

Addendum

RATIONAL ROSE® REALTIME

VERSION:2003.06.12

WINDOWS/UNIX/LINUX

Legal Notices

© Copyright IBM Corporation. 1999, 2004. All Rights Reserved.

Part Number: 800-026121-000

Version Number: 2003.06.12

This manual (the "Work") is protected under the copyright laws of the United States and/or other jurisdictions, as well as various international treaties. Any reproduction or distribution of the Work is expressly prohibited without the prior written consent of Rational Software Corporation.

Rational, Rational Software Corporation, the Rational logo, Rational Developer Network, AnalystStudio, ClearCase, ClearCase Attache, ClearCase MultiSite, ClearDDTS, ClearGuide, ClearQuest, ClearTrack, Connexis, e-Development Accelerators, DDTS, Object Testing, Object-Oriented Recording, ObjecTime, ObjecTime Design Logo, Objectory, PerformanceStudio, PureCoverage, PureDDTS, PureLink, Purify, Quantify, Rational Apex, Rational CRC, Rational Process Workbench, Rational Rose, Rational Suite, Rational Suite ContentStudio, Rational Summit, Rational Visual Test, Rational Unified Process, RUP, RequisitePro, ScriptAssure, SiteCheck, SiteLoad, SoDA, TestFactory, TestFoundation, TestStudio, TestMate, VADS, and XDE, among others, are trademarks or registered trademarks of Rational Software Corporation in the United States and/or in other countries. All other names are used for identification purposes only, and are trademarks or registered trademarks of their respective companies.

Portions covered by U.S. Patent Nos. 5,193,180 and 5,335,344 and 5,535,329 and 5,574,898 and 5,649,200 and 5,675,802 and 5,754,760 and 5,835,701 and 6,049,666 and 6,126,329 and 6,167,534 and 6,206,584. Additional U.S. Patents and International Patents pending.

U.S. GOVERNMENT RIGHTS. All Rational software products provided to the U.S. Government are provided and licensed as commercial software, subject to the applicable license agreement. All such products provided to the U.S. Government pursuant to solicitations issued prior to December 1, 1995 are provided with "Restricted Rights" as provided for in FAR, 48 CFR 52.227-14 (JUNE 1987) or DFARS, 48 CFR 252.227-7013 (OCT 1988), as applicable.

WARRANTY DISCLAIMER. This document and its associated software may be used as stated in the underlying license agreement. Except as explicitly stated otherwise in such license agreement, and except to the extent prohibited or limited by law from jurisdiction to jurisdiction, Rational Software Corporation expressly disclaims all other warranties, express or implied, with respect to the media and software product and its documentation, including without limitation, the warranties of merchantability, non-infringement, title or fitness for a particular purpose or arising

from a course of dealing, usage or trade practice, and any warranty against interference with Licensee's quiet enjoyment of the product.

Third Party Notices, Code, Licenses, and Acknowledgements

Portions Copyright ©1992-1999, Summit Software Company. All rights reserved.

Microsoft, the Microsoft logo, Active Accessibility, Active Client, Active Desktop, Active Directory, ActiveMovie, Active Platform, ActiveStore, ActiveSync, ActiveX, Ask Maxwell, Authenticode, AutoSum, BackOffice, the BackOffice logo, bCentral, BizTalk, Bookshelf, ClearType, CodeView, DataTips, Developer Studio, Direct3D, DirectAnimation, DirectDraw, DirectInput, DirectX, DirectXJ, DoubleSpace, DriveSpace, FrontPage, Funstone, Genuine Microsoft Products logo, IntelliEye, the IntelliEye logo, IntelliMirror, IntelliSense, J/Direct, JScript, LineShare, Liquid Motion, Mapbase, MapManager, MapPoint, MapVision, Microsoft Agent logo, the Microsoft eMbedded Visual Tools logo, the Microsoft Internet Explorer logo, the Microsoft Office Compatible logo, Microsoft Press, the Microsoft Press logo, Microsoft QuickBasic, MS-DOS, MSDN, NetMeeting, NetShow, the Office logo, Outlook, PhotoDraw, PivotChart, PivotTable, PowerPoint, QuickAssembler, QuickShelf, RelayOne, Rushmore, SharePoint, SourceSafe, TipWizard, V-Chat, VideoFlash, Visual Basic, the Visual Basic logo, Visual C++, Visual C#, Visual FoxPro, Visual InterDev, Visual J++, Visual SourceSafe, Visual Studio, the Visual Studio logo, Vizact, WebBot, WebPIP, Win32, Win32s, Win64, Windows, the Windows CE logo, the Windows logo, Windows NT, the Windows Start logo, and XENIX, are either trademarks or registered trademarks of Microsoft Corporation in the United States and/or in other countries.

Sun, Sun Microsystems, the Sun Logo, Ultra, AnswerBook 2, medialib, OpenBoot, Solaris, Java, Java 3D, ShowMe TV, SunForum, SunVTS, SunFDDI, StarOffice, and SunPCi, among others, are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Purify is licensed under Sun Microsystems, Inc., U.S. Patent No. 5,404,499.

Licensee shall not incorporate any GLOBEtrouter software (FLEXIm libraries and utilities) into any product or application the primary purpose of which is software license management.

BasicScript is a registered trademark of Summit Software, Inc.

Design Patterns: Elements of Reusable Object-Oriented Software, by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Copyright © 1995 by Addison-Wesley Publishing Company, Inc. All rights reserved.

Additional legal notices are described in the legal_information.html file that is included in your Rational software installation.

Contents

Preface	xi
What's New.	xi
Updates to Web Site Addresses.	xii
Audience.	xii
Other Resources	xii
Rational Rose RealTime Integrations with Other Rational Products	xiii
Contacting Rational Customer Support	xiv
1 Referenced Configurations	17
Requirements for Linux	17
Requirements for Windows NT.....	18
Requirements for Windows 2000	18
Requirements for Windows XP Pro	19
Requirements for UNIX	20
Referenced Configuration Requirements for the Eclipse and Rational Rose RealTime Integration	21
2 Installing Rational Rose RealTime on Linux.	23
Before You Install	23
Requirements for Linux.	24
Installing in Secure Environments	24
Installing Multiple Versions of Rational Rose RealTime for Linux.....	24
Stopping and Restarting an Installation	24
Installation Instructions.	25
After You Install.	29
Installing GNU 3.2	29
Source the Setup Script	29
Unmount the CD-ROM Drive	30
ClearCase Workstation Configuration	30
Command-line Access to the Source Control Tool.	30
Element Type Setup: type Manager	31

Configure the ClearCase Repository	31
Set the Connexis Variable	32
Verify the Connexis Installation.	32
Verifying your Installation using BasicTest	32
Host Configuration Installation Verification	32
BasicTest Server Output	34
BasicTest Client Output.	35
Starting Rational Rose RealTime on Linux.	36
3 General Issues	37
Startup Issues	37
Starting Rational Rose RealTime When an Instance is Currently Running	37
Toolset Freezes on Startup.	38
Virus Scanning Applications Affect Startup and Shutdown	38
Uninstall Issues	38
Windows-Specific Issues	39
Service Pack Requirement Update for Windows	39
Using Hummingbird Exceed 7.1 on a Computer Running Windows Applications and Rational Rose RealTime	39
Using Rational Rose RealTime on Windows XP Pro Configurations	39
Using Rational Rose RealTime without Appropriate Privileges	40
Building Dependencies on Case-Insensitive File Systems.	40
File Association for Compiled Scripts	41
Windows CE GetSystemTime() Function Does Not Return Milliseconds.	41
Symbolic Links with TargetRTS	42
Spaces in Directory Names	42
UNIX-Specific Issues	43
Printing on UNIX.	43
Browser Requirements for UNIX	43
Troubleshooting when the Toolset Freezes on UNIX	44
Rational Rose RealTime Crashes when You Debug Using Tornado 2.2 on UNIX.	45
Refresh Problems with Exceed.	45
Specifying a Location or File Name Containing Spaces (UNIX).	45
Starting vi as an External Editor from Rational Rose RealTime.	46
Unable to Open Some Links in the Online Help	46
Exceptions Occur When You Use Configuration Management in Rational Rose RealTime	46

Case Sensitivity within Paths	46
Window Order Policy	46
Non-GUI-based External Editors	47
Setting the Stack Space Limit	47
Rational Rose RealTime Fails to Build a Component on Solaris 2.6	47
Linux-Specific Issues	47
Using the C++ Analyzer on Linux.	48
Displaying the Version Tree on Linux	48
Using Connexis Viewer on Linux	48
Updating GNU Libraries on Linux 7.3	48
Using Shortcut Keys to Cycle Through Open Specification Dialog Boxes on Linux	49
Using Web Publisher on Linux	49
Viewing the Online Help on Linux	49
Using Context-Sensitive Help on the Preferences Dialog in Eclipse on Linux . .	49
Cross-Platform Issues	49
Synchronizing Code between Rational Rose RealTime and Eclipse.	50
Rational Rose RealTime and Eclipse Integration Does Not Support Java. . . .	50
Using Rational Rose RealTime with a Node Locked License Does Not Warn User About Expiration	50
Using Pathmaps in Rational Rose RealTime	51
Limitations on the Number of Open Windows	51
Limitations in the Specification History List	52
Loading a Workspace might Cause Default Color Settings to Change Permanently	52
Do Not Use \$& When You Define a PathMap	52
Use Caution When Modifying OutPutDirectory	52
Using Rational Rose RealTime on Non-English Installations Causes an Unreadable Font When Viewing Generated Code	53
Unable to use Parameters with the cm_getcaps Script	53
Problems Compiling Java Models	53
Problems Connecting to a Target	53
Using Sequence Diagrams	53
Using the Debugger-xxgdb Tool and Running your Component Instance	53
Using C and C++ Add-ins	54
Code Generator Runs Out of Memory When Generating Very Large Models. .	54
Using the Get and Set Methods in the Attribute and Operation Tools	54
Web Publisher Applet Does Not Load Properly	55
Using the Frameworks Dialog	55

Scoping Descriptors for Nested Classes	55
No Codesync Support for Java	56
Using the GetSelected Functions	56
Using the Find Command Might Return Too Many Results	58
No Support for Automatic Unwired Port Registration for Java	58
Error Occurs When Printing a Diagram	58
ROBERT_NO_FEEDBACK - Prompting for Information When an Exception Occurs	58
ROBERT_TORNADO_TIMEOUT - Modifying the Default Timeout for wtx Commands	59
Referenced Configuration for Nucleus Does Not Include Socket Support	59
Online Help Issues	60
Navigating Through the Online Help	60
Viewing Animated Demonstrations	60
Problems Accessing Rational Rose Help while Running Rational Rose RealTime on Windows	61
Using Context Sensitive Help Might Cause Message to Display	61
Maintaining a Single Favorites List	61
Using the Index Tab in the Online Help	62
Documentation Updates	62
Update to Abort Documentation	62
Update to Code for Example RTMessage_getPortIndex	63
Update to Code for Example RTMessage_getData	64
Update to Documentation for Launching Model Integrator	64
Code Example for rts and RTCapsule_context Causes Compilation Errors	65
Update to Example Model for Type Descriptors	66
Update to Referenced Host Configurations Information	66
Updated Reference to Example Model for Type Descriptors	67
Using Type Descriptor Functions	67
4 Using Globally Unique Identifiers (GUIDs)	69
Advanced Handling of Globally Unique Identifiers (GUIDs)	69
Generating GUIDs	71
Recommended Steps for Enabling GUIDs in Multi-Stream Development	71
Managing GUIDs	72
Known Issues with GUIDs	75

5 Eclipse and Rational Rose RealTime Integration.	77
Integration Overview	77
Communication Overview	79
Configuring a Listener	80
Configuring a Connector	80
Installing the Eclipse and Rational Rose RealTime Integration Software	80
Configuring Preferences in Eclipse	81
Communication Tab	82
Features Tab	84
Path Mappings Tab	86
Specifying Path Mappings	88
Configuring Preferences in Rational Rose RealTime.	89
Generating Code from Rational Rose RealTime	92
Removing Generated Code from Rational Rose RealTime	92
Refreshing an Eclipse Project	93
Synchronizing Code Between Rational Rose RealTime and Eclipse	93
Editing Code in Eclipse	94
Before You Edit.	95
Editing Choice Point, State (Entry and Exit), and Transition Code in Eclipse	96
Editing Operation Code in Eclipse	96
Editing Capsule and Class Code in Eclipse	97
Configuring Build Settings in Rational Rose RealTime	98
Configuring Build Settings in Eclipse	100
Building in Eclipse	101
Navigating to Build Errors	103
Navigating From Rational Rose RealTime to Eclipse	104
Navigating from a State Diagram	104
Navigating from a State Diagram or State Diagram Browser	106
Navigating from the Model View Tab in the Browser	107
Navigating from Eclipse to Rational Rose RealTime	108
Getting Started.	110
Example Workflow	113
Troubleshooting	131

6	OSEK/VDX Support in Rational Rose RealTime	133
	Overview of OSEK/VDX Support in Rational Rose RealTime	133
	Introduction to the OSEK OIL Files	134
	Configuring the TargetRTS and Adapting it for a Particular OSEK/VDX Operating System	135
	Updating Tasks	136
	Index	139

Preface

This addendum provides important information about this specific release of Rational Rose RealTime. The information in this addendum supersedes the information in the online Help and hardcopy documentation included in Rational Rose RealTime version 2003.06.00.

This addendum is organized as follows:

- *Referenced Configurations* on page 17
- *Installing Rational Rose RealTime on Linux* on page 23
- *General Issues* on page 37
- *Using Globally Unique Identifiers (GUIDs)* on page 69
- *Eclipse and Rational Rose RealTime Integration* on page 77
- *OSEK/VDX Support in Rational Rose RealTime* on page 133

What's New

This release includes the following new features:

- *Eclipse and Rational Rose RealTime Integration* on page 77
- *OSEK/VDX Support in Rational Rose RealTime* on page 133

This release includes the following updates in the following chapters:

- *Referenced Configurations* on page 17
- *General Issues* on page 37

Updates to Web Site Addresses

The following table identifies some of the Rational Web addresses that were updated in this release. If you encounter a link that is no longer active, use the <http://www.ibm.com/> link and perform a search on your desired topic.

Area	Web Address	Phone Number
Support	http://www.ibm.com/support/	1-800-IBM-SERV
AccountLink	http://www.ibm.com/software/rational/support/licensing/	1-800-IBM-SERV
Training and Education	http://www.ibm.com/services/learning/	1-800-IBM-TEACH
Technical Publications	http://www.ibm.com/software/rational/support/documentation/	1-800-IBM-SERV
Rational Software Products	http://www.ibm.com/software/rational/	1-800-IBM-SERV

Audience

This guide is intended for all readers, including managers, project leaders, analysts, developers, and testers.

Other Resources

- Online Help is available for Rational Rose RealTime.

Select an option from the **Help** menu.

All manuals are available online, either in HTML or PDF format. To access the online manuals for Windows configurations, click **Help > Contents**, **Help > Index**, **Help > Search**, or for other configurations, you can select the desired .pdf document file from the **<ROSERT_HOME/Help** directory.

- To access Rational Rose RealTime support online, see: <http://www.ibm.com/support/>.
- To provide feedback on documentation for Rational products, see <http://www.ibm.com/software/rational/support/documentation/>.

- For more information about Rational Software technical publications, see <http://www.ibm.com/software/rational/support/documentation/>.
- For more information on training opportunities, see the Rational University Web site: <http://www.ibm.com/services/learning/>.
- For articles, discussion forums, and Web-based training courses on developing software with Rational Suite products, join the Rational Developer Network by selecting **Start > Programs > Rational Suite > Logon to the Rational Developer Network**.

