right-click and click **Open Specification**.

3 Click the **Detail** tab.

4 In the **Operation mode** box, select **Debugger-Tornado** or **Debugger-Tornado2**.

5 In the **Server** box, you must specify the name of server that will be the target server.

6 Click **OK**.

7 In the **Model View** tab in the browser, select the component instance.

8 Right-click and click **Load**.

9 Righ-click and click **Run**. If prompted to build the component instance, click **Yes**.

Now, you can set breakpoints and debug your model.

To configure Windows CE for Debugger mode:

To set source level break point probes, you must select a debugger that will be loaded by the target control scripts for your platform.

Note: Basic mode (the default) is implied when one of the debugger options is not selected.

If you set the operation mode to **Debugger-EMVT** for the Windows CE target, you can set break points and debug your model.

1 In the **Model View** tab in the browser, select a component from the **Component View** folder.

2 Right-click and click **Open Specification**.

3 Click the **C++ Executable** tab.

4 In the **Default Arguments** box, you must specify an argument that instructs the executable on how it will comminicate with the toolset. Type the following:

   -obslisten=<*port_on_target_instance*>

   where *port_on_target_instance* is the target observability port.

5 Click **OK**.

6 In the **Model View** tab in the browser, select the Windows CE component instance from the **Deployment View** folder.

7 Right-click and click **Open Specification**.

8 Click the **Detail** tab.

9 In the **Connection delay** box, specify an interger, (the time in seconds) that specifies how long the toolset waits before listening for a connection from the target.

   Note: The default value for **Connection delay** is 1 and is not sufficient for this purpose. If you specify 60 in the **Connection delay** box, this time should be quite sufficient.

10 In the **Target timeout** box, specify an interger, (the time in seconds) that specifies how long the toolset listens for the connection from the model running on the target.

Note: If you specify 120 in the **Target timeout** box, this time should be quite sufficient. Depending on the size of your model, you may need to increase this value further.

11 In the **Operation mode** box, select **Debugger-EMVT**.

12 Click **OK**.

13 In the **Model View** tab in the browser, select the component instance.

14 Right-click and click **Load**.

15 Righ-click and click **Run**. If prompted to build the component instance, click **Yes**.

After the source debugger is loaded, it remains loaded until the **Unload** command for the component instance is selected. This means that the source debugger can remain open while the component instance runs and restarts multiple times.

On the target Windows CE device, it loads the **TCKill** agent for your specific target.

Now, you can set breakpoints and debug your model.

Note: For Windows CE, the EMVT debugger mode does not use the **TCKill** agent.

To configure xxgdb Debugger mode:

Use the GNU xxgdb debugger to load and run the component instance, as well as for setting, clearing, and displaying breakpoints.

Note: Basic mode (the default) is implied when one of the debugger options is not selected.

1 In the **Model View** tab in the browser, select the processor from the **Deployment View** folder.

2 Right-click and click **Open Specification**.

3 Click the **Detail** tab.

4 In the **Operation mode** box, select **Debugger-xxgdb**.

5 Click **OK**.

6 In the **Model View** tab in the browser, select the component instance.

7 Right-click and click **Load**.

8 Righ-click and click **Run**. If prompted to build the component instance, click **Yes**.

Now, you can set breakpoints and debug your model.

Unloading a Debugger

To unload the debugger:

You have to unload target platforms that require loading of modules before they are run.

1 In the **Model View** tab in the browser, select the component instance from the **Deployment View** folder.

2 Right-click and click **Unload**.

Device Specification

The Device specification dialog contains three tabs: the General tab, the Detail tab, and the Files tab:

General tab

Name

The name of the device.

Stereotype

A stereotype label for the device.

Documentation

Use this field to describe the device.

Detail tab

Characteristics

Use the Characteristics text field to specify a physical description of the hardware component. For example, you can describe the kind and bandwidth of a connection, the manufacturer, model, memory, and disks of a machine, or the kind and size of a device. You can set this field only through the specification. This information is not displayed in the deployment diagram.

Files Tab

A list of referenced files is provided here. The files list popup menu allows you to insert and delete references to files or URLs.

You can link external files to model elements for documentation purposes.

Connection Specification

The Connection specification contains three tabs: the General tab, the Detail tab, and the Files tab.

General Tab

Name

The name of the connection.

Stereotype

A stereotype label for the connection.

Documentation

Use this field to describe the connection.

Detail Tab

Characteristics

Use the Characteristics text field to specify a physical description of the hardware component. For example, you can describe the kind and bandwidth of a connection, the manufacturer, model, memory, and disks of a machine, or the kind and size of a device. You can set this field only through the specification. This information is not displayed in the deployment diagram.

Files Tab

A list of referenced files is provided here. The files list popup menu allows you to insert and delete references to files or URLs.

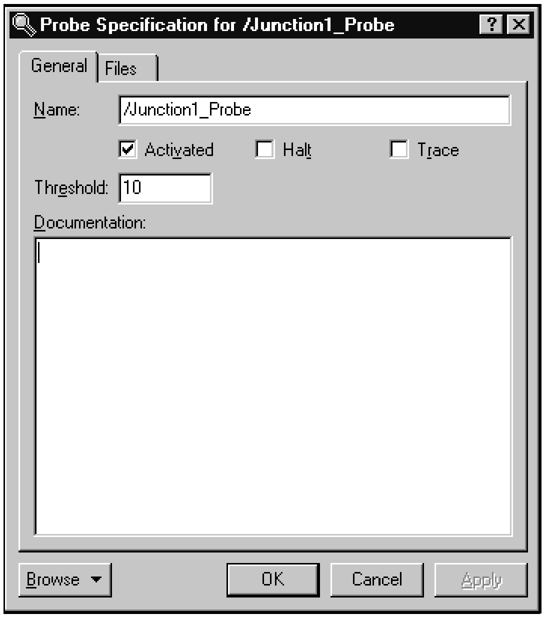
You can link external files to model elements for documentation purposes.

Probe Specification

The Probe Specification dialog contains two tabs: General and Files.

Probe Specification - General tab

Figure 91 Probe Specification - General tab



Probe Specification for /Junction1_Probe

Name: /Junction1_Probe

Activated [✓] Halt [ ] Trace [ ]

Threshold: 10

Documentation:

Name

The name of the probe, which you can edit if you choose.

Activated

Enables the probe.

Halt

Halts the execution at the point of the probe when it detects a message.

Trace

Opens the Trace window.

Threshold

Sets the size of the message buffer on the target.

Documentation

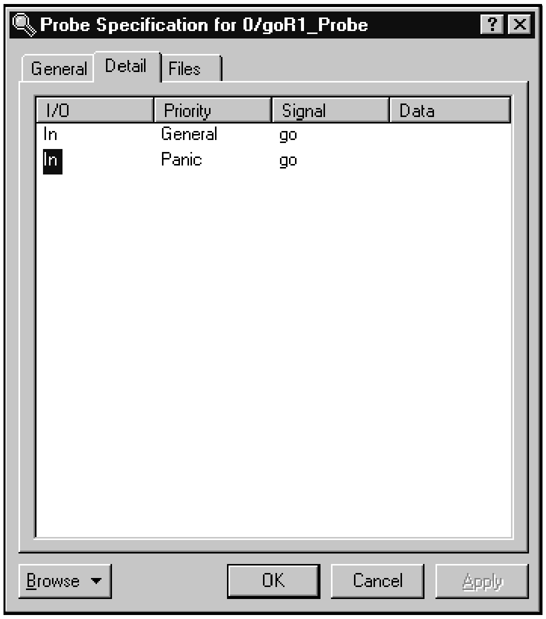
Use to describe this probe.

Probe Specification - Files tab

Allows for linking of external files.

Probe Specification - Detail tab

Figure 92 Probe Specification - Detail tab



I/O	Priority	Signal	Data
In	General	go	
In	Panic	go	

This tab is only available on port probes. It is used to specify and inject messages into the port on which the probe is attached.