Rational Rose RealTime Integrations with Other Rational Products

Integration	Description	Where it is Documented
Rational Rose RealTime–ClearCase	You can archive Rational Rose RealTime components in ClearCase.	<ul style="list-style-type: none"> ▪ <i>Toolset Guide: Rational Rose RealTime</i> ▪ <i>Guide to Team Development: Rational Rose RealTime</i>
Rational Rose RealTime–UCM	Rose RealTime developers can create baselines of Rose RT projects in UCM and create Rose RealTime projects from baselines.	<ul style="list-style-type: none"> ▪ <i>Toolset Guide: Rational Rose RealTime</i> ▪ <i>Guide to Team Development: Rational Rose RealTime</i>
Rose RealTime–Purify	When linking or running a Rational Rose RealTime model with Purify installed on the system, developers can invoke the Purify executable using the Build > Run with Purify command. While the model runs and when it completes, the integration displays a report in a Purify Tab in Rational Rose RealTime.	<ul style="list-style-type: none"> ▪ <i>Rational Rose RealTime Help</i> ▪ <i>Toolset Guide: Rational Rose RealTime</i> ▪ <i>Installation Guide: Rational Rose RealTime</i>

Integration	Description	Where it is Documented
Rational Rose RealTime–RequisitePro	You can associate RequisitePro requirements and documents with Rational Rose RealTime elements.	<ul style="list-style-type: none"> ▪ <i>Add-ins, Tools, and Wizards Reference: Rational Rose RealTime</i> ▪ <i>Using RequisitePro</i> ▪ <i>Installation Guide: Rational Rose RealTime</i>
Rational Rose RealTime–SoDA	You can create reports that extract information from a Rose RealTime model.	<ul style="list-style-type: none"> ▪ <i>Installation Guide: Rational Rose RealTime</i> ▪ <i>Rational SoDA User's Guide</i> ▪ <i>SoDA Help</i>

Contacting Rational Customer Support

If you have questions about installing, using, or maintaining this product, contact Rational Customer Support.

Your Location	Phone Number	E-mail
North, Central, and South America	1 - 800 IBM-SERV (toll free)	sw_support@us.ibm.com
Europe, Middle East, Africa	1 - 800 IBM-SERV (toll free)	sw_support_emea@nl.ibm.com
Asia Pacific	1 - 800 IBM-SERV (toll free)	sw_support_ap@au.ibm.com

Note: If you encounter any difficulty with the information in the preceding table, contact IBM General Services at 1-800-426-4968.

When you contact Rational Customer Support, be prepared to supply the following information:

- Your IBM customer number (ICN)
- Your name, company name, telephone number, and e-mail address
- Your operating system, version number, and any service packs or patches you have applied
- Product name and release number
- Your Service Request number (PMR#) if you are following up on a previously reported problem

If you send e-mail about a previously reported problem, include the following in the subject field:

[PMR XXXXX YYY ZZZ]

where XXXXX YYY ZZZ is the request number of the issue. For example:

[PMR 12345 678 999] - New data on Rational Rose RealTime install issue

Contents

This chapter is organized as follows:

- *Requirements for Linux* on page 17
- *Requirements for UNIX* on page 20
- *Requirements for Windows NT* on page 18
- *Requirements for Windows 2000* on page 18
- *Requirements for Windows XP Pro* on page 19
- *Referenced Configuration Requirements for the Eclipse and Rational Rose RealTime Integration* on page 21

Requirements for Linux

The minimum supported configuration for running Rational Rose RealTime on Linux is as follows:

- Red Hat 7.3 and 8.0.
- For Linux operation, the minimum workstation is a 450 MHz Pentium III.
- The minimum is 256 MB of RAM. We recommend 512 MB of RAM with approximately three times this amount of swap space.
- Minimum 450 MB of free disk space for the Rational Rose RealTime installation.

For a list of the required Linux patches applicable to your operating system, see the Rational Rose RealTime Web site (<http://www.ibm.com/software/rational/>), or run the **check_rose_reqs** script in the **\$ROSE_HOME/bin** folder.

The GNU libraries included with Red Hat Linux version 7.3 are not current enough for this version of Rational Rose RealTime. To run Rational Rose RealTime on Red Hat Linux version 7.3, you must first install GCC 3.2 or later, because Rational Rose RealTime depends on the run-time libraries included with the new versions of GCC (GNU Compiler Collection).

You can download GCC from <http://gcc.gnu.org/>.

Note: After you complete the GCC installation, ensure that the **lib** directory from the GCC installation is included in **LD_LIBRARY_PATH**. If the libraries are missing, you will receive errors.

Requirements for Windows NT

The minimum supported configuration for running Rational Rose RealTime on Windows NT is as follows:

- Windows NT 4.0, with service pack 6a and SRP
- Minimum Pentium II 150 MHz. We recommend 500 MHz or faster CPU
- Minimum 128 MB of RAM. We recommend 256 MB or more of RAM
- Minimum 325 MB of free disk space for the Rational Rose RealTime installation
- Minimum display 1024 X 768. We recommend 1280 X 1024 or higher
- Postscript printer for printing
- Browser requirement - Internet Explorer 5.5 or 6.0 or Netscape Navigator 4.72-4.78 or 7.0. We recommend Internet Explorer 5.5 or 6.0

Note: AccountLink is not accessible using Netscape 4.x browsers.

For additional information on requirements for installing Rational Suite DevelopmentStudio, see the book *Installation Guide Rational Suite*.

Requirements for Windows 2000

The minimum supported configuration for running Rational Rose RealTime on Windows 2000 is as follows:

- Windows 2000 Professional, with service pack 3 or 4.
- Minimum Pentium II 150 MHz. We recommend 500 MHz or faster CPU
- Minimum 128 MB of RAM. We recommend 256 MB or more of RAM
- Minimum 325 MB of disk space for the Rational Rose RealTime installation

- Minimum display 1024 X 768. We recommend 1280 X 1024 or higher
- Postscript printer for printing
- Browser requirement - Internet Explorer 5.5 or 6.0 or Netscape Navigator 4.72-4.78 or 7.0. We recommend Internet Explorer 5.5 or 6.0

Note: AccountLink is not accessible using Netscape 4.x browsers.

For additional information on requirements for installing Rational Suite DevelopmentStudio, see the book *Installation Guide Rational Suite*.

Requirements for Windows XP Pro

The minimum supported configuration for running Rational Rose RealTime on Windows XP Pro is as follows:

- Windows XP Pro with service pack 1
- Minimum Pentium II 300 MHz. We recommend 500 MHz or faster CPU
- Minimum 128 MB of RAM. We recommend 256 MB or more of RAM
- Minimum 325 MB of free disk space for the Rational Rose RealTime installation
- Minimum display 1024 X 768. We recommend 1280 X 1024 or higher
- Postscript printer for printing
- Browser requirement - Internet Explorer 5.5 or 6.0 or Netscape Navigator 4.72-4.78 or 7.0. We recommend Internet Explorer 5.5 or 6.0

Note: AccountLink is not accessible using Netscape 4.x browsers.

For additional information on requirements for installing Rational Suite DevelopmentStudio, see the book *Installation Guide Rational Suite*.

Requirements for UNIX

The minimum supported configuration for running Rational Rose RealTime on UNIX is as follows:

- Solaris 2.6, Solaris 2.7, Solaris 2.8, or Solaris 2.9
 - For Solaris operation, the minimum workstation is an UltraSparc 10 with 500 MB of RAM. We recommend an UltraSparc 60 with 600 MB of RAM. We recommend the Solaris 2.8 operating system.
 - For a list of the required UNIX patches applicable to your operating system, see the Rational Rose RealTime Web site (<http://www.ibm.com/software/rational/>), or run the **check_rose_reqs** script in the **\$ROSERT_HOME/bin** folder.
- The minimum is 256 MB of RAM. We recommend 512 MB of RAM with approximate three times this amount of swap space.
- Minimum 370 MB of free disk space for the Rational Rose RealTime installation.
- Browser requirement - Netscape Navigator 4.78 - 4.8.

For additional information on requirements for installing Rational Suite DevelopmentStudio, see the book *Installing Rational Suite DevelopmentStudio*.

Referenced Configuration Requirements for the Eclipse and Rational Rose RealTime Integration

The Rational Rose RealTime and Eclipse integration is supported on the following platforms:

- Linux (RedHat 8)
- Windows (2000 and XP Pro)
- UNIX (Sun Solaris 8 and 9)

The Rational Rose RealTime and Eclipse integration requires you to install Eclipse (SDK 2.1.1) and CDT (C/C++ Development Tools 1.2.1 and C/C++ Standard make build 1.2.1). In addition, you must also install software required by Eclipse and CDT, such as a JRE and Cygwin (Windows).

Installing Rational Rose RealTime on Linux

2

Contents

This chapter is organized as follows:

- *Before You Install* on page 23
- *Installation Instructions* on page 25
- *After You Install* on page 29

Before You Install

Before you install Rational Rose RealTime on Linux, refer to the items in Table 1 to direct you to information that can help you perform pre-installation tasks.

Table 1 Linux Pre-installation Tasks

Task	Reference
License your Rational software	See <i>Specifying the Rational License Server</i> on page 29 of the <i>Installation Guide, Rational Rose RealTime</i> , and <i>UNIX Licenses</i> on page 87 of the <i>Installation Guide, Rational Rose RealTime</i> . Note: The AccountLink link has changed to http://www.ibm.com/software/rational/support/licensing
Ensure that your system meets the minimum or recommended system and software requirements	See <i>Requirements for Linux</i> on page 24
System Requirements for Linux	See <i>Requirements for Linux</i> on page 17

Note: If you use Linux with Exceed, the BACKSPACE key does not function. Instead of using the BACKSPACE key, use CTRL + h.

Requirements for Linux

The minimum supported configuration for running Rational Rose RealTime on Linux is as follows:

- Red Hat 7.3 and 8.0.
- For Linux operation, the minimum workstation is a 450 MHz Pentium III.
- The minimum is 256 MB of RAM. We recommend 512 MB of RAM with approximately three times this amount of swap space.
- Minimum 450 MB of free disk space for the Rational Rose RealTime installation.
- For a list of the required Linux patches applicable to your operating system, see the Rational Rose RealTime web site (<http://www.ibm.com/software/rational/support/licensing/>), or run the `check_rose_reqs` script in the `$ROSERT_HOME/bin` folder.

Installing in Secure Environments

Problems may occur when you perform a remote installation of Rational Rose RealTime for Linux in a secure environment (for example, remote access to other machines is through `ssh`) that does not have access to `rsh` or `remsh`. To install Rational Rose RealTime for Linux in this situation, perform a local installation of the software rather than a remote installation. If you experience further problems, contact Rational Customer Support.

Installing Multiple Versions of Rational Rose RealTime for Linux

If you want to install different versions of Rational Rose RealTime for Linux on the same file server, we recommend that you install them in different Rational directories (referred to as `<rational_dir>`). If you install them in the same Rational directory, you cannot uninstall a specific version because the uninstall script removes all versions in the same Rational directory.

Stopping and Restarting an Installation

To stop an installation, type `q`. If you type `q`, most of your input is saved to a user defaults file in `<rational_dir>/config/defaults`. The file name is in the following format:

`rs_install.release_name.user_name`

The defaults file contains general purpose defaults that relate to the *username* and the license server that you configure. It also records the product-specific information for the installation of this specific product and version.

Note: If you type **q!**, only some of your entries are saved to the user defaults file.

To restart the installation, run **rs_install** again. Many of your entries appear as the default value. Press the **ENTER** key to continue with the installation.

Installation Instructions

Unless otherwise specified, your system administrator will perform these steps.

Note: For environments where there is more than one user of Rational Rose RealTime for Linux, we strongly recommend that you install the Rational Rose RealTime files on a centralized file server.

During the installation process, the default values are prefixed with the following notation:

-->

To accept the default value and continue with the installation, press **ENTER**.

To Install Rational Rose RealTime on Linux:

Note: The directory and file names indicated in the following steps are for example purposes only.

- 1 Log on to the install client. This can be any Linux computer that:
 - Gives you access to a CD-ROM drive
 - Mounts the file system into which you will load the Rational Rose RealTime for Linux release
 - Runs the operating system specified on the *Rational Rose RealTime for Linux* CD (Red Hat 7.3, or 8.0)
- 2 Place the *Rational Rose RealTime for Linux* CD in the CD-ROM drive.

Note: If the CD-ROM drive is not mounted, mount the CD-ROM drive.

As the root user, create a directory (if one does not exist already) to be the mount point for the CD-ROM drive. The following examples for each platform use the directory **/cdrom**. Ensure that you know the device name of the CD-ROM drive. If you do not know the device name, consult your system administrator.

The mounting commands for different operating systems are as follows:

- **Linux with Volume Management**

Linux with volume management mounts to the **/cdrom** directory automatically when you load the CD-ROM drive. You have volume management if the **vol** daemon is running on the system.

- **Linux without Volume Management**

```
# mkdir /cdrom
# mount -t iso9660 -r /dev/cdrom /mnt/cdrom
```

- 3 From a shell window, change the directory to the root level of the mounted CD-ROM device. For example: **cd /cdrom**, and press **ENTER**.

- 4 To run the setup script, type the following command:

rs_install

The **rs_install** command is a complete installer that includes licensing setup, license checking, product installation, and product setup. Rational recommends that you follow the menus and prompts and allow **rs_install** to guide you through the installation process.

Note: You can invoke **rs_install** with a number of options. For example, you can use the **-no_log (-nl)** option to stop **rs_install** from creating a log file. To see a listing of all available options, type **rs_install -help**.

- 5 After the **Using RS Install** script displays, press **ENTER** to continue.

In the **Enter Install Location** script, the installation process searches for Rational directories.

- 6 Press **ENTER** to continue.

- 7 Specify the directory in which to install Rational Rose RealTime for Linux, according to your type of installation:

Note: An arrow (- - >) opposite a number indicates the default used for this installation. Press **ENTER** to select the default.

- **First time installation** - If you are installing Rational Rose RealTime for Linux for the first time, you are automatically prompted to specify a directory for the installation, and then press **ENTER**.
- **Existing single installation** - If the installation process detects an existing Rational Rose RealTime directory, you can press **ENTER** to select that directory, or type **0** (zero) to specify a new directory.
- **Multiple existing installations** - If the Rational Rose RealTime installation detects multiple Rational Rose RealTime directories, type **0** to specify a new directory, or type a value associated with a listed directory, and then press **ENTER**.

If you specify a new directory, **rs_install** copies the Rational files to this location. The directory name must be specified as an absolute path name. The directory must be visible on all computers from which you want to run Rational Rose RealTime, and must be writable by the installer's user name.

Next, the license agreements appear and you are prompted to accept or reject the license agreements. You must accept both license agreements to proceed with the installation.

- 8 If you agree with the terms of the Rational Rose RealTime License Agreement, type **Y** and press **ENTER**.
- 9 If you agree with the terms of the Third Party License Agreement, type **Y** and press **ENTER**.

Note: If you do not agree with the terms of the license agreements, the installation stops. You should return all software and documentation to IBM Rational Software.

- 10 On the **Product and License Configuration** menu, type the number associated with **Rational Rose RealTime for Linux**, and then press **ENTER**.