Message list

This tab contains the list of messages that can be injected into the port. The list shows the direction (in/out), priority, signal name, and data of each message.

Creating Inject Messages

To create an inject message:

1 Right-click in the list and select **Insert** or press the **Insert** key.

2 A message editor appears in which you can configure the message that you want to send into or out of the port.

Note: You can only choose from the defined signals in the protocol associated with port instance on which the probe is attached.

3 Once the message has been defined, press **OK.**

The message appears in the inject list.

Injected Data Format

The Data area of the inject message is a string representation of the data to be injected with the message. The format of the string depends on the encoding and decoding scheme used by the data type that is being injected.

Therefore, the format of the inject data is linked directly to the encoding and decoding functions. If the encode and decode functions have not been overridden on a data type, the Services Library provides a default ASCII encoder/decoder.

In most cases you will be injecting data using the default ASCII decoder. If this is the case you can use the following syntax to specify the Data area of a message:

Note: You do not have to enclose the expression in double quotes.

```
Default ASCII encoding syntax

<type> ::= <type name>{ <attributes> }

<attributes> ::= <attribute name>{ <attributes> } |

<basic attribute><basic type>,<attributes> |


<basic attribute><basic type>

<basic type> ::= <value> | <basic type>,<value>
```

where

```
<attribute name> is an attribute of a composite type (e.g., a
type composed of other attributes - for example another class)

<basic attribute> is the name of an attribute of a basic type
(int, long, short, char, enum, double, float, string)

<value> is the value of an attribute of a basic type
```

Examples

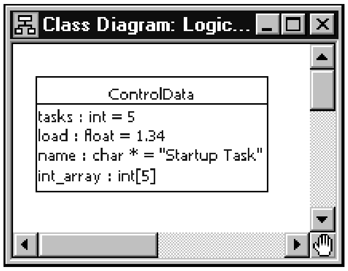
Basic types

If a signal has a basic type that is a data class

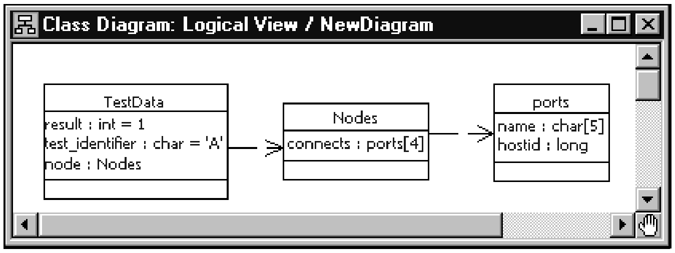
```
int -> int 5

char -> char'a'
```

Classes

Here are two examples of what should be entered into the Data area of an inject message to inject data of the following types. Do not enclose the data in double quotes. The string below each class diagram would be entered as is into the Data area of the inject message.



```
ControlData{tasks 43,load 3.22,name"NodeManager",int_array
0,0,0,0,0}
```



```
TestData{result
1,test_identifier'A',node{connects{name'\0','\0','\0','\0','\0'
, hostid 0},{name'\0','\0','\0','\0','\0',hostid
0},{name'\0','\0','\0','\0','\0', hostid
0},{name'\0','\0','\0','\0','\0',hostid 0}}}
```

Note: To help determine the format of data types remember that the inject data format will always be the same as you would see the data displayed in a trace window.

Injecting a Message

To inject a message that shows in the inject list, select the message and from the popup menu choose **Inject**.

If an error occurs parsing the Data area of the inject message, an error will not be returned to the toolset. The message will simply not get injected. The best method of determining whether a message was injected successfully is to open a trace window on the port into which the message is being injected. If the inject is successful you will see the message in the trace.

Inject messages can also be injected, modified, or deleted in the State monitor browser. They are child elements of port probes.

Using Code Sync to Change Generated Code 22

Contents

This chapter is organized as follows:

This chapter describes how you can use Code Sync to make changes to the generated code from outside a model within an IDE (Integrated Development Environment) or editor of your choice, and recapture the changes back into the model.

Code Sync Overview

The purpose of the Code Sync feature is to provide a facility to capture users' changes made to generated code, back into the model. This allows you to externally modify and debug the generated code outside of the toolset.

Modifying generated code helps to reduce the debug cycle on some RTOS's (Real Time Operating Systems), and allows you to make changes using a third-party IDE. Using Code Sync, changes to the generated code can be reconciled and re-integrated back into the "master copy" of the model files.

For the purposes of this feature description, "externally" means "outside the toolset".

Intended Code Sync Usage

The intended usage of Code Sync is as follows:

1. Build the model from the toolset. See *Starting a Build* on page 297. There must be generated code before Code Sync can function.
2. Browse the generated code using a third-party editor or IDE.
3. Modify the generated code in designated areas only. See "Identifying Code Sync Areas" on page 365.
4. Compile the code externally. See *Compiling Code Externally* on page 367.
5. Run the executable externally to test your changes. For more information, see *Running from Outside the Toolset* on page 337. Return to Step 3 above, until the external debugging cycle is complete.
6. From the toolset, invoke Code Sync. See *Invoking Code Sync from the toolset* on page 367.
7. From the toolset's Code Sync Summary dialog box, accept the desired changes. See *Reconciling Changes in the Code Sync Summary* on page 367.

Limitations

Code Sync cannot be used to create, delete or rename model elements, or to otherwise make structural changes to the model. Such changes must be made using the toolset.

After the generated code has been modified externally, the toolset should not be used to run the externally-built executable until all code Sync changes have been reconciled. For example, although state transitions could be observed and animated by the toolset, the toolset will still show the old transition action code which may be misleading during debugging.

Once the generated code has been modified manually, Clearmake cannot provide complete traceability back to model files, and Clearmake cannot provide wink-in. In Clearmake terms, generated code that has been manually modified is no longer considered a *derived object*, but rather a *view-private* file.

Enabling and Disabling Code Sync

In order for the correct Makefile pattern to be generated for Code Sync, Code Sync must be enabled before the code is initially generated from the toolset.

By factory default, Code Sync is enabled on new components. Code Sync can be disabled from the CodeSyncEnabled flag on the Generation tab of each component, if necessary to accommodate a particular *make* utility or to accommodate local coding conventions.

Components that are dependent on a component with Code Sync enabled, do not necessarily need to have Code Sync enabled.

Identifying Code Sync Areas

The generated code can only be modified in certain designated areas. For convenience, these designated areas are tagged using language-specific comments.

Code Sync Identification Tags

You should only modify code that is delimited by the Code Sync identification tags.

Designated areas for Code Sync are identified in the generated C++ code with the following tags:

```
// {{{USR capsuleClass 'NewCapsule1' tool 'OT::Cpp' property
'HeaderPreface'

<insert or modify code here>

// }}}USR capsuleClass 'NewCapsule1' tool 'OT::Cpp' property
'HeaderPreface'
```

Designated areas for Code Sync are identified in the generated C code with the following tags:

```
/* {{{USR capsuleClass 'NewCapsule1' tool 'OT::C' property
'HeaderPreface' */

<insert or modify code here>

/* }}}USR capsuleClass 'NewCapsule1' tool 'OT::C' property
'HeaderPreface' */
```

Other similar tags (RME tags) are generated for tracing compilation error messages back to the applicable model element. These tags are irrelevant to Code Sync. Code Sync only recognizes code delimited by the Code Sync identification tags.

Designated Code Sync Areas

The following areas are designated as available for Code Sync users:

- Action code for transitions in capsules
- Action code for operation implementations in capsules and data classes
- The HeaderPreface, HeaderEnding, ImplementationPreface and ImplementationEnding fields for data classes and capsules
- The CommonPreface field for components
- Guard code for the event triggers on capsule transitions
- Choice-point condition code for capsules
- Entry Action and Exit Action code for capsule states
- The PublicDeclarations, Protected Declarations, and Private Declarations fields for C++ data classes
- The InitFunctionBody, CopyFunctionbody, DecodeFunctionBody, EncodeFunctionBody and DestroyFunctionBody fields for data classes
- The NumElementsFunctionBody field for capsule attributes
- The ConstructorInitializer field for C++ capsule constructor operations

In some cases where a field is omitted or left as its default, the code generator may generate an optimized code pattern that does not provide the empty Code Sync areas or its identification tags. If you wish to use Code Sync area for an area which has been optimized out, you must provide a non-default value for the field (such as a comment) **within the model**, then re-generate before you can modify that Code Sync area.