Note: If the installation process detects any existing license configurations, you can specify their use and continue with the installation prompts. Otherwise, you must obtain and specify a valid license and continue with Step 11. For additional information on licenses, see *Specifying the Rational License Server* on page 29 of the *Installation Guide, Rational Rose RealTime*, and *UNIX Licenses* on page 87 of the *Installation Guide, Rational Rose RealTime*.

- 11 On the **Rational Rose RealTime - Licensing Options** menu, select a licensing option.

Option	Description
1	Use an existing Rational license (FLEXlm) file or a server that is already configured.
2	Set up a permanent or term license(s). <ul style="list-style-type: none">▪ Request Node-Locked or floating keys through AccountLink (http://www.ibm.com/software/rational/support/licensing/).▪ After you request Node-Locked key(s) from AccountLink, you will receive an e-mail from Rational that contains an attachment (a .upd file). You must save this file to a secure location on your workstation.
3	Set up a temporary license file.

Depending on the licensing option you select, answer the questions and follow the directions.

- 12 After licensing, on the **Rational Rose RealTime - Product Customization** menu, verify that Rational Rose RealTime for Linux will be installed, and that you have enough space to install it.
- 13 Press **f** to continue.
- 14 Select an installation option. **Typical** installs all components, and **Custom** allows you to specify only those components that you want installed.
- 15 In **Rational Rose RealTime - Enter Install Mode** menu, indicate how you want **rs_install** to deal with components that are already installed.
- 16 Press **ENTER** to continue.
- rs_install** installs Rational Rose RealTime for Linux. If there is not sufficient disk space, the installation process stops.
- 17 After the installation process completes, press **ENTER** to continue.

After You Install

After you install, complete these steps:

- *Installing GNU 3.2 on page 29*
- *Source the Setup Script on page 29*
- *Unmount the CD-ROM Drive on page 30*
- *ClearCase Workstation Configuration on page 30*
- *Configure the ClearCase Repository on page 31*
- *Set the Connexis Variable on page 32*
- *Verify the Connexis Installation on page 32*

Installing GNU 3.2

The GNU libraries included with Red Hat Linux version 7.3 are not current enough for this version of Rational Rose RealTime. To run Rational Rose RealTime on Red Hat Linux version 7.3, you must first install GCC 3.2 or later, because Rational Rose RealTime depends on the run-time libraries included with the new versions of GCC (GNU Compiler Collection).

You can download GCC from <http://gcc.gnu.org/>.

Note: After you complete the GCC installation, ensure that the **lib** directory from the GCC installation is included in **LD_LIBRARY_PATH**. If the libraries are missing, you will receive errors similar to the following:

```
RoseRT: error while loading shared libraries: libstdc++.so.5:
cannot open shared object file: No such file or directory.
```

Source the Setup Script

After you install Rational Rose RealTime for Linux, you should **source** your **rs_setup** script to automatically set your environment variables.

- For the Rational Rose RealTime point product, type the following:

```
source <rational_dir>/rosert_setup.csh
```

or

```
. <rational_dir>/rosert_setup.sh
```

Unmount the CD-ROM Drive

For CD-ROM installations, unmount the CD-ROM drive with the following command:

```
umount /Name
```

where *Name* is the name of the device or resource.

Note: You cannot eject the CD if you are at the directory **/cdrom** or **/cdrom/cdrom0**. If you receive a "Device busy" error, change your directory location to a location other than the CD-ROM and repeat the preceding commands.

ClearCase Workstation Configuration

The following setup must take place on all workstations that will be accessing a VOB or view. For Linux, this includes all machines that are view servers.

These steps will also need to run on all machines that act as view servers for the ClearCase views used by Rational Rose RealTime. If you use ClearCase MultiSite, you will need to do this at all the sites where the VOBs containing the Rose RealTime elements are replicated.

You can determine which machines are view servers by typing:

cleartool lsview

in a command window. The second item on each output line indicates the machine name where the view server is running. For example, if you see the following line in the output of the **lsview** command:

```
myview \\mymachine\vws\myview.vws
```

then "*mymachine*" is the name of the computer on which the view server for **myview** exists.

For further details, see your ClearCase administrator.

Command-line Access to the Source Control Tool

For any user who wants to use the Rational Rose RealTime integration with ClearCase, **cleartool** must be accessible from the command prompt.

Element Type Setup: type Manager

The following steps are required for making ClearCase clients aware of the new element type.

Linux

Use the `$ROSERT_HOME/bin/$ROSERT_HOST/cc/mi_typeman` script to install the type manager in each ClearCase installation. To configure the extensions and tool mappings, the user who runs the script must have write access to the following directories in the ClearCase installation:

```
<CC_HOME>/lib/mgrs  
<CC_HOME>/config/ui/icons  
<CC_HOME>/config/ui/bitmaps  
<CC_HOME>/config/magic
```

Use the following command-line to configure the proper file extensions and tool invocations:

```
<ROSERT_HOME>/bin/<ROSERT_HOST>/cc/mi_typeman.sh install-server
```

Configure the ClearCase Repository

Each VOB must be configured to allow files of the new element type to be created. Follow the steps that apply to your platform below for each VOB that will be storing Rational Rose RealTime files.

Linux

Use the `$ROSERT_HOME/bin/$ROSERT_HOST/cc/mi_typeman` script to register the `rosert_unit` element type in each VOB using the following syntax:

```
<ROSERT_HOME>/bin/<ROSERT_HOST>/cc/mi_typeman.sh install-eltype -vob  
<vob_path>
```

Test the Type Manager

To determine if the `rosert_unit` element type has been successfully registered in the VOB, perform the following command from a command prompt after changing to a directory contained in the VOB:

```
cleartool lstype -long eltype:rosert_unit
```

A listing of the type details will verify that it is correctly registered.

Set the Connexis Variable

After you install Rational Rose RealTime, you must set the environment variable for `CONNEXIS_HOME` to the appropriate location, such as:

```
setenv CONNEXIS_HOME $ROSET_HOME/Connexis
```

Note: Set this environment variable after `$ROSET_HOME` is created (either by `setenv ROSET_HOME` or in a `rs_install` setup), and then type:

```
source <rational_dir>/rosert_setup.csh
```

or

```
. <rational_dir>/rosert_setup.sh
```

Verify the Connexis Installation

To increase efficiency and eliminate improper installations, or misconfigurations, you are strongly encouraged to verify your installation.

Verifying your Installation using BasicTest

You can verify your Connexis installation by using the BasicTest model provided with Connexis in `$ROSET_HOME/Connexis/C++/Examples`. This model uses the CDM transport.

Host Configuration Installation Verification

The following instructions are for the Linux host platform.

To verify your host configuration installation:

Note: Use the information in Table 2 and Table 3, when you complete the following steps:

- 1 Start Rational Rose RealTime.
- 2 Load the BasicTest model from `$ROSET_HOME/Connexis/C++/examples`.
- 3 From the **Component View**, expand the component package that corresponds to your host platform.
- 4 Select the client component and from its item menu, click **Build > Rebuild All** to recompile it.
- 5 Select the server component and from its item menu, click **Build > Rebuild All** to recompile.

- 6 In the **Deployment View** package, expand the processor that corresponds to your host platform.

The client will use port 9100 and the server will use port 9900. If these ports are being used by other another application on your workstation, you will need to change them.

- 7 Open the server component instance's specification sheet and change the 9900 in the **-CNXep** startup parameter to an available port number.
- 8 Open the client **Component Instance Specification** dialog box and change 9900 specified in the **-s** argument to the server's port number.
- 9 Change the 9100 in the **-CNXep startup** parameter to an available port number.
- 10 Save your changes.
- 11 Select the server component instance and click **Run**.
- 12 On the **RunTime View** tab of the instance, click **Start** to execute the server.
- 13 On the **Model View** tab, select the client component instance and click **Run** from its item menu. On the **Runtime View** tab of the instance, click **Start** to execute the client.
- 14 Verify that the output for client and server looks similar to the output shown in sections *BasicTest Server Output* on page 34 and *BasicTest Client Output* on page 35.

Table 2 Components for Referenced Configurations

	Component Package	Client Component	Server Component
Linux	REDHAT73-X86-gnu-3-2	basicTestClient_43	basicTestServer_43
Linux	REDHAT80-X86-gnu-3-2	basicTestClient_44	basicTestServer_44

Table 3 Component Instances for Referenced Configurations

	Component Package	Client Component	Client Component Instance	Server Component Instance
Linux	Red Hat 7.3	MyRedHat73Workstation	basicTestClient_43 Instance	basicTestServer_43 Instance
Linux	Red Hat 8.0	MyRedHat80Workstation	basicTestClient_44 Instance	basicTestServer_44 Instance

BasicTest Server Output

```
Rational Rose RealTime C++ Target Run Time System
Release 6.50.B.82 (+c)
Copyright (c) 1993-2004 Rational Software
rosert: observability listening at tcp port 30399
*****
*           Please note: STDIN is turned off.           *
*   To use the command line, telnet to the above mentioned port.   *
*   The _output_ of any command will be displayed in _this_ window. *
*****
Rational Software Corp. Connexis(tm) - Distributed Connection
Service (dcs)
Release 6.50.B.82
Copyright (c) 1999-2004 Rational Software Corporation

dcs: CRM Transport : enabled
dcs: CDM Transport : enabled
dcs: CRM listening at [crm://192.139.251.167:2005]
dcs: CDM listening at [cdm://192.139.251.167:9900]
dcs: target agent enabled
dcs: locator service not available
dcs: metric service enabled

BasicTest-Server-started:

Server : Received simple greeting message... sending it back
Server : test cycle completed, received rtunbound !
Server : Received simple greeting message... sending it back
Server : test cycle completed, received rtunbound !
```

Note: The preceding output results represents a partial listing of the BasicTest Server Output.

BasicTest Client Output

```
Rational Rose RealTime C++ Target Run Time System
Release 6.50.B.82 (+c)
Copyright (c) 1993-2004 Rational Software
rosert: observability listening at tcp port 30380
*****
*           Please note: STDIN is turned off.           *
*   To use the command line, telnet to the above mentioned port.   *
* The _output_ of any command will be displayed in _this_ window. *
*****
Rational Software Corp. Connexis(tm) - Distributed Connection
Service (dcs)
Release 6.50.B.82
Copyright (c) 1999-2004 Rational Software Corporation

BasicTest-Client-started:

dcs: CRM Transport : enabled
dcs: CDM Transport : enabled
dcs: CRM listening at [crm://192.139.251.167:2010]
dcs: CDM listening at [cdm://192.139.251.167:9100]
dcs: target agent enabled
dcs: locator service not available
dcs: metric service enabled

Client : sending a greeting message...

->Client: received message:
RTString"Hello, Welcome to the Connexis world!"

Client : unbound received
```

```
Client : reregistering SAP

Client : sending a greeting message...

->Client: received message:
RTString"Hello, Welcome to the Connexis world!"

Client : unbound received
```

Note: The preceding output results represents a partial listing of the BasicTest Client Output.

Starting Rational Rose RealTime on Linux

To start Rational Rose RealTime on Linux configurations, run the command displayed at the end of the **rs_install** process. For example:

/myInstall/Rose RealTime/bin/RoseRT

Note: The installation process creates **rosert_setup.csh** or **rosert_setup.sh**.

You can source the setup file to help you start the programs from this installation. If you install Rational Rose RealTime for other users, they should add one of these commands to their login environment:

- Users of **csh**, **tcsh** and other **csh**-compatible shells must add the following command:
source /myInstall/Rose RealTime/rosert_setup.csh
- Users of **sh**, **ksh**, **bash** and other bourne-compatible shells must add the following command:
./myInstall/Rose RealTime/rosert_setup.sh

Contents

This chapter is organized as follows:

- *Startup Issues* on page 37
- *Uninstall Issues* on page 38
- *Windows-Specific Issues* on page 39
- *UNIX-Specific Issues* on page 43
- *Linux-Specific Issues* on page 47
- *Cross-Platform Issues* on page 49
- *Online Help Issues* on page 60
- *Documentation Updates* on page 62
- *Using Type Descriptor Functions* on page 67

Startup Issues

If you encounter startup issues for Rational Rose RealTime, review the following topics:

- *Starting Rational Rose RealTime When an Instance is Currently Running* on page 37
- *Toolset Freezes on Startup* on page 38
- *Virus Scanning Applications Affect Startup and Shutdown* on page 38

Starting Rational Rose RealTime When an Instance is Currently Running

If you encounter problems starting Rational Rose RealTime, look at the Task Bar and use the Task Manager to see if other copies of Rational Rose RealTime are running. Close any other copies to allow the new copy of the tool to start properly.

More than one copy of Rational Rose RealTime can run at the same time; however, if you have start-up problems, find and terminate any runaway processes.

Note: On UNIX, if a Rational Rose RealTime version 2003.06.12 (version 6.5) closes unexpectedly, prior to starting an earlier version of Rational Rose RealTime (2002.05.xx), we strongly recommend that you delete the Windows directory in your \$HOME directory. If you do not delete the Windows directory, you will not be able to print.

Toolset Freezes on Startup

Under certain configurations, when starting Rational Rose RealTime on Solaris and displaying to a Hummingbird Exceed X-server, the "Welcome to ..." window appears momentarily, and then disappears. After this, the main application GUI is locked out and freezes.

To work around this problem:

- 1 Stop the Rational Rose RealTime process.
- 2 Edit the **RoseRT.ini** file in the **~/.registry.2003.06.00** directory and change **ShowStartupDialog=Yes** to **ShowStartupDialog=No**.
- 3 Run **RoseRT -cleanup**.
- 4 Run **RoseRT**.

Virus Scanning Applications Affect Startup and Shutdown

On startup or shutdown, virus scanning applications, such as McAfee VirusScan and Trend, will scan the RoseRT.ini file many times, causing a very slow startup or shutdown. To enable a much faster startup and shutdown, exclude the file RoseRT.ini from your virus scanning software.

Uninstall Issues

Occasionally, files remain after you uninstall Rational Rose RealTime. For example, if a model was saved in one of the Rational Rose RealTime subdirectories, the subdirectory and its parent directories are not removed. To return your system to a clean state, you must manually remove these directories.

Note: After you uninstall, your license files will remain in your **/Common** directory, for example, **C:/Program Files/Rational/Common**. We strongly recommend that you copy the license files to another secure location before deleting the **/Common** directory.

Windows-Specific Issues

For information on Windows-specific issues, review the following topics:

- *Service Pack Requirement Update for Windows* on page 39
- *Using Hummingbird Exceed 7.1 on a Computer Running Windows Applications and Rational Rose RealTime* on page 39
- *Using Rational Rose RealTime on Windows XP Pro Configurations* on page 39
- *Using Rational Rose RealTime without Appropriate Privileges* on page 40
- *Building Dependencies on Case-Insensitive File Systems* on page 40
- *File Association for Compiled Scripts* on page 41
- *Windows CE GetSystemTime() Function Does Not Return Milliseconds* on page 41
- *Symbolic Links with TargetRTS* on page 42
- *Spaces in Directory Names* on page 42

Service Pack Requirement Update for Windows

Rational Rose RealTime now supports Windows 2000 service pack SP4; however, Windows 2000 service pack SP2 is no longer supported.

Using Hummingbird Exceed 7.1 on a Computer Running Windows Applications and Rational Rose RealTime

If you use a computer running Microsoft Windows XP Pro, Microsoft Office XP Service Pack 1, Microsoft Word (selected as editor for Microsoft Outlook), while Microsoft Outlook is open, if you start Rational Rose RealTime from a Hummingbird Exceed (version 7.1) window, shortly after launching Rational Rose RealTime, Microsoft Outlook, Microsoft Word, and **spoolsv.exe**, CPU activity increases to near 100%. These applications remain at that level for approximately one minute after Rational Rose RealTime closes, then your Rational Rose RealTime session will begin to function normally after approximately one minute. However, Microsoft Outlook and any open Microsoft Word documents become unusable.