Compiling Code Externally

Building a model externally is discussed in the *Guide to Team Development - Rational Rose RealTime*. However, since the code will already be generated and manually modified, it is normally sufficient to compile without generating, as shown in the following example:

```
cd /MyHome/OutputDirectory
cd build
make -f Makefile RTcompile
```

In a multi-component model, it is safer to build from the Component Makefile and iterated through each dependent component's compilation. This is particularly true if a header file was manually modified.

```
cd /MyHome/OutputDirectory
make -f Makefile RTcompile
```

Note that this will check for any required generation for each component, then compile each component. If the model has changed, your manual modifications may be lost (overwritten during generation). Consequently, it is recommended that you do not modify the model while you are modifying the generated code.

Invoking Code Sync from the toolset

To propagate the changes into the model, you need to invoke Code Sync and then decide which changes you want to accept.

Select **CodeSync** from the component's drop down menu. Alternatively, if the component is set as active, click **Build > CodeSync** from the Rose Real Time menu.

Any pending changes to the model-files are written to the file-system. It is not advisable to make further changes at this time, since they will be overwritten upon reconciliation.

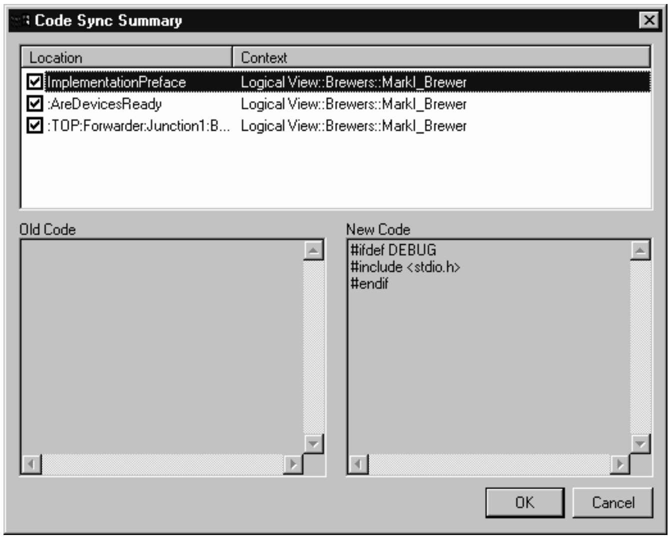
If you wish to abort a CodeSync, click the **Stop-Build** icon from the standard toolbar.

Reconciling Changes in the Code Sync Summary

After Code Sync examines the generated code, a Code Sync Summary dialog appears.

This summarizes the differences, for designated code sync areas, between the generated code and the corresponding elements in the model.

Figure 93 Code Sync Summary dialog



Location	Context
☑ ImplementationPreface	Logical View::Brewers::MarkI_Brewer
☑ :AreDevicesReady	Logical View::Brewers::MarkI_Brewer
☑ :TOP:Forwarder:Junction1:B...	Logical View::Brewers::MarkI_Brewer

```
#ifdef DEBUG
#include <stdio.h>
#endif
```

Location

The location within the model element, of the code that was changed by the user.

Context

The location within the model of the model element, where the changes were made by the user.

Old code block

The appearance of an element of code within the model. If there is no action code, this block will be empty.

New code block

The appearance of an element of code from the generated code that has been modified (appears different from the model). If there is no action code, this block will be empty.

Accepting Changes

To accept changes:

1 From the **Code Sync Summary** dialog, double-click each location you wish to view. The old code block and new code block appears for the selected location.

  You can right-click on a change to bring up its context within the toolset. Be sure to return to the Code Sync Summary before modifying the model.

2 To reject changes that you do not wish to propagate into the model, deselect the check box(es). These rejected changes may include debug information placed in the Code Sync area while debugging within your IDE.

3 Ensure that you have not rejected any code that is required for the model. Click **OK** to accept the selected changes.

Model files are checked out of version control as necessary once the changes are accepted.

Common Code Sync Errors

It is possible to change the model within the toolset before Code Sync is invoked, however, this is not advised. The changed model will be saved when Code Sync is invoked and used during the Code Sync comparison. This can result in either a fatal Code Sync error (if the model changed outside of designated areas), or the model changes may be interpreted as "old code" in the Code Sync Summary dialog (this may be confusing while reconciling changes, and result in the model changes being overwritten). It is recommended not to change the model before invoking Code Sync.

It is possible to change the model within the toolset after Code Sync is invoked, while the Code Sync Summary dialog is visible; this is also not advised. Code Sync reconciliation is based on the unchanged model, and changes to the model may result in reconciliation results getting lost. You may need to view the model while reconciling Code Sync changes, however, you should not modify the model until the Code Sync Summary dialog has been dismissed (by cancel or accept).

Error: Cannot code-sync; file I/O error on: <filename>

This occurs if the code generator cannot open the expected file during Code Sync, for example, if you have started a Code Sync without a previous code generation.

Error: Cannot code-sync <filename> beyond line <lineNum>

This usually indicates that:

- you have modified the code outside the Code Sync identification tags, or
- you have changed the model (for example, changed the CommonPreface) since it was last generated.

Error: Could not find trailing CodeSync tag for [<LocationSpecifier>]

This usually indicates that a starting Code Sync tag does not have a corresponding trailing CodeSync tag, for example, if the trailing tag has been accidentally modified. The Location Specifier (location of modified code, such as ImplementationPreface) and the format of the entire line (including spacing) must match exactly in the two Code Sync tags.

Warning: Use tabs for indenting code-sync regions

The code-generator indents many code-sync regions by one or more tab stops. This warning will appear in the Build Log if, after modification, any line in a code-sync region (including newly-added lines of code) is missing this indentation or is indented with spaces. The region will appear (and continue to reappear) in the Code Sync Summary even if there are no changes to the region. The white-space difference can be resolved by properly indenting the region manually, or by generating the code.

It is recommended that you use an editor which indents with tabs. Furthermore, while the tab-width rarely affects the appearance of the generated code, the code-generator assumes a tab-width of eight characters.

Generating Documentation 23

Contents

This chapter is organized as follows:

Linking External Files to Model Elements

All model elements can have external files linked to them for maintaining documentation or linking requirements.

To link an external file to a model element:

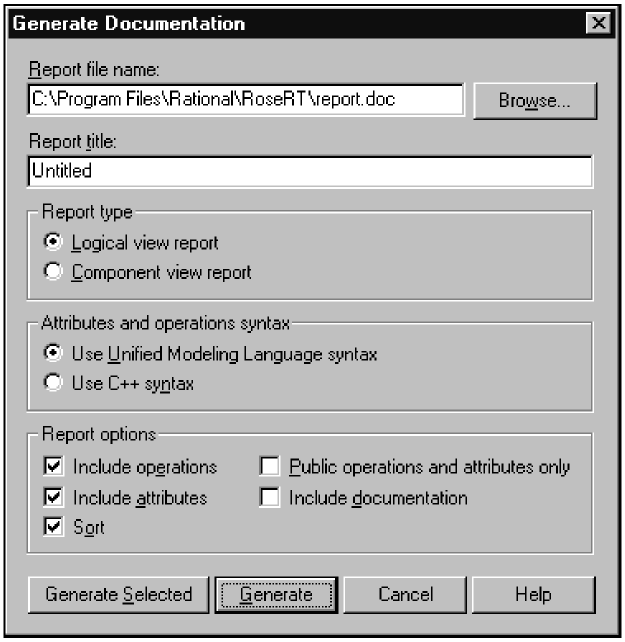
1. Right-click on the model element in the model browser or in a diagram.
2. Select **Open Specification** from the selected item's menu.
3. Click on the Files tab in the Specification Dialog.
4. Right-click under the Filename header.
5. Select **Insert File** from the menu.
6. Use the File Browser to select the appropriate file to link to.
7. Click **Open**.
8. Close the Specification Dialog by clicking **OK**.