Using Rational Rose RealTime on Windows XP Pro Configurations

Before you install Rational Rose RealTime on any Windows XP Pro configuration, ensure that you have service pack 1 installed. The *Installation Guide, Rational Rose RealTime* is incorrect because it indicates that you can install Rational Rose RealTime on Windows XP Pro and Windows XP Pro with service pack 1.

Using Rational Rose RealTime without Appropriate Privileges

To run Rational Rose RealTime on Windows 2000, you must log on with local Administrator or Power User privileges. You will encounter the following types of problems running Rational Rose RealTime 2003.06.00 on Windows 2000 if you log on as a user without local Administrator or Power User privileges:

- When opening Rational Rose RealTime, you may see the error message, "**Failed to Update the System Registry. Try using REGEDIT.**"
- The configuration of the Add-in manager cannot be restored.
- The two menu items **Add Class Dependencies** and **Component Wizard** are missing from the **Build** menu.
- When creating a component, you are not able to define the **Environment** in the **Component Specification** for the **C++ TargetRTS**.
- When selecting **Rebuild** from the **Build** menu, or clicking the **Build** tool from the Toolbar, there is no activity.
- After selecting **Run** from the **Build** menu, and then clicking **Yes** to select **Build** the component, you will receive the error message "**Operation not allowed**".
- When clicking **Help > About**, there is no **Version** or **Company** information.
- You will not be allowed to set a top capsule in the **Component Specification** dialog.

Building Dependencies on Case-Insensitive File Systems

During a build, Rational Rose RealTime detects and records build dependencies for comparison during subsequent builds. This is done to facilitate build-avoidance by only regenerating or recompiling targets when a build dependency changes. These build dependencies preserve the case of the file names involved, including situations when the underlying file system (for example, NTFS) is case-insensitive. This may cause problems when using names that are distinct within the toolset, and distinct on case-sensitive file systems, but indistinct on case-insensitive file systems. In most cases, the toolset or code-generator will identify and avoid or warn against potential case-insensitive name collisions.

However, some case-insensitive file name collisions cannot be detected. For example, if a component is renamed from "myComponent" to "MyComponent", a build may incorrectly reuse all previous build results, since the underlying build dependencies will be indistinct according to the file system.

Note: We recommend that you use case-sensitive file-systems wherever possible, and avoid case-insensitive name collisions when you create or rename classes, components, or controllable units.

File Association for Compiled Scripts

On Windows NT, Rational Rose RealTime does not install a file association for compiled scripts (.ebx). This means that compiled script file cannot automatically run by double-clicking the file in Windows Explorer.

Windows CE GetSystemTime() Function Does Not Return Milliseconds

Some Windows CE targets do not return milliseconds in the **GetSystemTime()** call. This causes the resolution of the clock, which is used for timers, to be 1 second. If milliseconds timers are required, you must modify the **getclock.cc** function to use **GetTickCount()**.

For example, in the **getlock.cc** file in the directory **\$RTS_HOME/src/target/WINCE/RTTimespec/getclock.cc**, you will change the **GetSystemTime()** call and replace it with **GetTickCount()** as follows:

```
#include <RTDiag.h>

#include <RTTimespec.h>

#include <windows.h>

void RTTimespec::getclock( RTTimespec & ts )
{
    DWORD ticks = GetTickCount();

    ts.tv_sec = ( (long)ticks / 1000L );

    ts.tv_nsec = 1000000L * ( (long)ticks % 1000L );
}
```

Note: **GetTickCount()** is the system uptime and will wrap around after 49.7 days. For systems that are up for longer, a combination of **GetSystemTime()** and **GetTickCount()** could be used to gain the desired length and resolution of clock time. This implementation is system dependent.

Symbolic Links with TargetRTS

When using LynxOS 3.1.0, do not install Visual Lynx 3.1.0 for Windows NT on a network (NFS) disk. It should only be installed on a local NTFS drive; otherwise, symbolic links to some **include** directories will not work properly. Compilation errors will occur if you re-compile the TargetRTS.

If you installed Visual Lynx 3.1.0 on a network disk, and if you see compilation errors stating that the include directories **netinet/in.h** or **net/if.h** are not found, locate the **net** and **netinet** entries in the following directory:

\$LYNX_HOME/usr/lynx/3.1.0/ppc/usr/include

If these entries are text files containing the following text: **!<symlink>bsd**, rename these files, create new symbolic links called **net** and **netinet**, and have both of them point to the directory called **bsd**.

Spaces in Directory Names

To enable the use of cross-compilers that do not allow spaces in the path names, use the **subst** command and map a drive to the value of **%ROSSERT_HOME%** after the installation. For example, if you want to use the K: drive, and your Rational Rose RealTime installation directory is:

ROSSERT_HOME=C:\Program Files\Rational\Rose RealTime

you must map this directory to the drive by running the following commands from a console window:

subst K: "%ROSSERT_HOME%"

set ROSSERT_HOME=K:

Spaces in directory names can cause problems with the following "operating system.compiler library set.development platform" systems:

- OSE411T.ppc603-Diab-4.3f.NT4
- VRTX4T.ppc603-Microtec-1.4.NT40
- TORNADO2T.ppc630-GreenVX-1.8.9.NT40
- TORNADO2T.ppc630-GreenVX-2.0.NT40
- TORNADO2T.m68040-cygnus-2.7.2-960126.NT40
- TORNADO2T.ppc-cygnus-2.7.2-960126.NT40
- TORNADO2T.ppc860-cygnus-2.7.2-960126.NT40

For Tornado configurations, you should also use the substituted drive to point to the load script directory when running Target Observability.

On the **Detail** tab of the **Processor Specification** dialog, ensure that the **Load Script** path does not contain any spaces. If spaces are present, when you attempt to load the executable in **Basic** or **Debugger-Tornado2** modes, you will receive the error message "**Unable to Execute**". This does not occur when loading the executable in **Manual** mode.

Note: Spaces in directory names can also cause problems with **ClearMake**.

UNIX-Specific Issues

For information on UNIX-specific issues, review the following topics:

- *Printing on UNIX* on page 43
- *Browser Requirements for UNIX* on page 43
- *Troubleshooting when the Toolset Freezes on UNIX* on page 44
- *Rational Rose RealTime Crashes when You Debug Using Tornado 2.2 on UNIX* on page 45
- *Specifying a Location or File Name Containing Spaces (UNIX)* on page 45
- *Starting vi as an External Editor from Rational Rose RealTime* on page 46
- *Unable to Open Some Links in the Online Help* on page 46
- *Exceptions Occur When You Use Configuration Management in Rational Rose RealTime* on page 46
- *Case Sensitivity within Paths* on page 46
- *Window Order Policy* on page 46
- *Non-GUI-based External Editors* on page 47
- *Setting the Stack Space Limit* on page 47
- *Rational Rose RealTime Fails to Build a Component on Solaris 2.6* on page 47

Printing on UNIX

If you click the **Print** icon in the Toolbar, the **Print Topics** dialog appears with two options: **Print the selected topic** and **Print the selected heading and all subtopics**. Because the **Print the selected heading and all subtopics** option is not available, it is not possible to print all of the subtopics on a Solaris configuration.

Browser Requirements for UNIX

The *Installation Guide, Rational Rose RealTime* indicates that the browser requirement for UNIX is Netscape 7.0. This is not correct. The correct browser requirements for UNIX are Netscape 4.78 and Netscape 4.8.

Troubleshooting when the Toolset Freezes on UNIX

In the unlikely event that the Rational Rose RealTime toolset freezes on UNIX, the toolset does not respond to user input, or the window does not refresh.

To recover from a toolset freeze:

- 1 Run `/usr/proc/bin/pstack [Rose RealTime pid] > [outfile]`.
- 2 Run `/bin/truss -o [output file] -p [Rose RealTime pid]` for 15 seconds.
- 3 Stop the Rational Rose RealTime process by running:
kill -SEGV [Rose RealTime pid]

You are also prompted to do the following:

- Provide information on how to reproduce this problem.
- Run a resource cleanup utility on your session.
- Indicate whether you want to be contacted by Rational Customer Service.

If you answer **Yes** to "**Would you like to be contacted by Rational Customer Service**", a Rational Customer Support engineer will contact you shortly. You are also prompted to provide the information you gathered in steps 1 through 6.

- 4 Click **Help > About Rational Rose RealTime** and obtain the Rational Rose RealTime build number, (for example, 6.5.444.0).
- 5 Attempt to reproduce the problem:
 - If you can reproduce the steps that caused the toolset freeze, note these steps.
 - If you are unable to reproduce the steps that caused the toolset freeze, any information you can provide on the activities you were doing at the time of the toolset freeze will be helpful to the investigation.
- 6 If you use ClearCase, and were in a ClearCase view at the time of the toolset freeze, record the version of ClearCase that you are using.

Forward the preceding information to Rational Customer Service (for contact information, see *Contacting Rational Customer Support* on page xiv). Ensure that you include "Rose RealTime" in the subject of your email.

Rational Rose RealTime Crashes when You Debug Using Tornado 2.2 on UNIX

If you are using debugger integration with Tornado 2.2, Rational Rose RealTime will crash when it loads a component instance. To resolve this problem, go to the WindRiver Web site (www.windriver.com) and download the latest patch for Tornado 2.2 that includes an update to the file **libwtxapi.so**.

Refresh Problems with Exceed

Occasionally, the screen does not refresh completely when you run the UNIX version of Rational Rose RealTime on a PC using Hummingbird Exceed.

To change your Exceed settings:

- 1 Start Xconfig.
- 2 Open the **Performance** dialog.
- 3 Use these settings:
 - **Save Unders = No**
 - **Maximum Backing Store = When Mapped**
 - **Default Backing Store = None**
 - **Minimum Backing Store = None**

Specifying a Location or File Name Containing Spaces (UNIX)

To properly process a location or file name that contains one or more spaces, the command line must be properly quoted. You need two levels of quotation marks: the first set of quotation marks (') encloses the second ("). Specifying this type of quotation marks ensures that the RoseRT script does not interpret the space character, and that it passes the file name (including space characters) as a single argument to the RoseRT executable. For example, given the following path and file name:

Test with Spaces/Model with spaces.rtmdl

you would invoke the toolset using the following command:

RoseRT "'Test with Spaces/Model with spaces.rtmdl'"

The file name parameter is quoted as follows:

<open single quote><open double quote> path/filename<close double quote><close single quote>

Note: You must use the quotation marks as described. If you use double quotation marks to wrap the single quotation marks, the string for the file name is processed as a command and it will produce errors.

Starting vi as an External Editor from Rational Rose RealTime

While using Rational Rose RealTime on Solaris version 2.9, if you perform an action that starts the vi editor (the vi editor starts in a separate xterm window), clicking on the "X" on this window displays the following message:

This will terminate your X client session.

If you click **OK**, the vi xterm window does not close.

Unable to Open Some Links in the Online Help

Some links to multimedia content do not work on UNIX. Where possible, an alternative method for opening certain files is included in the online help.

Exceptions Occur When You Use Configuration Management in Rational Rose RealTime

If an exception occurs when you are working with models under source control on UNIX, your temporary file, **/var/tmp**, is likely full.

Case Sensitivity within Paths

The UNIX temporary directory name is translated to lowercase. If you set the environment variable **TEMP**, ensure that the path name is all lowercase; otherwise, the directory will not be found. This will cause problems when you open the online Help.

Window Order Policy

When you use the CDE window manager, to ensure that the proper Secondary and Transient window policy is in effect, in **.Xdefaults**, set the following environment variable:

Dtwm*secondariesOnTop: True

Setting this variable to True ensures that an opened secondary window in Rational Rose RealTime for UNIX (such as an external editing window) does not appear behind the main primary window. Since the secondary window is the active window, you may be unable to regain focus of this secondary window.

When you use CDE as your XWindow manager, the **Allow Primary Windows on Top** and **Raise Windows When Made Active** options are enabled by default. These options should be disabled when setting the **Dtwm*secondariesOnTop** option to **True**.

Non-GUI-based External Editors

On UNIX, the toolset will freeze if you specified **/bin/vi** as the external editor and you attempt to start the external editor from the **Code edit** window. If you use a non-GUI-based external editor, to ensure that the editor has a terminal (**tty**) to display to, specify **xterm -e /bin/vi**.

Setting the Stack Space Limit

Some operations with large models might require a value larger than 32 MB to be set manually. If your UNIX administrator set a hard limit on the stack size, you can set a higher limit by using the **limit** command in **csh**, or the **ulimit** command in **sh** or **ksh**.

Rational Rose RealTime Fails to Build a Component on Solaris 2.6

Rational Rose RealTime displays an error when you attempt to build a component on Solaris 2.6 when you use the default **make** command.

Linux-Specific Issues

For information on Linux-specific issues, review the following topics:

- *Using the C++ Analyzer on Linux* on page 48
- *Displaying the Version Tree on Linux* on page 48
- *Using Connexis Viewer on Linux* on page 48
- *Updating GNU Libraries on Linux 7.3* on page 48
- *Using Shortcut Keys to Cycle Through Open Specification Dialog Boxes on Linux* on page 49
- *Using Web Publisher on Linux* on page 49
- *Viewing the Online Help on Linux* on page 49
- *Using Context-Sensitive Help on the Preferences Dialog in Eclipse on Linux* on page 49

Using the C++ Analyzer on Linux

Although Rational Rose RealTime for Linux supports Red Hat versions 7.3 and 8.0, the C++ Analyzer is not available for these versions.

Rational Rose RealTime uses the C++ Analyzer for reverse engineering and the Code Import feature. Consequently, these features will not function in this version of Rational Rose RealTime for Linux. Alternatively, you can launch the C++ Analyzer from a UNIX or Windows version of Rational Rose or Rational Rose RealTime, and then import the code from the Linux version of Rational Rose RealTime.

Displaying the Version Tree on Linux

Rational Rose RealTime uses the ClearCase version tree feature to display a version tree from the toolset. Currently, this feature will not display the version tree because ClearCase 2002.05.00 on Linux Red Hat 8.0 does not support Unicode.

To display the version tree in Rational Rose RealTime, use a non-Unicode locale such as C, from a Linux command window, and then type the following:

```
setenv LC_ALL C
```

Using Connexis Viewer on Linux

When using Connexis Viewer with RedHat Linux versions 7.3 and 8.0, the data is not legible. Alternatively, you can launch Connexis Viewer using Exceed as your Xserver, or run Connexis Viewer on Linux and use a Solaris computer as your Xserver.

Updating GNU Libraries on Linux 7.3

The GNU libraries included with Red Hat Linux version 7.3 are not current enough for this version of Rational Rose RealTime. To run Rational Rose RealTime on Red Hat Linux version 7.3, you must first install GCC 3.2 or later, because Rational Rose RealTime depends on the run-time libraries included with the new versions of GCC (GNU Compiler Collection).

You can download GCC from <http://gcc.gnu.org/>.

Note: After you complete the GCC installation, ensure that the **lib** directory from the GCC installation is included in **LD_LIBRARY_PATH**. If the libraries are missing, you will receive errors.

Using Shortcut Keys to Cycle Through Open Specification Dialog Boxes on Linux

When using Red Hat Linux 7.3, the SHIFT+ALT+RIGHT ARROW (next **Specification** dialog box) and SHIFT+ALT+LEFT ARROW (previous **Specification** dialog box) shortcut keys do not cycle between open Specification dialogs. Instead, specify these shortcuts to cycle through Linux workspaces.