The link is stored in the model as a relative path. If the file is moved, or the model is relocated, the link may be broken. You can undo and redo the action of adding a link.

Generate Documentation Dialog

The Generate Documentation dialog shows options for creating documentation from the model.

Figure 94 Generate Documentation dialog



Report file name: C:\Program Files\Rational\RoseRT\report.doc

Report title: Untitled

Report type
- [✓ Logical view report]
- Component view report

Attributes and operations syntax
- [✓ Use Unified Modeling Language syntax]
- Use C++ syntax

Report options
- [✓ Include operations]
- [✓ Include attributes]
- [✓ Sort]
- Public operations and attributes only
- Include documentation

Report File Name

The name for the report file to be created. A File Browser can be used to select the location by selecting the **Browse...** button.

Report Title

Give the report a title.

Report Type

Select the type of document to generate from the following options:

- Logical View Report - generates documentation only for elements in the logical view.
- Component View Report - generates documentation only for elements in the component view

Attributes and Operations Syntax

- Use Unified Modeling Language Syntax
- Use C++ Syntax

Report Options

- **Include Operations** - includes all class operations in the document.
- **Include Attributes** - includes all class attributes in the document.
- **Sort** - specifies that the reports appear in alphabetical order
- **Public Operations and Attributes Only** - includes only publicly visible class operations and attributes in the document.
- **Include Documentation** - includes user-specified documentation entered in specification dialogs in the document.

Generate Selected

Generate documentation for only selected model elements.

Generate

Generate documentation for all model elements.

Cancel

Cancel the operation.

Inserting a Diagram into an MS Word Document

There are two ways to print a diagram into a Microsoft Word document.

Option A

1. Click on the diagram you want to put into your document and select **Edit >Select All**.
2. Copy the diagram to the clipboard using **Edit > Copy**.
3. Position the cursor in the word document where you want the diagram to be placed and select **Edit > Paste**.

Option B

1 Click on the diagram you want to put into your document and select **Edit >Select All**.

2 From the **File > Print** and then click **Options**.

3 Check **Print to file**.

The Print to File dialog appears.

4 Choose the directory in which you want to save the file.

5 Type a file name in the File name: field.

6 Click **Save**.

7 Open Word.

8 Select **Insert > Picture > From File...**

9 Select the file you just saved.

Note: You won't see the actual diagram in your Word document; only a postscript reference is displayed.

Clicking the **Print** icon displays the **Print Specifications** dialog.

Using OLE

OLE is an object-oriented technology, designed for creating, managing and accessing object-based components across process and machine boundaries.

You can create a link between a diagram in your model (the source) and another application such as Microsoft Word. By creating this link, any changes you make to your diagrams are automatically reflected in the document containing the link (the container).

Creating a link

After creating and saving your model, copy the contents of a diagram, either by CTRL + C or **Edit > Copy**.

Note: If the model is new, it must first be saved for this operation to work.

Inserting a link

In an OLE container, for example a Microsoft Word document:

1 Select **Edit > Paste Special**.

2 Click the **Picture** option and **Paste Link**.

3 Click **OK**.

If you select just **Paste** you will get a meta file picture inserted into the container. This meta file is not navigable and becomes native data in the container.

Navigating

To navigate from your OLE container (for example, your Microsoft Word document) to the application, use the steps for opening OLE linked objects, typically, double-click or **Open** from the **Edit Object** menu. The application opens the diagram independent of the Load of Units setting.

Note: Moving your linked files may break the link. It does not, however, affect the object in the container. If the link breaks, you can manually reestablish it from most OLE containers with the Change Source option from the Links dialog.

Editing Diagrams

Unless the unit is read-only, you can edit your linked (source) diagram. When you modify your diagram, the link is updated to reflect the new state. Depending on the application containing the diagram, you may have to do a manual update to see your changes. Refer to your application manual for details.

Customizing the Toolset 24

Contents

This chapter is organized as follows:

Stereotypes

This topic describes the following:

Creating a Custom Framework for Rose RealTime Models

You can create a custom framework from a Rose RealTime model. The contents of the framework define the template to be used when creating new models. For example, if several models with similar characteristics are required, you can create a framework with these characteristics to be used as a template.

To create a custom framework:

1. If you do not have a model file that defines the contents of the framework, create a framework model. You create the framework model in the same way as you would create any other model in Rational Rose Realtime. See *Building Basics* on page 295.
2. Optionally, you may create the following files for the model:
   - a documentation file (.TXT) that contains a description of the framework.
   - an icon file (.ICO) that contains the icon to be used as a symbol for the new framework in the Create New Model dialog.
3. Select **File > New**. The Create New Model dialog appears.
4. Select **New Framework** to enter the Framework wizard.
5. When prompted by the wizard, enter the following information:
   - Framework Name - this name will appear as a label for your framework in the Create New Model dialog.
   - Model file - the name of your framework model file (.rtmdl)
   - Documentation file (optional)
   - Icon file, if created (optional)
6. Follow the prompts and click Finish to exit the wizard.

Creating a New Stereotype for the Current Model

You can create a new stereotype by typing a new name in the Stereotype field of a model element's specification. The new stereotype is then available in the Stereotype field for all model elements of that type (which are assigned the same language) in the current model.

If you want the stereotype to be available in all Rose RealTime models, see *Creating a New Stereotype Configuration File* on page 379. If you already have a stereotype configuration file, skip to *Creating a New Stereotype for all Rose RealTime Models* on page 379.

Creating a New Stereotype Configuration File

The stereotypes in Rose RealTime must be defined in a stereotype configuration file. Rose RealTime is delivered with a default stereotype configuration file, called DefaultStereotypes.ini. If possible, add your stereotypes to that file. If you do not want to use that file, follow these steps to create a new stereotype configuration file:

1 Quit Rose RealTime.

2 Create a text file (called, for example, **`MyStereotypes.ini`**) using Notepad or another text editor, and save it in the Rose RealTime installation folder.

3 Edit the new stereotype configuration file. For information on how to create a new stereotype and add it to a stereotype configuration file, see "Creating a New Stereotype for all Rose RealTime Models" on page 379.

4 Run the Windows Registry Editor (regedit.exe) by selecting **Run** from the **Start** menu. Type "`regedit`" and click **OK**.

5 Locate and select the section entitled `[HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software\Rose RealTime\6.3\StereotypeCfgFiles]` in the registry list.

6 On the **Edit** menu, select **New** and click **String Value**. Give the new registry key the name "file#", where # is the next consecutive number (1, 2, or 3, etc.).

7 Double-click the new key, and enter the name of your configuration file (for example, **MyStereotypes.ini**).

Close the registry. Next time you open a model in Rose RealTime, the stereotypes defined in your new stereotype configuration file will be available in the model.

Creating a New Stereotype for all Rose RealTime Models

You can quickly create a new stereotype by typing a new name in the Stereotype field of a model element's specification. The new stereotype is then available in the Stereotype field for all model elements of that type and language, but only in the current model.

To create a new stereotype and make it available in all models in Rose RealTime:

1 Quit Rose RealTime.

2 Optionally, create icons for the stereotype to be used in diagrams, lists, and diagram toolboxes. See "Controlling the Display of Stereotypes" on page 383.