Using Web Publisher on Linux

When using Web Publisher with RedHat Linux versions 7.3 and 8.0, if the publishing process fails, change the display to use a lower color depth. For example, if the current color depth is set to 24 bits, change the value to 16 bits.

When using Web Publisher with Exceed, Rational Rose RealTime crashes.

Viewing the Online Help on Linux

When Viewing the online Help, you might encounter only Windows and UNIX buttons and links. For Linux, you can select the UNIX buttons and links.

Using Context-Sensitive Help on the Preferences Dialog in Eclipse on Linux

If you press F1 on any of the options on the **Preferences** dialog in Eclipse, the context-sensitive help does not display. Use the **Index** tab in the online help to search for help on these options.

Cross-Platform Issues

For information on cross-platform issues, review the following topics:

- *Synchronizing Code between Rational Rose RealTime and Eclipse* on page 50
- *Rational Rose RealTime and Eclipse Integration Does Not Support Java* on page 50
- *Using Rational Rose RealTime with a Node Locked License Does Not Warn User About Expiration* on page 50
- *Using Pathmaps in Rational Rose RealTime* on page 51
- *Limitations on the Number of Open Windows* on page 51
- *Limitations in the Specification History List* on page 52
- *Loading a Workspace might Cause Default Color Settings to Change Permanently* on page 52
- *Do Not Use \$& When You Define a PathMap* on page 52
- *Use Caution When Modifying OutPutDirectory* on page 52

- *Using Rational Rose RealTime on Non-English Installations Causes an Unreadable Font When Viewing Generated Code* on page 53
- *Unable to use Parameters with the cm_getcaps Script* on page 53
- *Problems Compiling Java Models* on page 53
- *Problems Connecting to a Target* on page 53
- *Using Sequence Diagrams* on page 53
- *Using the Debugger-xxgdb Tool and Running your Component Instance* on page 53
- *Using C and C++ Add-ins* on page 54
- *Code Generator Runs Out of Memory When Generating Very Large Models* on page 54
- *Using the Get and Set Methods in the Attribute and Operation Tools* on page 54
- *Web Publisher Applet Does Not Load Properly* on page 55
- *Using the Frameworks Dialog* on page 55
- *Scoping Descriptors for Nested Classes* on page 55
- *No Codesync Support for Java* on page 56
- *Using the GetSelected Functions* on page 56
- *Using the Find Command Might Return Too Many Results* on page 58
- *No Support for Automatic Unwired Port Registration for Java* on page 58
- *Error Occurs When Printing a Diagram* on page 58
- *ROSSERT_NO_FEEDBACK - Prompting for Information When an Exception Occurs* on page 58
- *ROSSERT_TORNADO_TIMEOUT - Modifying the Default Timeout for wtx Commands* on page 59
- *Referenced Configuration for Nucleus Does Not Include Socket Support* on page 59

Synchronizing Code between Rational Rose RealTime and Eclipse

In Rational Rose RealTime, codesync only occurs when a single source file is updated and saved in Eclipse. In Eclipse, do not use the **File > Save All** menu option to save your changes. You must save one file at a time for codesync to occur in Rational Rose RealTime.

Rational Rose RealTime and Eclipse Integration Does Not Support Java

The Rational Rose RealTime and Eclipse integration does not support Java models; the integration only works with C and C++ models.

Using Rational Rose RealTime with a Node Locked License Does Not Warn User About Expiration

Previously, Rational Rose RealTime prompted users before the expiration date of their node locked license. Now, Rational Rose RealTime does not warn users before the license expires. When the license expires, you receive the following error message:

Unable to Obtain License.

Using Pathmaps in Rational Rose RealTime

When using pathmaps in Rational Rose RealTime, you can use the following symbols:

- @ - Represents the path where the model file (for example, the **.rtmdl** file) is stored.
- & - This symbol represents the path where the controlled unit's file is stored, and it can be applied to any controlled unit.

For example, for a model named **C:\Models\myModel.rtmdl**, the symbol @ maps to **C:\Models**. The root logical view file **LogicalView.rtlogpkg** is located in the child directory of the model. Therefore, for the root logical view, the symbol & maps to **C:\Models\myModel**.

To use the file identified in the example:

- 1 Add an external file reference to any model element by right-clicking on a model element in the model explorer, and click **New > File**.
- 2 After adding the file, save the model and open the appropriate model file in a text editor.
- 3 If there are no controlled units, open the model file. If there are controlled units, open the file of the unit in which you added the external file reference.
- 4 Look for the name of the external file you added in the toolset. If the external file reference is located in a path that is under the model's file structure, use the & or @ symbols instead of an absolute path.

Note: You can define a pathmap variable in Rational Rose RealTime that maps to the & symbol. For example, **CURDIR=&**. If this pathmap exists, use **CURDIR** instead of the & symbol.

Limitations on the Number of Open Windows

GDI handles are required to create graphic objects (such as windows, menus, cursors, and bitmaps). Opening windows consumes handles. We recommend that you open a reasonable number of windows. This means that on Windows platforms, do not to exceed 200 open windows at any given time.

Limitations in the Specification History List

The following limitations apply to items in the **Specification History** list:

- If an RTS object (Runtime View) is **locked**, it will not be loaded into the **Specification History** window with the workspace the next time the model is opened.
- RTS objects are visible in the **Specification History** window after RTS shutdown. However, if you attempt to open this object with no Runtime View running, the object is removed from the **Specification History** list (this also occurs when using the **Refresh** menu item).

Loading a Workspace might Cause Default Color Settings to Change Permanently

When opening a model and its corresponding workspace, if the workspace specifies a different color scheme for the **Line** and **Fill** colors (the third and fourth color boxes in the **Custom colors** area in the **Color** dialog box), it causes the default color settings in the file **RoseRT.ini** to change permanently.

To restore the default colors:

- 1 Click **Tools > Options**.
- 2 Click the **Font/Colors** tab.
- 3 In the **Custom Colors** area, set the following options:
 - For **Line Color**, select burgundy.
 - For **Fill color**, select light yellow.

Alternatively, you can choose not to open the workspace when you open a model.

Do Not Use \$& When You Define a PathMap

Do not use \$& in your PathMap definitions. If you define a PathMap that includes \$&, you will receive a build error similar to the following:

**Error: INTERNAL ERROR: File I/O error while trying to open
\$&/ComponentView.rtcmppkg**

Use Caution When Modifying OutPutDirectory

Do not prepend **OutputDirectory** if **ExecutableName** appears to be an absolute path.

Using Rational Rose RealTime on Non-English Installations Causes an Unreadable Font When Viewing Generated Code

When you use Rational Rose RealTime on non-English installations, if the specified font is not available, the default font is mapped to an equivalent that is unreadable. To resolve this problem, click **Tools > Options**, click the **Fonts** tab, and then select another font.

Unable to use Parameters with the cm_getcaps Script

The **cm_getcaps** script returns a set of capabilities supported by a source control system. The Rational Rose RealTime toolset does not implement the **Parameters** button for this operation. This means that you cannot specify any parameters for the **UnCheckout** operation specified for **cm_getcaps** scripting.

Problems Compiling Java Models

If you have QuickTime for Java installed, you might encounter problems when you compile your Java models because QuickTime for Java embeds double quotation marks in **CLASSPATH**. If you remove the double quotation marks, Rational Rose RealTime for Java will compile your models.

If your **CLASSPATH** is too long, you will receive the error message "**Unable to execute.**"

Problems Connecting to a Target

If you receive the message "**Unable to connect to target**" when you attempt to connect to a target (both host and embedded), change the value for the **Connect Delay** box on the **Component Instance Specification** dialog by increasing the value by two or more seconds.

Using Sequence Diagrams

There are a few conditions under which a Sequence Diagram will incorrectly draw messages and FOCs (Focus of Control). If a message or FOC appears to be drawn incorrectly, select the message (or the message that starts the FOC) and, using the center re-orient handle, move the message slightly. Moving the message causes the Sequence Diagram to recalculate the correct display values for that message.

Using the Debugger-xxgdb Tool and Running your Component Instance

When you use the **xxgdb** tool, to enable breakpoints, run your component instance, add any breakpoints, and then restart **xxgdb**.

Using C and C++ Add-ins

You cannot use the C and C++ add-ins at the same time in Rational Rose RealTime.

Code Generator Runs Out of Memory When Generating Very Large Models

When you attempt to generate a very large model, the code generator might run out of memory. To improve memory usage when you generate very large models, we recommend that you control all the controllable elements in your model as individual units.

To control elements in your model:

- 1 On the **Model View** tab in the browser, select a model.
- 2 Right-click and click either **File > Control Units** or **File > Control Child Units**.
- 3 When prompted to control all child units recursively, click **Yes**.
- 4 Click **Yes** in the subsequent confirmation dialog.

For details on controlled units, see the topic, "What is a controllable Element?" in the *Guide to Team Development, Rational Rose RealTime*.

Using the Get and Set Methods in the Attribute and Operation Tools

In the Attribute and Operations tools, you can clear the **Get method** and **Set method** check boxes.

To prevent the deletion of modified code for the Get and Set methods:

- 1 Open the Specification dialog for an **Attribute** or **Operation**.
- 2 Click the **General** tab.
- 3 In the **Documentation** box, delete the following text:

`//GENERATED BY ATTRIBUTE TOOL`

or

`//GENERATED BY OPERATION TOOL`

Deleting this text prevents the deletion of the modifications you made to the Get or Set methods if you cleared either the Get method or Set method options.

Web Publisher Applet Does Not Load Properly

When you use Microsoft Internet Explorer with Microsoft JIT, the **Documentation** window and the Scroll Bar will not appear the first time you start Rational Rose RealTime. Also, if you use JDK version 1.3.3, the Scroll Bar will not appear.

If you perform a refresh, the **Documentation** window and the Scroll Bar will appear. If you use **appletviewer** on JDK version 1.2.2, the **Documentation** window and the Scroll Bar will appear.

Using the Frameworks Dialog

If you create a framework whose name comes alphabetically before the Empty, RTC, and RTC++, by default, on the **Frameworks** dialog the highlighted icon is the RTC++ icon, but the text on the **Description** tab is for the RTC framework. If you click **Open**, the RTC framework is loaded.

Workaround

Always use your mouse to select a framework, and then click **Open**.

Scoping Descriptors for Nested Classes

If you have the following externally defined classes:

```
class ExternalClassA
{
    public:
        int x;
        class ExternalClassB
        {
            public:
                int y;
        }
}
```

when you model them in Rational Rose RealTime by **ExternalClassA** and a nested **ExternalClassB**, for both, **generateClass** is cleared and **generateDescriptor** is selected. The type descriptor for **ExternalClassB** is not generated.

The descriptor for the nested class must be scoped within the top-level class because the nested class is not necessarily public. Because the top-level class is not generated, the descriptor for the nested class (which must be part of the declaration of top-level class), cannot be generated.

No Codesync Support for Java

Although codesync options might appear on the menus, Rational Rose RealTime does not support codesync for Java.

Using the GetSelected Functions

In Rational Rose RealTime, the browser selections only apply when the browser window has focus (Rational Rose RealTime uses a different definition of what is selected than Rational Rose). In all other cases, the selection is from the active diagram window (if any). When you run a script, you are in the script window; the browser window does not have focus. To return focus to the browser window, you can add a delay to your script.

The **GetSelected** functions (for example, **RoseRTApp.CurrentModel.GetSelectedClasses**) only find the selected elements in the model browser if the model browser is active. This means that if you select elements in the browser, and then run your script (from the Script Editor, or the **Tools** menu), the model browser is no longer active (the selected elements are gray, not blue) and these functions will not find anything.

For example, if you open the following script in Rational Rose 2000, select your classes in the model browser, and then run the following script:

```
Sub Main ()
    Dim theSelectedClasses As ClassCollection
    Viewport.Open
    Viewport.Clear
    Print "Selected classes from model browser:"
    Set theSelectedClasses = RoseApp.CurrentModel.GetSelectedClasses()
    If theSelectedClasses.Count > 0 Then
        For i = 1 To theSelectedClasses.Count
            Print "Class name: "; theSelectedClasses.GetAt(i).Name
        Next i
    Else
        Print " No classes have been selected "
    End If
End Sub
```

However, this script will not work in Rational Rose RealTime. It will only work in Rational Rose RealTime if you make the model browser active. For example, if you add the sleep line to the following script, run the script, select the classes in the model browser, and wait 5 seconds, it will find the selected classes.

```
Sub Main ()
    Dim theSelectedClasses As RoseRT.ClassCollection
    Viewport.Open
    Viewport.Clear
    Sleep(5000)
    Print "Selected classes from model browser:"
    Set theSelectedClasses =
    RoseRTApp.CurrentModel.GetSelectedClasses()
    If theSelectedClasses.Count > 0 Then
        For i = 1 To theSelectedClasses.Count
            Print "Class name: "; theSelectedClasses.GetAt(i).Name
        Next i
    End If
End Sub
```

```
Else  
Print " No classes have been selected "  
End If  
End Sub
```

Using the Find Command Might Return Too Many Results

When using the **Find** command, the results returned may be more than expected. We recommend that you refine your searches whenever possible.

No Support for Automatic Unwired Port Registration for Java

In Rational Rose RealTime for Java, **rtBound** messages do not work because there is no support for unwired port registration.

Error Occurs When Printing a Diagram

If you use an 8 bit, 256 color terminal, you may receive the following error message when you print a diagram from your model if the graphics use the default gray shading:

Error Writing to print file <file_name>.

To print diagrams within your models, you must redefine the default fill color.

To redefine your fill color:

- 1 On the **Tools** menu, select **Options**.
- 2 Click the **Font/Color** tab.
- 3 Click **Fill Color**.
- 4 Select a color other than the default gray.

ROSERT_NO_FEEDBACK - Prompting for Information When an Exception Occurs

On Windows-hosted versions of Rational Rose RealTime, a dialog appears after an exception occurs that allows you to specify information about the steps leading up to the exception, your current system configuration, and whether you want to be contacted.

On Linux-hosted and UNIX-hosted versions of Rational Rose RealTime, a script runs after an exception occurs, that has the same functionality as that provided by Windows-hosted versions. Whether you specify the requested information or decide not to specify any information, the script runs **RoseRT -cleanup**.

Note: After any Rational Rose RealTime UNIX exception, you must run **RoseRT -cleanup** to ensure that any remaining crash artifacts are removed.

On Linux -hosted and UNIX-hosted versions, if you do not want to be prompted for information after exceptions, set the environment variable **ROSSERT_NO_FEEDBACK** to **True**. When this variable is set, **RoseRT -cleanup** will continue to run after an exception. Later, if you want to be prompted to provide feedback, set the variable **ROSSERT_NO_FEEDBACK** to **False**.

ROSSERT_TORNADO_TIMEOUT - Modifying the Default Timeout for wtx Commands

By default, when you download a VxWorks module to a target, the **timeout** value for **wtx** commands is set to 30 seconds (**ROSSERT_TORNADO_TIMEOUT=30000**). This default value might not be sufficient when you download large modules, or when you use a slow network. You might receive an error if the **timeout** value is exceeded. Use this variable to increase the **timeout** period, if required.

Referenced Configuration for Nucleus Does Not Include Socket Support

The reference configuration for Nucleus does not include socket support. This can be seen by the definition of **HAVE_INET** in

\$ROSSERT_HOME/C++/TargetRTS/target/NUCLEUS11T/RTTarget.h. If

HAVE_INET is set to zero (0), Target Observability will not work. For example, the following definition will not work:

```
#define HAVE_INET 0
```

You can use the source code included in

\$ROSSERT_HOME/C++/TargetRTS/src/target/NUCLEUS/RTinet for socket support as a starting point; however, the code has not been tested and is not supported.