3 Open the default stereotype configuration file, **DefaultStereotypes.ini** in `%ROSERT_HOME%`.

4 In the stereotype configuration file, add a line for the new stereotype in the section called [Stereotyped Items]. For example, to add the class stereotype Controller to an existing configuration file, add a corresponding line as follows:

```
[Stereotype Items]
Class:Model
Class:View
Class:Controller
```

5 Create a section for the new stereotype, named exactly as the line you added in the [Stereotype Items] section, for example:

```
[Class:Controller]
Item=Class
Stereotype=Controller
```

6 If you have created a diagram icon for the stereotype, specify the name of that file (Metafile). Note that you can use "&" instead of the folder of the stereotype configuration file. For example:

```
Metafile=&MyStereotypeIconscontroller.emf
```

7 If you want to create a diagram toolbox button for this stereotype, specify the name of the file where you created the corresponding small toolbox icon (SmallPaletteImages) and the location of the icon in that file (SmallPaletteIndex). You can also specify the name of the file where the corresponding large toolbox icon is defined (MediumPaletteImages) and the location of the icon in that file (MediumPaletteIndex). For example:

```
SmallPaletteImages=&\MyStereotypeIcons\small_palette_icons.bmp
SmallPaletteIndex=3
MediumPaletteImages=&\MyStereotypeIcons\medium_palette_icons.bmp
MediumPaletteIndex=3
```

8 If you want to graphically display this stereotype in specification lists or in the browser, specify the name of the file where you created its list icon (ListImages) and the location of the icon in that file (ListIndex). For example:

```
ListImages=&\MyStereotypeIcons\list_icons.bmp
ListIndex=2
```

9 Add any other setting needed to define the new stereotype. For a list of all available settings, information on the meaning of each setting, the possible values, and the default values, please refer to the “Stereotype Configuration File” topic in the online help. Note, however, that you only have to include settings for which you want to give other values than their default values.

10 Save your changes to the stereotype configuration file.

11 Run Rose RealTime. View the Log tab to ensure there were no problems loading your icons.

12 If you created a diagram toolbox icon for the new stereotype, and want to add it as a button on a diagram toolbox, please see “Adding Stereotypes to the Diagram Toolbox” on page 381.

The new stereotype is now available in Rose RealTime. For information on how to control the display of the new stereotype in diagrams and in the browser, see “Controlling the Display of Stereotypes” on page 383

For detailed samples on user-defined stereotypes, please refer to http://www.rational.com/products/rosert/

Creating Stereotypes for Classes

To create a stereotype for a class:

1 Double-click on the class in the model browser to open the Class Specification.

2 Select the General tab.

3 In the Stereotype field, type the name of the stereotype for the class, or select it from the pull-down menu beside the field.

You can use any label for the stereotype. It does not have to be one of the built-in stereotype labels.

4 Click **OK** to close the specification dialog.

Adding Stereotypes to the Diagram Toolbox

To make a stereotype available as a button on a diagram toolbox:

1 The stereotype and a corresponding diagram toolbox icon have to be created and made available in Rational Rose RealTime. For information on how to do that, see *Creating a New Stereotype for all Rose RealTime Models* on page 379.

2 Select **Tools > Options**, to open the Options Dialog.

3 Select the Toolbars tab. Under Customize Toolbars, click on the diagram type you want to change the toolbar for.
or in an open diagram, right-click in the diagram toolbar and click **Customize**.

The Customize Toolbar dialog is displayed. The left-most column provides the list of available icons.

4 Select the icon you want to appear on the diagram toolbar and click **Add**.

Creating Stereotype Icons

For each stereotype, four different icons may be supplied:

- A diagram icon (to customize the appearance of model elements with this stereotype in diagrams).
- A small and a large diagram toolbox icon (to be able to add a button for this stereotype to the diagram toolbox). Two different sizes correspond to the Use Large Buttons option on the Toolbars tab of the Options dialog.
- A list view icon (to graphically display the stereotype for model elements in specification lists and in the browser).

Creating a Diagram Icon

Diagram icons have to be in Windows Metafile format (.wmf) or Enhanced Metafile format (.emf) . You can download drawing packages that support these formats at various shareware sites on the Internet. Enhanced Metafiles are recommended if possible.

1 Using a vector-based (as opposed to bitmap) drawing application, draw your icon the size you want it to appear in Rose RealTime. It is best not to use a drawing application that forces the icon to fit a certain area, such as a page, as is the case with PowerPoint.

2 Consider the following: Make sure that the scaling factor is set to 100% when deciding on the icon's size. Use colors if you like. If you want the name of the model element to appear within the stereotype icon, leave some blank space for it.

Select the icon and export it in either the Windows Metafile format or the Enhanced Metafile format. If you use CorelDraw, make sure the **Include header** option is checked if you save your selection as a Windows Metafile.

Controlling the Display of Stereotypes

As stereotypes are refined model element types, it is important to be able to distinguish them in the model. The stereotype can be indicated in several different ways in the browser and in diagrams. See "Stereotype Display" on page 389 for more information.

Controlling Stereotype Display in the Browser

To control how stereotypes are displayed in the browser:

1 On the **Tools** menu, click **Options**, and click the Browser tab.

2 Clicking **Show Stereotype Name** displays the stereotype name and icon of stereotypes in the browser. Also clicking **Hide stereotype name if there is an icon for it** hides the name and displays only the icon.

Controlling How Existing Stereotypes Display in a Diagram

To control how existing stereotypes are displayed in a diagram:

1 Select the model element in the diagram.

2 Click **Diagram Object Properties** from the **Edit** menu, or use the context menu.

3 To control the display of relationship stereotypes, use the **Stereotype Label** option.

4 To control the display of, for example, a class, a device, or a component, click **Stereotype Display** and select the appropriate option from the displayed menu.

5 To control the display of operation and attribute stereotypes in the class compartment, use the **Show compartment stereotypes** option.

Controlling the Display of Stereotypes Added to Diagrams

To control how stereotypes that are added to diagrams hereafter are displayed:

1 Select **Tools > Options > Diagram** tab.

2 To control the display of relationship stereotypes, use the **Show labels on relations and associations** option under **Stereotype display**.

3 To control the display of, for example, class, device, or component stereotypes, use the **None**, **Label**, **Decoration and Label**, **Label Only** or **Icon** options under **Stereotype display**.

4 To control the display of operation and attribute stereotypes in class compartments, use the **Show stereotypes** option under **Compartments**.

Note: For user-defined stereotypes, the stereotype display may be controlled by the settings in the stereotype configuration file where the stereotype is defined. Any such settings override the settings on the Diagram tab of the Options dialog.

Toolset Options

This section describes the Options Dialog, Customizing the Diagram Toolbox, and the Customize Toolbar Dialog.

Options Dialog

The Options dialog provides control over many general properties of the model.

The Options dialog contains the following tabs: General Tab, File tab, Font/Color Tab, Diagram Tab, Filtering Tab, Compartments Tab, Browser Tab, Toolbars Tab, Editor Tab, and Language/Environment Tab.

General Tab

At Startup options

Reload last workspace

Automatically loads the last workspace (and corresponding model) that were in use when the tool was last shut down.

Show splash screen

Toggles whether the splash screen at appears at startup. The default is set to true.

Emulate REI

When enabled, Rose RealTime emulates Rose 2000 from a COM server perspective at startup. This lets you use Rose 2000 Add-Ins with Rose RealTime. Rose 2000 and Rose RealTime are both REI servers. Which one is used to serve a request depends on the specific launched/shutdown sequence of Rose 2000 and Rose RealTime instances that occurred on the server system. Regardless of whether it emulates REI, Rose RealTime always serves as an RRTEI server.

The option can be overridden by the following command line arguments:

- -emulateREI: Emulates REI, regardless of the default specified in the option dialog.
- -noEmulateREI: Does NOT emulate REI, regardless of the default specified in the option dialog.

Picklists

Show Classes

The ShowClassesInPickLists setting works with selection lists to provide a defined set of types to choose from. The picklists are used to define such things as return types and argument types. You can also change this setting directly in the rose.ini file.

Error Log

Log size

Sets the number of lines in the error log.

Log warnings

Enables warnings to be sent to the log. The default is set to true.

Log commands

Sends a description of executed commands to the log. The default is set to true.

Undo

Undo level

Sets the number of undo levels supported. A higher number consumes more memory.

Technical Support

Email address

Sets the email address to which error files are automatically sent if the tool crashes. These error files contain only internal callstack information from the tool. They do not contain any model-specific information.