Online Help Issues

For information on online Help issues, review the following topics:

- *Navigating Through the Online Help* on page 60
- *Viewing Animated Demonstrations* on page 60
- *Problems Accessing Rational Rose Help while Running Rational Rose RealTime on Windows* on page 61
- *Using Context Sensitive Help Might Cause Message to Display* on page 61
- *Maintaining a Single Favorites List* on page 61
- *Using the Index Tab in the Online Help* on page 62

Navigating Through the Online Help

You can either select a topic on the **Contents** tab or click the **Next** or **Previous** buttons, and the topic appears in the main window. As you click **Previous**, although the previous topic appears in the main window, the incorrect topic might be highlighted on the **Contents** tab. However, this does not affect the navigation to the selected topic.

Viewing Animated Demonstrations

Rational Rose RealTime includes animated demonstrations of various topics. To see the current list of animated demonstrations, on the **Help** menu, click **Contents**. The Rational Rose RealTime Online Help Start Page appears. Click the **Animated Demonstrations** button for your specific platform.

You can view Viewlets on several platforms. However, your browser must have Java 1.1 and JavaScript support to play a Viewlet. Additionally, Java must be enabled in your browser.

It is normal for some larger Viewlets to take 10 to 15 seconds to load. Ensure that your browser cache is not disabled; otherwise, some Viewlets may take longer to load.

If a Viewlet takes a long time to open, or a gray box appears instead of the Viewlet, try running the Viewlet in your browser.

To run a Viewlet from your browser:

- 1 In your browser, navigate to the `<ROSERT_HOME >/Help` directory.
- 2 Select `rosert_watch_demos.htm`.
- 3 Select a Viewlet to run that demonstration.

Problems Accessing Rational Rose Help while Running Rational Rose RealTime on Windows

If you run Rational Rose RealTime and Rational Rose at the same time, Rational Rose might not have access to its Help files. When accessing the online Help for the Rational Rose RealTime toolset, Rational Rose RealTime temporarily replaces a registry key value that is installed by Rational Rose. This registry key is located in [HKEY_LOCAL_MACHINE\Software\Rational Software\Rose] and is named **HelpFileDir**. When the toolset starts, it substitutes this key's value for its own **Help** directory. The value installed by the Rational Rose installation program is backed up and restored when the last instance of the running toolset closes.

Workaround

You can disable this behavior by setting the value of the key in [HKEY_LOCAL_MACHINE\Software\Rational Software\Rose RealTime\6.5], whose name is **ReplaceRoseHelpDir**, to **No**.

Using Context Sensitive Help Might Cause Message to Display

If you have version 4.7.3 of the HTML Help compiler installed on your computer, you will receive the following message:

The window name "Default" passed to HH_GET_WIN_TYPE has not been specified.

To solve this problem, install version 4.7.4 of the HTML Help compiler.

Maintaining a Single Favorites List

The Rational Rose RealTime online Help is modularized. If you select **Add** on the **Favorites** tab to add the current Help topic to your favorites list, this entry only appears in the favorites list for that component of the online Help.

To maintain a single list of favorite Help topics:

- 1 Use the **Search** or **Index** tabs to find the desired online Help topic.
- 2 Click the **Locate** button in the Toolbar to see where this Help topic appears in the **Contents** tab.
- 3 Close the online Help.
- 4 Open the online Help by clicking **Help > Contents**.

- 5 From the **Contents** tab only, find the Help topic.
- 6 Click the **Favorites** tab.
- 7 Click **Add**.

Using the Index Tab in the Online Help

When using the Rational Rose RealTime online Help system, if the **Index** tab has no index entries, close the online Help, and then open the online Help again. If the **Index** tab remains empty, delete the file `%ROSET_HOME%/Help/rosert_rrtolhst.chw`, and then open the online Help again.

Documentation Updates

For information about updates to the documentation, review the following topics:

- *Update to Abort Documentation* on page 62
- *Update to Code for Example RTMessage_getPortIndex* on page 63
- *Update to Code for Example RTMessage_getData* on page 64
- *Update to Documentation for Launching Model Integrator* on page 64
- *Code Example for rts and RTCapsule_context Causes Compilation Errors* on page 65
- *Update to Example Model for Type Descriptors* on page 66
- *Update to Referenced Host Configurations Information* on page 66
- *Updated Reference to Example Model for Type Descriptors* on page 67

Update to Abort Documentation

The online documentation for abort is incorrect. Currently, it states the following:

```
RTController::abort  
  
void abort( void );
```

Calling this operation on any controller terminates the controller on which the capsule instance runs. The results is that it will destroy all capsule instances running on that controller. Messages that have not been processed are deleted. Calling this operation on the main thread causes the destruction of all threads, and the process quits. Instead, you can call `context()->abort()`. This operation also terminates the main thread, all threads are destroyed, and then the process quits. Messages that were not processed are deleted.

Note: Depending on the implementation used, the debugger thread may continue to run.

Update to Code for Example `RTMessage_getPortIndex`

The following code from the online Help example for `RTMessage_getPortIndex` located in **/C/C Reference /Services Library API Reference /Minimally Configured Services Library/RTMessage_getPortIndex** causes compilation errors.

Currently, the example is as follows:

```
const RTMessage * msg = RTCapsule_getMsg( this );  
/* reply to message */  
RTPort_sendAt(  
    port,  
    RTMessage_getPortIndex( msg ),  
    RTPort_createOutSignal( port, reply ),  
    RTPriority_High,  
    &someData,  
    &RTType_typeOfSomeData );
```

To avoid compilation errors, change the example to the following:

```
const RTMessage * msg = RTCapsule_getMsg( &this->std );  
/* reply to message */  
RTPort_sendAt(  
    &this->port,  
    RTMessage_getPortIndex( msg ),  
    RTPort_createOutSignal( port, reply ),  
    RTPriority_High,  
    &someData,  
    &RTType_typeOfSomeData );
```

Update to Code for Example `RTMessage_getData`

The code from the online Help example for `RTMessage_getData` in */C/C Reference/Services Library API Reference Minimally Configured Services Library/RTMessage_getData* causes a compilation error.

Currently, the example is as follows:

```
aDataType dt =  
*(aDataType *)RTMessage_getData( RTCapsule_getMsg( this ) );
```

Change that code to this code:

```
aDataType dt =  
*(aDataType *)RTMessage_getData( RTCapsule_getMsg( &this->std ) );
```

Update to Documentation for Launching Model Integrator

The online Help for Model Integrator (from the **Contents** tab, click **Model Integrator Guide > Introduction > Using Model Integrator from the Command Line**) is as follows:

To Launch Model Integrator from the command line:

For Windows, type **modelintRT**.

For UNIX, type **RoseRT -modelintRT**.

The correct UNIX command to start Model Integrator from the command line is as follows:

RoseRT -modelint

Code Example for `rts` and `RTCapsule_context` Causes Compilation Errors

The code from the online Help example for `rts` and `RTCapsule_context` in */C/C Reference/ Services Library API Reference/Minimally Configured Services Library/rts* and `RTCapsule_context` causes compilation errors.

The code example is as follows:

```
int result =

RTPort_send( port,RTPort_createOutSignal( port, hey ),
RTPriority_General, (void *)0, (RTObject_class
*)0 );

if( ! result )
{
    RTController * context = RTCapsule_context( this );
    log.show("Error on physical thread: ");
    RTLog_show( RTController_name( context ) );
    RTController_perror( context, "send");
}
```

The following corrections are required for the online Help example to compile without errors:

- Change the first parameter of the method `RTPort_send()` to be a **const `RTPort *`**. Use **`&this->port`** because `port` refers to the name of the port.
- The method `RTCapsule_context(const RTCapsule *)` expects a parameter that is a pointer to a **`RTCapsule`**. The parameter `this` in the example points to a capsule's **`instanceData` struct**. Use **`&this->std`** as a pointer to the **`RTCapsule`**.
- There is no `log.show()` method. Use `RTLog_show_string()`.
- There is no `RTLog_show()` method. Use `RTLog_show_string()`.

After you apply the changes, the example will look as follows:

```
int result =

RTPort_send( &this->port,RTPort_createOutSignal( talk, ack ),
RTPriority_General, (void *)0,
(RTObject_class *)0 );

if( ! result )

{

    RTController * context = RTCapsule_context( &this->std );

    RTLog_show_string("Error on physical thread: ");

    RTLog_show_string( RTController_name( context ) );

    RTController_perror( context, "send");

}
```

Update to Example Model for Type Descriptors

The Help for type descriptors (from the **Contents** tab, **C++ Reference > Model Properties Reference > Type Descriptors > Example Usage Patterns and Associated Type Descriptors**) refers to an model **Example.rtmidl**. The correct model to use is as follows:

\$ROSERT_HOME/Examples/Models/C++/TypeDescriptors/typedescript.rtmidl

Update to Referenced Host Configurations Information

In the online Help, from the *Installation and Getting Started Guide*, select the topic **Referenced Configurations and Toolchain Requirements** then **Referenced Host Configurations**. Table 6 states the following:

Host Configuration(s): Windows

Target RTOS: Nucleus 1.1

Compiler/Processor: Diab 4.2b, ppc

RTS Library: C++

Connexis DCS Library : -

The correct information is as follows:

Host Configuration(s): **Solaris**, Windows

Target RTOS: Nucleus **11**

Compiler/Processor: Diab 4.2b, **ppc**

RTS Library: C++

Connexis DCS Library : -

Updated Reference to Example Model for Type Descriptors

On the **Contents** tab in the online Help, the type descriptor topic **C++ Reference > Model Properties Reference > Type Descriptors > Example Usage Patterns and Associated Type Descriptors** refers to a model **Example.rtmidl**. The correct name and location for the example model file for type descriptors is as follows:

\$ROSE_HOME/Examples/Models/C++/TypeDescriptors/typedescript.rtmidl

Using Type Descriptor Functions

The TargetRTS Services Library uses type descriptors when it manipulates objects of a class type. The five generated type descriptor functions for a given class are **init**, **copy**, **destroy**, **encode**, and **decode**.

When the object is...	The following functions are used...
Output to the System Log, trace window, or watch window	encode
Modified at run-time in the watch window	init, copy, destroy, decode
Injected in a message	init, copy, destroy, decode
Sent by value (without Connexis)	copy, destroy
Sent by value (with Connexis)	init, copy, destroy, decode, encode
Sent by value (with Connexis, but in the same process)	copy, destroy

If the class is a subclass of **RTDataObject** or used in **RTSequenceOf**, or **RTWrapper** operations, the five type descriptor functions can also be used.

Note: For ObjecTime models converted into Rational Rose RealTime, only use the **RTDataObject**, **RTSequenceOf**, and **RTWrapper** classes for backward compatibility.

Using Globally Unique Identifiers (GUIDs)

4

Contents

This chapter is organized as follows:

- *Advanced Handling of Globally Unique Identifiers (GUIDs)* on page 69
- *Generating GUIDs* on page 71
- *Managing GUIDs* on page 72
- *Known Issues with GUIDs* on page 75

Advanced Handling of Globally Unique Identifiers (GUIDs)

Rational Rose RealTime assigns Globally Unique Identifiers (GUIDs) to primary model elements (such as classes, packages, components, and use cases). These unique identifiers allow Model Integrator to easily identify model changes (such as when a model element is renamed or moved) and merge elements from contributor models.

In previous releases of Rational Rose RealTime, all GUIDs were computed using a time-based algorithm to ensure that the GUIDs were unique. GUIDs for additional element properties (such as operations, attributes, states, transitions, and triggers) are optional and, by default, are not set. Enabling and then disabling optional GUIDs resulted in the removal of any previously existing optional GUIDs from the model.

Most model elements (such as classes, capsules, and protocols) always have GUIDs generated; however, the generation of GUIDs for a number of other elements can be enabled or disabled through the user interface. Rational Rose RealTime generates GUIDs for these model elements when you select the option **Generate for all elements** in the **Unique Identifiers** area on the **General** tab of the **Model Specification for Model** dialog box for the top model file.

Problem

Before enabling the feature to turn on GUIDs on in a multi-stream or multi-model environment, it was necessary to collapse all the models and streams into a single model. If a model element with a unique ID was imported into a model with this feature not set, the optional GUIDs were removed. If a model element without a unique ID was imported into a model with this feature set, a unique ID was added. Because the unique ID was not always preserved, it was important to ensure that all models had this feature enabled to ensure that unique identifiers were preserved.

Resolution

The current GUIDs feature provides a controlled mechanism for propagating optional GUIDs into a multi-stream model environment. This feature supports a one-time upgrade of a model to enable GUIDs without having to create a single master model. It introduces an alternate position-based, not time-based, algorithm for GUID generation, which guarantees that the optional GUID for a property will be the same in all models where the element property has the same name and resides in the same location. For example, **NewClass1** in model A and B has a state machine. Each state machine has a state S1. The old behavior of GUID generation would have assigned a time-based GUID in each model causing merge conflicts. The new algorithm uses a name-position algorithm to assign the same GUID in each model.

If a model element property has a GUID and it is loaded into a model with optional GUIDs disabled, the GUID will be preserved. If a model element property without a GUID is loaded into a model with optional GUIDs enabled, a GUID will not be added. It is possible to have optional GUIDs enabled, but no GUIDs present in the model. This new feature provides an easy method to determine if optional GUIDs are missing from a model, and it also provides you with an easy method to add additional alternate GUIDs if optional GUIDs are not present. In rare circumstances, the alternate GUID algorithm can generate a GUID which conflicts with a time-based GUID. For these situations, the new feature supports the generation of a single GUID. To activate the GUIDS feature for optional model elements, see *Managing GUIDs* on page 72.

Note: By default, all models created from Rational Rose RealTime Frameworks will have the GUIDs feature activated. We strongly recommend that you do not change the default settings for generating GUIDs in your models to ensure optimal Model Integrator merge sessions.

Generating GUIDs

Previously, before you could enable the **Generate for all model elements** option, all development streams had to be merged into a single stream. Rational Rose RealTime now includes a new method of GUID generation which attempts to generate the same GUIDs for the same elements across development streams.

Recommended Steps for Enabling GUIDs in Multi-Stream Development

To avoid conflicts between time-based GUIDs and alternate GUIDs, you can enable optional GUIDs in a single model in a single stream. If you have a multi-stream development environment where GUIDs are not enabled, we recommend that you merge all streams into a single stream before enabling optional GUIDs. (For a detailed explanation, see the topic *To set the Generate unique identifiers for all elements option* in the online Help.)

For those cases where merging into a single stream is not practical, this feature also supports the incremental introduction of GUIDs into a multi-stream environment. When optional GUIDs are added incrementally in a multi-stream environment, there is a higher risk for a GUID collision (where different model elements share the same GUID).

To minimize the change and impact of collisions:

You want to ensure that no duplicate GUIDs (different model elements with the same GUID) exist in any of the streams because duplicate GUIDs will complicate the process.

- 1 Open each model from the development streams and select **Add/turn on unique identifiers**.

Note: We recommend adding optional GUIDs one model at a time and then propagating the results across a team before starting the next model.

- 2 Open Model Integrator and perform a trial merge of the models.
- 3 Load the resulting model you created from the trial merge process into Rational Rose RealTime.

Note: If you receive the following error message:

Warning: This model has multiple objects with the same unique id (XXXXXXXXXXXX).

record the duplicate unique IDs, and then identify them in the stream in which they appear.