Test button

Tests the given email address to ensure its validity.

File tab

Save options

Use Temporary File

Enable this option to write to a temporary file whose name is derived from the destination file. Once the temporary file has been completely written, the temporary file is copied or renamed to the destination file.

Create Backup File

Enable this option to create a backup file.

If Create Backup Files is true and Use Temporary Files is false, and the destination file already exists, the destination file is copied or renamed to the backup file before the Petal file is written to the destination file.

If Create Backup Files and Use Temporary Files are both true, then once the temporary file has been written successfully, the original destination file, if it exists, is copied or renamed to the backup file. The temporary file is then copied or renamed to the destination file.

Keep Two Backup Files

Enables Rational Rose to maintain two backup files: the most recent and the baseline copy. If Create Backup Files is true when writing a backup file, the most-recent generated name for the backup file is deleted. The newly created backup file is assigned the most-recent generated name for backup files. The oldest copy of the file is saved and remains untouched. The oldest version of the file is retained as a baseline copy.

If a backup file does not already exist, a backup file is created and assigned the most-recent generated name.

Update by Copy

Enable this option to manipulate files by copying. The temporary file is copied to the destination and the destination is copied to the backup. If this option is false, manipulation of the files is done by renaming.

Save Settings on Exit

Use this command to save the arrangement of diagram and specification windows and icons when you exit. The next time you open the model, the arrangement that was last saved is displayed.

Use spaces in generated file names

Toggles whether spaces are removed from generated filenames.

Always use generated file names

Use this option to always use generated file names, rather than being queried every time.

Load

When set, the toolset does not query you for missing scratch pad files.

Font/Color Tab

Default font

Invokes the Font dialog, through which you can specify font characteristics.

Documentation window font

Invokes the Font Dialog, through which you can specify font characteristics to be applied to the documentation window.

Code font

Specify a default font for code boxes.

Line Color...

Changes the color of any lines used on diagrams.

Fill Color...

Changes the color of any element fills used on diagrams.

Background color...

Changes the background color for diagrams.

Use Fill Color

You must select the checkbox to see the icons displayed in the colors set in the Line or Fill Color selection. (If it is not checked, a color is defined, but not applied.)

Use background color

Toggles whether to use background color. If not checked, system defaults are used; otherwise, the color specified in Background color... is used.

Diagram Tab

Display

Unresolved Adornments

Enables adornment of icons representing components not currently loaded in the model. The unresolved view adornment is a small octagon containing the letter “M” with a slash through it.

Collaboration Numbering

Enables the display of message sequence numbers on Collaboration diagrams.

Sequence Numbering

Enables the display of message sequence numbers on Sequence diagrams.

Focus Of Control

The DefaultViewFocusOfControl setting is an advanced notational technique that enhances sequence diagrams. Focus of Control is portrayed through narrow rectangles that adorn the vertical lines that descend from each object. You can also change this setting (DefaultViewFocusOfControl) directly in the rose.ini file. The DefaultViewFocusOfControl default setting is Yes.

Default line attributes...

Opens the Line Attributes dialog, which let you define line styles, routing, smoothing, and intersecting links.

- Line style - lets you decide whether line styles are oblique or rectilinear. Note that if you choose Rectilinear, Smoothing is grayed out.
- Routing - lets you decide whether routing is normal, closest distance, or avoids obstructions. Note that if you choose Closest distance, Smoothing is grayed out.
- Smoothing - lets you choose how smooth lines are.
- Intersecting links - let you choose whether to jump links and specify the type of jump. As well, you can choose whether to reverse jump links.

Miscellaneous

Double-click to diagram

The DoubleClick setting specifies what action will occur when you double click on an icon representing a logical package or component package. A checkmark indicates a main diagram will be displayed when you double click on the icon. An unchecked box indicates the specification of a logical or component package will be displayed when you double click on the icon.

Automatic Resizing

Enables the automatic resizing of icons to accommodate text.

Class Name Completion

Activates a popup box listing all current class names. You can select one of these names by double-clicking or by hitting the Enter or Tab key when you highlight the correct name.

Auto-adjust transitions

Enables auto-adjusting transitions on creation. The default is set to true.

Show Diagram Browsers

If not enabled, diagram browsers are not created the first time a diagram is opened.

Note: Once a diagram is opened, its state is saved in the workspace, so this option has no effect.

Stereotype Display

Use the options to control the display of stereotypes in diagrams. The selection is applied to new model elements (except relationships) that are added to diagrams hereafter.

- None - The stereotype is not indicated for new model elements.
- Label - The stereotype name is displayed for new model elements. The stereotype name appears inside angle brackets, << >>.
- Decoration and Label - The stereotype icon (if it exists) is displayed as a decoration in the upper right hand corner of the view. The label is displayed just under the decoration centered above the name.
- Decoration Only - The stereotype icon (if it exists) is displayed as a decoration in the upper right hand corner of the view. No label is displayed.

- Icon - The stereotype icon (if it exists) is displayed for new model elements.
- Show labels on relations - enables the display of stereotype labels on new relationships. The stereotype names appear inside angle brackets, << >>. The selection is applied to new relationships that are added to diagrams.

If you want to display/hide the stereotype name of a previously created relationship in a specific diagram, select the relationship in that diagram. Select **Edit > Diagram Object Properties > Stereotype Display**. On the displayed menu, select the appropriate option. You can also use the same option on the shortcut menus.

If you want to change the display of previously created stereotypes in a specific diagram, select the stereotype in that diagram. Select **Edit > Diagram Object Properties > Stereotype Display**. On the displayed menu, select the appropriate option. You can also use the same options on the popup menus.

Grid

Grid Size

Use this command to specify the grid pitch in pixels. The value that you enter in the Grid Size edit box is saved to the GridSizeX and Y settings.

Snap to Grid

A check mark in the checkbox indicates that new or moved icons will align with a grid whose pitch is specified by the grid size.

UML Options

Aggregation whole to part

Controls which way an aggregation can be drawn. Aggregates can be drawn whole (client) to part (supplier) or vice versa. The default is set to true.

Classifier name on roles

Lets you turn off the classifier name portion of a role label.

Protocol name on ports

Lets you turn off the classifier name portion on ports.

Base UML notation

Converts the structure diagram so that it uses only UML base notation.

Target Observability

Animation timeout

Set a delay for displaying animation of events (state changes) in the state monitor. The delay value is in 1/100ths of a second, i.e., a value of 100 will delay event animation for 1 second. This provides the ability to slow down the animation of a model to make state changes more observable.

Filtering Tab

Class Diagram

Use these options to filter information on the class diagram.

State Diagram

Use these options to filter information on the state diagram.

Structure/Collaboration Diagrams

Use these options to filter information on Structure/Collaboration diagrams.

Compartments Tab

Class

Use these options to display/hide information in the compartments of a class on the class diagram.

Capsule

Use these options to display/hide information in the compartments of a class on the class diagram.

Protocol

Use these options to display/hide information in the compartments of a class on the class diagram.

Browser Tab

Stereotypes

Show stereotype names

Use this option to enable or disable viewing of stereotype names of model elements in the browser.

To display only stereotype icons (if any), select the **Hide Stereotype name if there is an icon for it** option.

To display both stereotype icons (if any) and stereotype names, clear the Hide Stereotype name if there is an icon for it option.

Hide stereotype name if there is an icon for it

The StereotypeBitmapsOnly setting enables or disables stereotype icons, but not stereotype names, of model elements in the browser. This setting can also be changed in the rose.ini file.

Class and package name display

Show related components

Use this option to toggle whether to decorate referenced components in the browser.

Editor Tab

External editor

Specify an external editor to be launched when editing detailed code.

Note: If you use an external editor that requires a console terminal, you must specify an application, such as **`xterm`**, that provides the terminal, followed by the editor command itself.

Example on Solaris: /usr/openwin/bin/xterm -e /bin/vi

Example on HPUX: /usr/bin/X11/xterm -e /bin/vi

Toolbars Tab

The standard toolbar and diagram toolbox properties can be set on the Toolbar tab. The choices are grouped as follows:

- Standard toolbar
  - Show Standard Toolbar - toggles whether the standard toolbar is visible.
  - Enable docking - toggles whether to allow the toolbar to be docked.
  - Use large buttons - toggles whether to display small buttons or large buttons on the toolbar.
    - Diagram toolbar
  - Show Diagram toolbar - toggles whether the diagram toolbox is visible.
  - Enable docking - toggles whether to allow the toolbox to be docked.
  - Lock selection - toggles whether to lock the current toolbox selections.
  - Use large buttons - toggles whether to display small buttons or large buttons on the toolbox.
  - Auto show - toggle whether the toolbox is displayed for read-only diagrams.
- Customize toolbars - provides a list of toolbars whose layout can be customized. Click on a toolbar button to bring up the Customize Toolbar Dialog for that particular toolbar.

Language/Environment Tab

Default Language

Select the language from the available installed language add-ins. When a new class is created, this selection determines which:

- language property tab is displayed for classes
- set of fundamental types is used for picklists
- set of predefined stereotypes is used

When a new component is created, the language is set using this default.

If you do not have any language add-ins, the default language is set to Analysis, which is equivalent to having no default language. If this is the case, analysis types are shown in the picklists and no language property tabs are available.

Default Environment

Sets the default environment on new components.

Customizing the Diagram Toolbox

You can access the Customize Toolbar dialog using any of the following:

- Right click anywhere on the toolbox and then click **Customize** from the shortcut menu.
- Double click anywhere on the toolbox not occupied by a button.
- From the **View** menu, point to **Toolbars** and click **Configure**.

With the exception of the **Separator** button, only one instance of any tool can be placed on the toolbox. Since multiple instances of the **Separator** button are allowed on the toolbox, this button is always available regardless of the number of times it is added to the **Toolbox** buttons list.

Customize Toolbar Dialog

The customize toolbar dialog (opened from the Options Dialog), allows you to change the arrangement of buttons on various toolbars.

Toolbar Button List

The Toolbar buttons list contains the ordered list of all the buttons that will appear on the diagram toolbox. Once buttons are moved onto this list they can be moved to any position.

Add-Ins

Add-In Manager Dialog

The Add-In Manager dialog is used to view, activate or deactivate Rose RealTime Add-Ins.

The dialog shows the add-ins currently loaded, with check boxes beside the add-ins showing which ones are currently activated.

Managing Model Properties

Each Rational Rose RealTime model has its own default properties. These default properties are defined in a property file and are grouped into sets based on:

- **Type of model element** - Class, component, relation, attributes, operations, etc - the objects that make up the model
- **Tool** - Corresponds to a tab in the property specification; a tool can be a programming language tool, such as Java or C++; a database tool, such as Oracle8; a user-defined add-in to Rational Rose, or some other tool.
- **Properties** - The actual properties and property values defined in the set; these must be appropriate to the model element and tool for which they are being defined.

Note: You can define multiple sets of default properties for the same tool and model element. For example, you might want one set of properties for a class with a stereotype of Actor and a different set of properties for a class with a stereotype of Interface. Both of these sets are still considered default properties in that they are predefined for the model. Defining multiple sets saves you work by minimizing the need to override properties as you go.

Displaying or Modifying the Values of Model Properties

1. Display a diagram that contains an icon representing the model element.
2. Select the model element in the diagram.
3. Open the model element's specification. To do so, double-click on an element in a diagram, or click on the element and select **Browse > Specification**.
4. Select the **Code Generation** tab. The model property set attached to the element is displayed in the **Set** field. The model properties related to the model element are displayed in the **Model Properties List.**
5. To edit a model property value, select it and click on it a second time. This places the model property in edit mode.
6. Select your choice from the drop down menu. If no drop down menu is available, you may type in your changes.
7. To complete the edit, click outside the edit box.
8. Click **OK** or **Apply** to commit the changes to the item.

Model properties that are specified explicitly by the item, and hence override the attached model property set value, are drawn in normal text. Model properties that have been changed since the last apply are indicated by an asterisk in the left column.

Removing an Overriding Item Level Model Property

Editing a model property automatically makes it an overriding item-level model property.

To remove the overriding value from the item and once again inherit from the attached model property set:

1 Select one or more model properties and click **Default**.

2 Click **OK** or **Apply** to commit the changes to the item.

Making a Model Property Item Specific

1 Select the model property(s) and click Override.

2 Click **OK** or **Apply** to commit the changes to the item.

Reinstalling the State and Value of the Last Committed Change

Select the model property(s) and click **Revert**.

Attaching a Model Property Set to a Single Element or a Collection of Elements

1 Display a diagram that contains an icon representing a model element.

2 Select the model element in the diagram.

3 Open the model element's specification. To do so, double-click on an item in a diagram, select a diagram item and execute the **Specification** command in the **Browse** menu, or select the specification from the shortcut menu.

4 Select the **Code Generation** tab. The model property set attached to the item is displayed in the **Set** field. The model properties related to the model item are displayed in the **Model Properties List**.

5 Select a different model property set from the **Set** combo box.

6 Commits are made as you move from page to page. Also, as you move from set to set or type to type within the set-level model property page, any changes you have made to the currently displayed set are committed.

Displaying or Editing a Specific Model Property Set

1 Select the element from the diagram. If you are selecting a collection of elements, ensure that all the elements are of the same type. Selecting different model elements will result in a warning.

2 From the **Tools** menu, select **Model Properties > Edit**. The code generator displays the **Code Generation tab** of the **Options** dialog. The kind of model item chosen is displayed in the **Type** field.

3 Select the model property set name in the Set combo box. All the model properties and values will be displayed.

4 Modify model property set values by following instructions to edit a specific model property set, as listed above.

5 Click **Apply** or **OK** to accept your changes.

Note: Changes made to a model property are accepted whenever you activate ANY control in the editor. For example, after editing a model property, you may select another model property to both accept the changes to the original model property and begin editing the newly selected model property.

Creating a New Model Property Set

1 Select a model property set from the Set combo box to base your new model property set off of.

2 Click **Clone**.

3 Type the new model property set name in the dialog and click **OK**. A new model property set is created as a copy of the current model property set.

4 Modify model property set values by following instructions to edit a specific model property set, as listed above.

Deleting a Model Property Set

1 Select a model property set from the **Set** combo box.

2 Click **Remove**. The model property set is deleted from the model. An attempt will be made to find all the elements in the model that reference that set and change those elements to reference the default model property set.

Submitting Problem Reports, Feature Requests and Support Requests

25

Contents

This chapter is organized as follows:

This section describes how to submit problem reports, feature requests and support requests to Rational Technical Support.

Submitting Problem Reports

With Rational Rose RealTime, you can email problem reports to the Rational Software Technical Support department that services your location. When you email a problem report directly from the Rose RealTime application, a wizard guides you through the process, ensuring that you provide the correct information to the Rational Technical Support team. This information includes contact and location information, and a detailed description of the problem that you are reporting.

To submit a problem report:

1. From the **Help** menu, click **Email Technical Support**.
2. A submenu appears, providing you with three options.
3. Click **Problem Report**.
4. The General Information dialog appears.
5. Type your contact and location information in the text areas provided and click **Next**.
6. The Problem Report - Additional Information dialog appears.
7. In the **Defect Title** text area, type a detailed name for the problem that your are reporting.
8. Select the type of problem that you are reporting from the appropriate list boxes.

9 Describe the problem, using the categories provided in the **Details** area.

10 Click **Next**.

11 The Email Summary dialog appears.

12 Ensure that the information that appears in the Email Summary dialog is accurate.

Note: The email information displayed in the Technical Support Email Address is choosen based on the location information that you provided in the General Information dialog. It is not recommended that you edit the e-mail address.