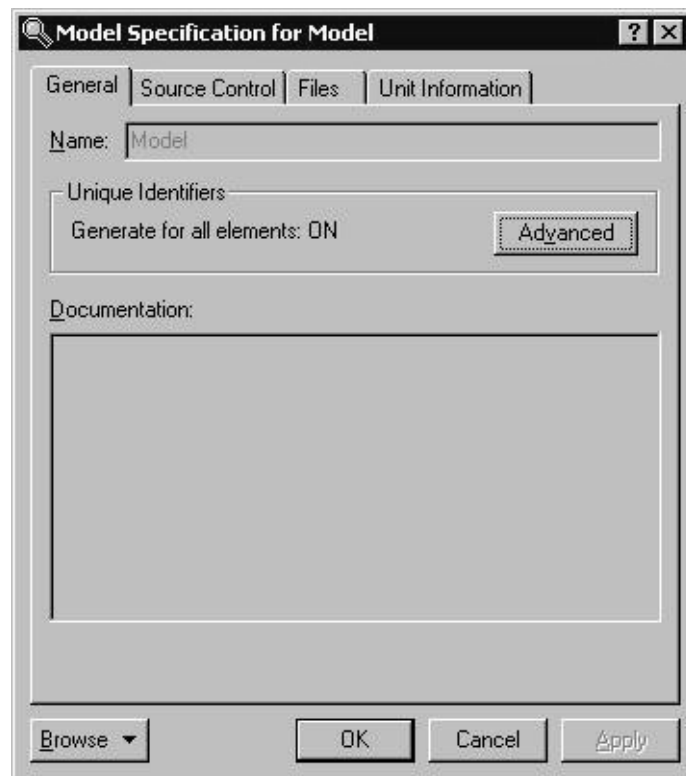
- 4 For each of the duplicate IDs you encounter, open the model corresponding to the ID and then select **Regenerate unique identifier** to regenerate it.

Note: This scenario is not likely to occur.

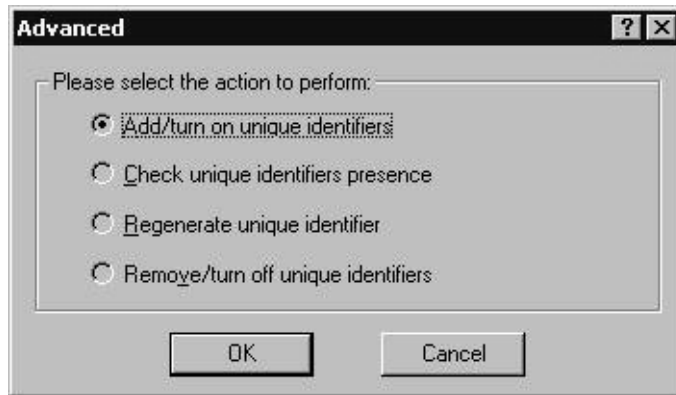
- 5 Perform a merge again to ensure that all duplicate GUIDs have been resolved. If there are no errors, save the result as the new baseline.

Managing GUIDs

The **General** tab in the **Model Specification for Model** dialog box includes the **Generate for all elements** option that specifies the current state of a model for generating unique identifiers for the model: **ON** or **OFF**. By default, this option is set to **ON**.



Click **Advanced** to select an action to perform with the unique identifiers.



Add/turn on unique identifiers

Activates the GUIDs feature and adds optional GUIDs to the elements that support optional GUIDs, but do not currently have a GUID.

When you select this option, the following dialog box appears:



Click **Yes** to start the process.

Note: To see details in the log, ensure that the toolset option for command logging is enabled. Click **Tools > Options**, and then set the **Log commands** option on the **General** tab. The **Log** tab in the **Output** window shows the number of GUIDs generated using a message similar to the following:

Generated NNN hash GUIDs.

where **NNN** represents the actual number of GUIDs generated in the current model by Rational Rose RealTime.

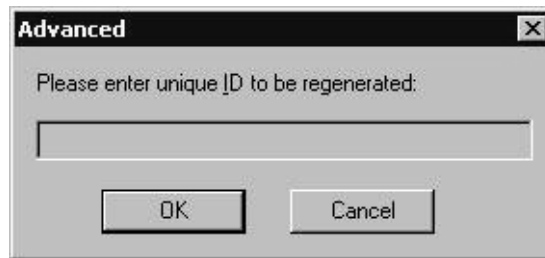
Check unique identifiers presence

Rational Rose RealTime will report the number of model elements that support optional GUIDs that do not currently have optional GUIDs assigned. If Rational Rose RealTime encounters any model elements that do not have optional GUIDs assigned, you must click the **Add/turn on unique identifiers** option.

Regenerate unique identifier

When this option is selected, you can regenerate an alternate GUID for a model element.

When you select this option, the following dialog box appears:



When you specify the GUID to regenerate, type it exactly as it appears in the model file or in the duplicate ID error message. For example:

3EC115960177

If a GUID cannot be found in the model, the model will not be modified and the following error message appears:



If a valid GUID was found in the model, a GUID is regenerated for the first model element found that matches the supplied GUID, and references to the old GUID are replaced with references to the new GUID.

If the model contains more than one element with the specified GUID, the GUID is regenerated for the model element that has an optional GUID.

Note: To see details in the log, ensure the toolset option for command logging is enabled. Click **Tools > Options**, and then set the **Log commands** option on the **General** tab. After the process of regenerating GUIDs completes, check the log for messages.

Remove/turn off unique identifiers

This option deactivates the GUIDs feature and removes optional GUIDs from the model.

Note: We recommend that optional GUIDs always be enabled. Use this feature to remove GUIDs in situations where optional GUIDs are already enabled and you want to replace them with alternate GUIDs.

Known Issues with GUIDs

- Adding or removing unique identifiers might force Rational Rose RealTime to check out controlled units from your source control tool that will not be modified. Typically, those are units that use elements with changing GUIDs, but do not use that GUID to reference the element.
- When using alternate GUIDs, it is possible to have more than one object with the same GUID, which is referred to as a collision. We estimate that the number of collisions is approximately one collision per 10 MB of model for each development stream. This feature automatically resolves collisions in a single model; however, you can expect GUID collisions when enabling GUIDs in different streams. See *Recommended Steps for Enabling GUIDs in Multi-Stream Development* on page 71.
- Naming of junction points on State Diagrams and connectors on Collaboration and Structure Diagrams:
 - Previously, junction points on State Diagrams, and connectors on Collaboration and Structure Diagrams were named **Junction *n***, and connectors were named **C*n*** (where *n* represents a number assigned to the element). Now, these elements are named **J<GUID>** for junction points and **C<GUID>** for connectors, where GUID is replaced with the GUID of the object. This change affects only UI operations and will not change existing models.

- The new naming of junction points on State Diagrams and connectors on Collaboration and Structure Diagrams prevents Model Integrator merge conflicts. For example **contributor 1** adds a transition **t1** from state **S1** to state **S2**. **Contributor 2** adds a transition **t2** from state **S1** to **S3**. This should be a non-conflicting change, but previous versions of Rational Rose RealTime named the originating junction in **S1** for **t1** and **t2** with the same name (for example, **JUNCTION1**), which results in a merge conflict requiring user intervention during the merge process. Using the new naming convention, the merge would be a non-conflicting change and Model Integrator is capable of automatically resolving the changes.

Eclipse and Rational Rose RealTime Integration

5

Contents

This chapter is organized as follows:

- *Integration Overview* on page 77
- *Communication Overview* on page 79
- *Installing the Eclipse and Rational Rose RealTime Integration Software* on page 80
- *Configuring Preferences in Eclipse* on page 81
- *Configuring Preferences in Rational Rose RealTime* on page 89
- *Generating Code from Rational Rose RealTime* on page 92
- *Removing Generated Code from Rational Rose RealTime* on page 92
- *Refreshing an Eclipse Project* on page 93
- *Synchronizing Code Between Rational Rose RealTime and Eclipse* on page 93
- *Editing Code in Eclipse* on page 94
- *Configuring Build Settings in Rational Rose RealTime* on page 98
- *Configuring Build Settings in Eclipse* on page 100
- *Building in Eclipse* on page 101
- *Navigating to Build Errors* on page 103
- *Navigating From Rational Rose RealTime to Eclipse* on page 104
- *Navigating from Eclipse to Rational Rose RealTime* on page 108
- *Getting Started* on page 110
- *Example Workflow* on page 113
- *Troubleshooting* on page 131

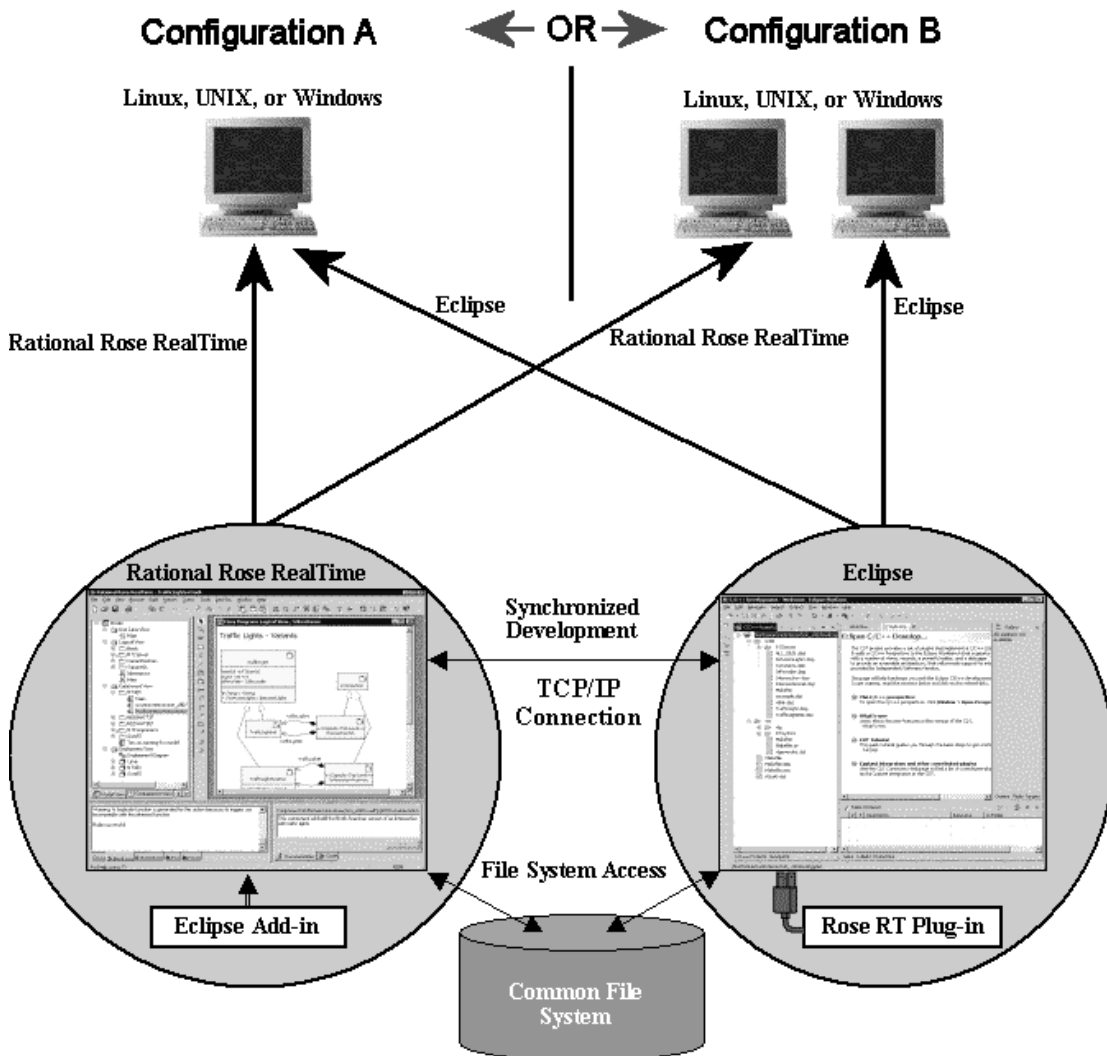
Integration Overview

The Eclipse and Rational Rose RealTime integration links these applications so that they can communicate with each other. The seamless integration lets you leverage the strengths of both applications in a single workflow. This means that you can do your model-centric development in Rational Rose RealTime, while using Eclipse for your code-centric development.

The Eclipse and Rational Rose RealTime integration lets you easily navigate between the two applications, and view data in various formats (Model View in Rational Rose RealTime, and Code View in Eclipse). Meanwhile, your development remains synchronized across the two applications.

Figure 1 shows the integration between Rational Rose RealTime and Eclipse, as well as the different configuration options available for installing the applications.

Figure 1 Eclipse and Rational Rose RealTime Integration



Both applications can run on the same physical computer, or on two different computers. Additionally, when running the applications on two different computers, they can be on different configurations (Linux, UNIX, and Windows). For example, Eclipse can run on a computer with a Windows configuration, while Rational Rose RealTime runs on a computer with a UNIX configuration. However, the following requirements must be met:

- Both computers must have access to a common file system, where the generated code will be stored.
- To perform a build in Eclipse, Rational Rose RealTime and Cygwin must be installed on the computer that runs Eclipse. Rational Rose RealTime does not have to be running because Eclipse only requires access to the libraries and header files in Rational Rose RealTime; the build process requires access to the Rational Rose RealTime Runtime system because it must be able to compile for all targets.
- In Rational Rose RealTime, preferences are associated with a model, and stored in the same directory as the model file, with the extension `.rteri`. Running two or more Rational Rose RealTime sessions simultaneously on the same computer is supported, provided that each session uses its own model file.
- If Eclipse and Rational Rose RealTime are running on two different computers, both computers cannot be located behind two different firewalls, unless one of the computers has port forwarding enabled.
- The integration supports running two or more Eclipse sessions simultaneously on the same computer, provided that each session uses its own workspace.

Note: For additional information about the referenced configurations required for the Eclipse and Rational Rose RealTime integration, see the topic *Referenced Configuration Requirements for the Eclipse and Rational Rose RealTime Integration* on page 21.

Communication Overview

To establish communication between Rational Rose RealTime and Eclipse, one of the applications must be designated as a listener, and the other must be the connector. Both the Rational Rose RealTime plug-in in Eclipse and the Eclipse Add-in in Rational Rose RealTime can act as either a TCP/IP listener, or connect to a listener. The listener has one parameter that specifies the port number to listen to. The connector has two parameters; the host name or IP address, and the port number to connect to.

After a connection is established, it does not matter whether Eclipse or Rational Rose RealTime is designated the listener or the connector. The information sent across the established link is exactly the same regardless of which application is the listener and which application is the connector. By default, Eclipse is designated the listener, and Rational Rose RealTime is the connector.

Configuring a Listener

The listener runs on a separate thread. Once enabled, the listener (either Eclipse or Rational Rose RealTime) starts to listen on a specified port. For Eclipse, the port is specified on the **Communication** tab in the **Preferences** dialog box, and for Rational Rose RealTime, the port is specified on the **Eclipse Integration Settings** dialog. When a connection is made to the listener, that thread services the established connection. Therefore, there can be only one (or no) entity connected to a listener. If an established connection becomes disabled, the listener reverts to its listening state.