13 If you want to save or print a copy of the email, click the appropriate button.

14 Click the **Send Email** button to send your email.

Submitting Feature Requests

With Rational Rose RealTime, you can email feature requests to the Rational Software Technical Support department that services your location. When you email a feature request directly from the Rose RealTime application, a wizard guides your through the process, ensuring that you provide the correct information to the Rational Software Technical Support department. This information includes contact and location information, and a detailed description of the feature that your are requesting.

To submit a feature request:

1 From the **Help** menu, click **Email Technical Support**.

A submenu appears, providing you with three options.

2 Click **Feature Request**.

The **General Information** dialog appears.

3 Type your contact and location information in the text areas provided and click **Next**.

The **Feature Request - Additional Information** dialog appears.

4 In the **Request Title** text area, type a detailed name for the Feature that you are requesting.

5 Select the level of urgency for the feature that you are requesting.

6 Describe the feature, using the categories provided in the **Details** area.

7 Click **Next**.

   The **Email Summary** dialog appears.

8 Ensure that the information that appears in the **Email Summary** dialog is accurate.

   Note: The email information displayed in the **Technical Support Email Address** is choosen based on the location information that you provided in the **General Information** dialog. It is not recommended that you edit the e-mail address.

9 If you want to save or print a copy of the email, click the appropriate button.

10 Click **Send Email** to send your email.

Submitting Support Requests

With Rational Rose RealTime, you can email Support requests to the Rational Software Technical Support department that services your location. When you email a Support request directly from the Rose RealTime application, a wizard guides you through the process, ensuring that you provide the correct information to Rational Software Technical Support department. This information includes contact and location information, and a detailed description of the support request that you are submitting.

To submit a support request:

1 From the **Help** menu, click **Email Technical Support**.

   A submenu appears, providing you with three options.

2 Click **Support Request**.

   The **General Information** dialog appears.

3 Type your contact and location information in the text areas provided and click **Next**.

   The **Support Request - Additional Information** dialog appears.

4 In the **Request Title** text area, type a detailed name for the request that you require.

5 Select the level of urgency for the question with which you need help.

6 Type your question in the **Question** text area.

7 Click **Next**.

   The **Email Summary** dialog appears.

8 Ensure that the information that appears in the **Email Summary** dialog is accurate.

Note: The email information displayed in the **Technical Support Email Address** is choosen based on the location information that you provided in the **General Information** dialog. It is not recommended that you edit the e-mail address.

9 If you want to save or print a copy of the email, click the appropriate button.

10 Click **Send Email** to send your email.

Keyboard Shortcuts

A

Contents

This chapter is organized as follows:

General Shortcuts

Table 1 General desktop navigation

Key Name(s)	Description
CTRL+TAB	Move between windows
ALT or META + key	Display the contents of a menu—in combination with the underlined letter in the menu's name
ESCAPE	Close an open menu
ESCAPE	Cancel a dialog
ENTER	Perform the action in a dialog
TAB or SHIFT+TAB	Move between areas of a dialog
SPACE BAR	Select an item in a dialog

Table 2 General shortcuts

Key Name(s)	Description
CTRL + +	Go Inside
CTRL + -	Go Outside
CTRL + A	Select All

Key Name(s)	Description
CTRL + B	Browse specification
CTRL + C	Copy
CTRL + E	Expand
CTRL + F	Find — displays the **Find** dialog
CTRL + I	Zoom in
CTRL + SHIFT + R	Relocate
CTRL + L	Change line attribute
CTRL + M	Zoom to selected
CTRL + N	New — opens the **Create New Model** dialog
CTRL + P	Print
CTRL + O	Open
CTRL + R	Browse referenced items
CTRL + S	Save
CTRL + T	Browse capsule state diagram
CTRL + SHIFT + T	Browse capsule structure diagram
CTRL + U	Zoom out
CTRL + V	Paste
CTRL + W	Fit to window
CTRL + X	Cut
CTRL + Y	Redo
CTRL + Z	Undo
DEL	Delete
ESC	Cancel
F1	Context-sensitive help
SHIFT + F1	Context help cursor
F2	Refresh
F3	Browse previous diagram

Key Name(s)	Description
F4	Browse parent
CTRL + F6	Browse next pane
CTRL + SHIFT + F6	Browse previous pane
SHIFT + F6	Browse class diagram
SHIFT + F7	Browse use case diagram
F8	Edit inline
SHIFT + F8	Browse collaboration diagram
SHIFT + F9	Browse sequence diagram
SHIFT + F10	Browse component diagram
SHIFT + F11	Browse deployment diagram
F12	Options

Scripting Shortcuts

Table 3 Scripting Shortcuts

Key Name(s):	Description
UP ARROW	Moves the insertion point up one line.
DOWN ARROW	Moves the insertion point down one line.
LEFT ARROW	Moves the insertion point left by one character position.
RIGHT ARROW	Moves the insertion point right by one character position.
PAGE UP	Moves the insertion point up by one window.
PAGE DOWN	Moves the insertion point down by one window.
CTRL + PAGE UP	Scrolls the insertion point left by one window.
CTRL + PAGE DOWN	Scrolls the insertion point right by one window.
CTRL + LEFT ARROW	Moves the insertion point to the start of the next word to the left.
CTRL + RIGHT ARROW	Moves the insertion point to the start of the next word to the right.

Key Name(s):	**Description**
HOME	Places the insertion point before the first character in the line.
END	Places the insertion point after the last character in the line.
CTRL + HOME	Places the insertion point before the first character in the script.
CTRL + END	Places the insertion point after the last character in the script.

Debugging Shortcuts

Table 4 Debugging Shortcuts

Key Name(s):	**Description:**
CTRL + A	Select all
CTRL + C	Copy
CTRL + F	Find
CTRL + G	Go to line
CTRL + H	Replace
CTRL + N	New script
CTRL + O	Open script
CTRL + P	Print
CTRL + SHIFT + P	Edit path map
CTRL + R	Replace
CTRL + V	Paste
CTRL + X	Cut
CTRL + Y	Redo
CTRL + Z	Undo
DEL	Delete
ENTER or F2	Displays the **Modify Variable** dialog for the selected watch variable, which enables you to modify the value of that variable.

Key Name(s):	**Description:**
F5	Runs the current script.
SHIFT + F5	Stops script execution
CTRL + SHIFT + F5	Restarts the current script beginning with the line at which it was stopped using the Break command.
F7	Compiles the current script without executing it
F6	If the watch pane is open, switches the insertion point between the watch pane and the edit pane.
F9	Sets or removes a breakpoint on the line containing the insertion point.
SHIFT + F9	Displays the **Add Watch** dialog, in which you can specify the name of a BasicScript variable. The Script Editor then displays the value of that variable, if any, in the watch pane of its application window.
F10	Steps through the script code line by line without tracing into called procedures.
F11	Steps through the script code line by line, tracing into called procedures.
CTRL + BREAK	Suspends execution of an executing script and places the instruction pointer on the next line to be executed.

Build and RTS Shortcuts

Table 5 Build and RTS Shortcuts

Key Name(s):	**Description:**
F5	Runs the selected component instances
SHIFT + F5	Build/Run
CTRL + SHIFT + F5	Restart
F7	Build
F10	Step

Specification Code Editor Shortcuts

Table 6 Specification Code Editor Shortcuts

Key Name(s):	Description:
CTRL + A	Select all
CTRL + C	Copy
CTRL + E	Clear
CTRL + F	Find
F3	Find again
CTRL + H	Launch external editor
CTRL + I	Import
CTRL + L	Select line
CTRL + P	Print
CTRL + R	Replace
CTRL + T	Font
F4	Replace again
CTRL + V	Paste
CTRL + W	Select word
CTRL + X	Cut
CTRL + Z	Undo

Browser Shortcuts

Table 7 Browser Shortcuts

Key Name(s):	Description:
CTRL + B	Browse specifications
CTRL + D	Delete from model
CTRL + SHIFT + G	Get latest

Key Name(s):	Description:
CTRL + SHIFT + I	Check in
CTRL + SHIFT + O	Check out
CTRL + SHIFT + U	Undo checkout

Index