Configuring a Connector

The connector runs on a separate thread. Once enabled, the application designated as the connector (either Eclipse or Rational Rose RealTime) attempts to connect to a specified **host name or IP address**, and **port number**. For Eclipse, this information is specified on the **Communication** tab in the **Preferences** dialog box, and for Rational Rose RealTime, this information is on the **Eclipse Integration Settings** dialog. If the specified host is not found, or the connection is rejected, the connector waits for ten seconds before attempting another connection. If an established connection becomes disabled, the connector attempts to connect every ten seconds until it establishes a connection.

Installing the Eclipse and Rational Rose RealTime Integration Software

For instructions on installing the Eclipse plug-in and the Rational Rose RealTime add-in, see the following Rational Rose RealTime patch note file:

[rosert_2004_06_12_GA04_patch_note.html](#).

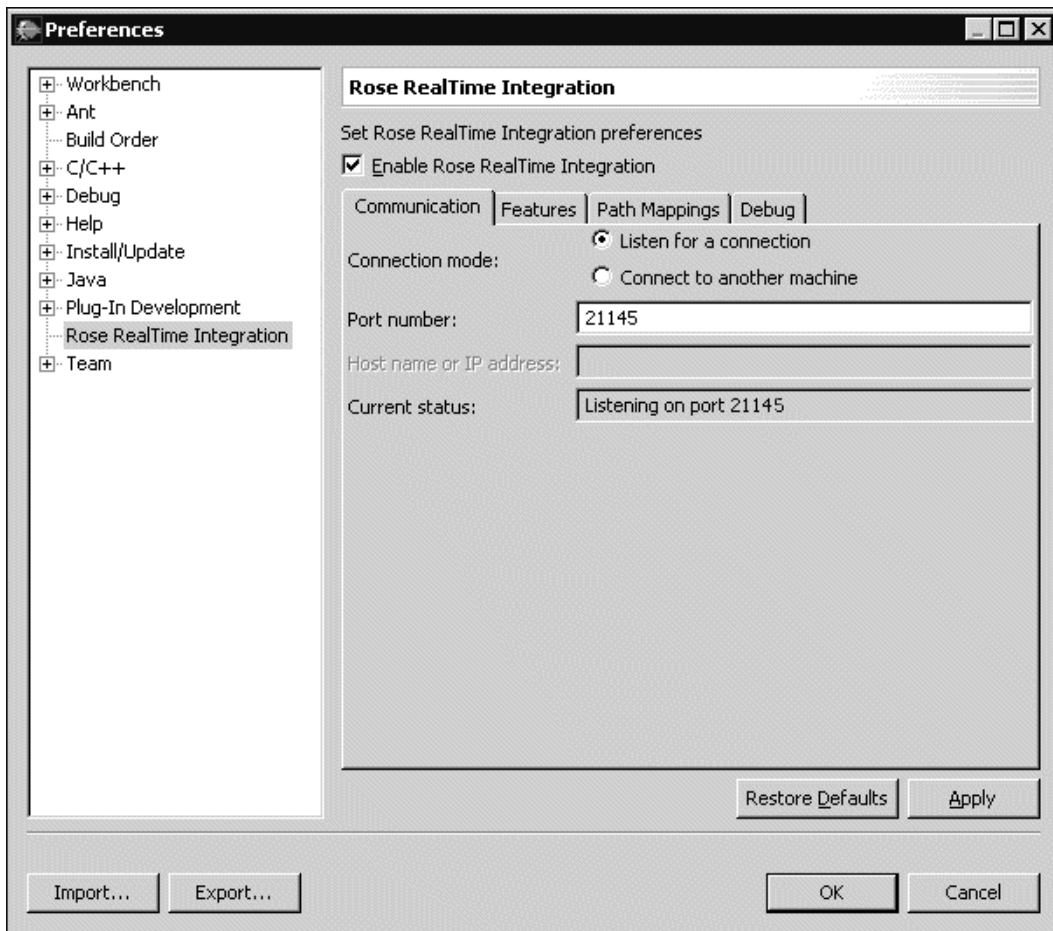
You can find this file on the FTP site where you downloaded this patch.

Configuring Preferences in Eclipse

Although the Eclipse plug-in and the Rational Rose RealTime add-in have basic connectivity configured, you may want to modify these settings from their default values, or verify the current status of your connection.

To access the integration settings in Eclipse, click **Window > Preferences**, and then select **Rose RealTime Integration**. Figure 2 on page 81 shows Eclipse listening (waiting for a connection).

Figure 2 Eclipse - Preferences Dialog Box



Enable Rose RealTime Integration

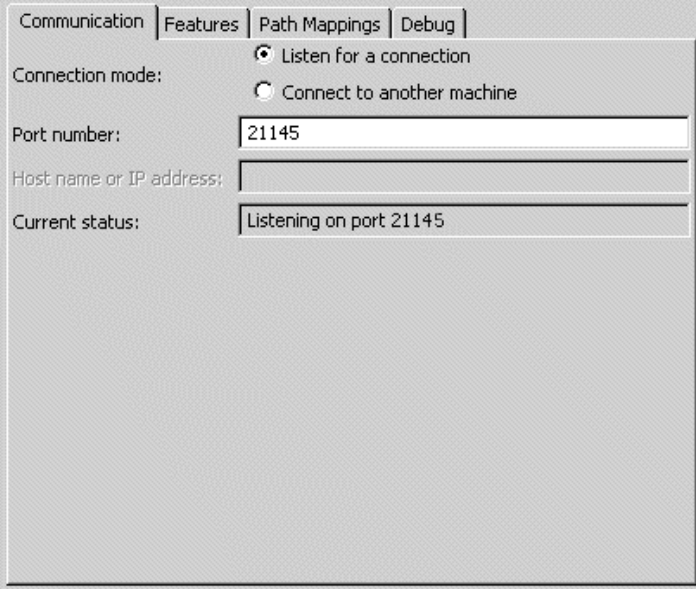
This check box determines whether or not integration occurs. When this check box is selected, Eclipse is able to establish a connection to Rational Rose RealTime. When this check box is cleared, no integration functionality will be available.

Communication Tab

Use the **Communication** tab (Figure 3) to specify how Eclipse will connect to Rational Rose RealTime and to view the current status of the connection. For additional information about connectivity issues, see the topic *Troubleshooting* on page 131.

To access the communication settings in Eclipse, click **Window > Preferences**, select **Rose RealTime Integration**, and then click the **Communication** tab.

Figure 3 Eclipse - Communication Tab



The screenshot shows the Eclipse IDE's Preferences dialog box, specifically the 'Communication' tab for 'Rose RealTime Integration'. The dialog has four tabs: 'Communication', 'Features', 'Path Mappings', and 'Debug'. The 'Communication' tab is active. It contains the following elements:

- Connection mode:** Two radio buttons. The first, 'Listen for a connection', is selected. The second, 'Connect to another machine', is unselected.
- Port number:** A text field containing the value '21145'.
- Host name or IP address:** An empty text field.
- Current status:** A text field displaying 'Listening on port 21145'.

Connection Mode

Specifies whether Eclipse should listen for a connection, or connect to another computer. Either Eclipse or Rational Rose RealTime can act as the listener; however, if one application is the listener, the other must be the connector. By default, Eclipse is set as the listener to establish socket communication at start-up.

Port Number

Specifies the port number that Eclipse listens on, or connects to. The valid range for the port number is between 1024 and 65535. The default value is 21145.

Note: Only one listener can use any specified port number; multiple listeners on the same port are not permitted.

Host name or IP address

Specifies the computer name or the IP address that Eclipse should connect to. When Eclipse is designated as the listener, this option is grayed out.

Current status

Specifies the current state of the connection from Eclipse. The state can be one of the following options:

- **Connected to** *<hostname_or_IP_address>* - A successful connection was made between Eclipse and Rational Rose RealTime. This message occurs in Eclipse and Rational Rose RealTime while the connection remains established.
- **Connecting to** *<hostname_or_IP_address>:<port_number>* - Eclipse is attempting to connect to the other end (Rational Rose RealTime). For additional information about connectivity issues, see the topic *Troubleshooting* on page 131.
- **Can't connect to** *<hostname or IP address>:<port_number>* - Eclipse could not connect to the other end (Rational Rose RealTime) because Rational Rose RealTime was not running, or it is not configured to listen to the specified port. For additional information about connectivity issues, see the topic *Troubleshooting* on page 131.

- **Can't listen to port** <port_number> - Eclipse cannot listen on the specified port number because that port is currently in use. Choose a different port number. For additional information about connectivity issues, see the topic *Troubleshooting* on page 131.
- **Feature not enabled** - The **Enable Rose RealTime Integration** option is currently disabled.
- **Host** <hostname or IP address> **can't be resolved** - The host cannot be found, or the IP address is invalid. For additional information about connectivity issues, see the topic *Troubleshooting* on page 131.
- **Listening on port** <port_number> - Eclipse is currently listening for a connection on the specified port number, but a connection has not yet been made. Eclipse will continue to listen until a connection is established. This message occurs when Eclipse is the listener (**Connection mode** is set to **Listen for a connection**).

Restore to Defaults

Returns all modified settings to their default values.

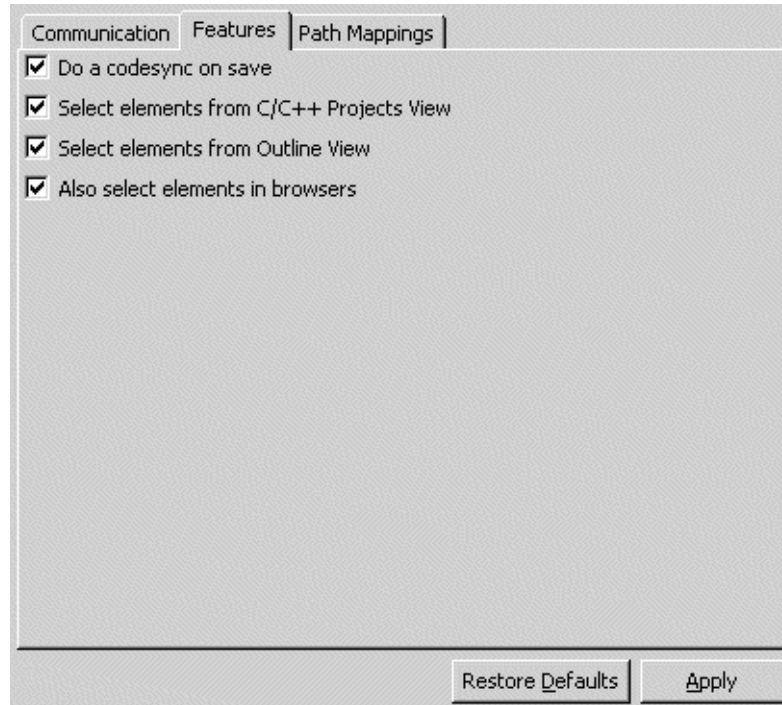
Features Tab

Use the **Features** tab to specify how Eclipse selects elements, and to specify whether a codesync is performed when you save in Eclipse.

To access the feature settings in Eclipse, click **Windows > Preferences**, select **Rose RealTime Integration**, and then click the **Features** tab.

Figure 4 shows the default settings for the **Features** tab in Eclipse.

Figure 4 Eclipse - Features Tab



Do a codesync on save

Specifies that Rational Rose RealTime will perform a codesync when you save code changes in Eclipse. For large components, you may not want this option selected because of the delay involved in synchronizing; however, if this feature is not enabled, synchronization does not occur.



Note: In Eclipse, if you select **Save > Save All**, a codesync will not be performed if more than one file was saved; codesync occurs only when you save one file at a time. For additional information about codesync, see the topic *Synchronizing Code Between Rational Rose RealTime and Eclipse* on page 93.

Select elements from C/C++ Projects View

Specifies when you select a method from the **C/C++ Projects** view in Eclipse, the corresponding model element in Rational Rose RealTime (if any) is selected from its State Diagram.

Note: Some elements selected from the **C/C++ Projects** view in Eclipse do not have a corresponding element in Rational Rose RealTime. Navigation can only occur for user-defined operations, choice points, states (entry and exit), and transitions.

Select elements from Outline View

Specifies when you click on a method from the **Outline** view in Eclipse, the corresponding model element in Rational Rose RealTime (if any) is selected from its State Diagram.

Note: Some elements selected from the **C/C++ Projects** view in Eclipse do not have a corresponding element in Rational Rose RealTime. Navigation can only occur for user-defined operations, choice points, states (entry and exit), and transitions.

Also select elements in browsers

Specifies when you click a method from either the **Outline** view or the **C/C++ Projects** view in Eclipse, that model element is selected in the State Diagram, in the State Diagram browser, and on the **Model View** tab from the main browser in Rational Rose RealTime.

Note: Some elements selected from the **C/C++ Projects** view in Eclipse do not have a corresponding element in Rational Rose RealTime. Navigation can only occur for user-defined operations, choice points, states (entry and exit), and transitions.

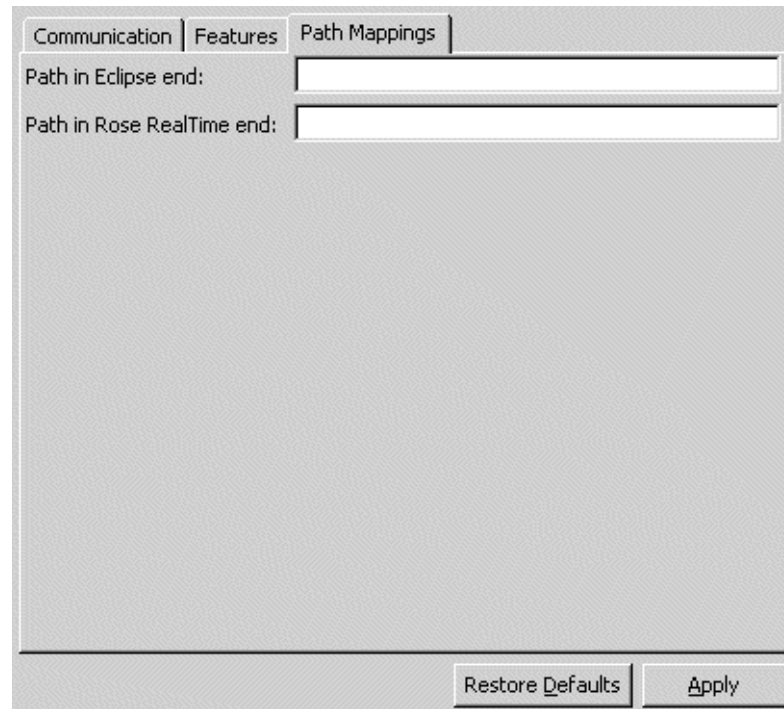
Path Mappings Tab

Use the **Path Mappings** tab to specify the settings for differences in paths between Eclipse and Rational Rose RealTime. You only need to specify differences to the common storage area where the model and code reside.

Note: The path mappings identified on **Path Mapping** tab in Eclipse are not related to the pathmaps specified for a model in Rational Rose RealTime.

To access the path mapping settings in Eclipse, click **Window > Preferences**, select **Rose RealTime Integration**, and then click the **Path Mappings** tab.

Figure 5 Eclipse Path Mappings Tab



Note: In Rational Rose RealTime, you must modify the value for the **TargetServicesLibrary** on the **C++ Compilation** tab (**C Compilation** tab for the C Language) for a component if you generate code on one computer and build it on another, and the computers are on different configurations. For example, if you generate a Rational Rose RealTime model on a Windows computer, and attempt to build it on a computer with a Linux configuration, the build will not be successful. To successfully build when doing cross-platform development, open the **Component Specification** dialog and click the **C++ compilation** tab. Typically, the value for **TargetServicesLibrary** is **\$ROSERT_HOME/C++/TargetRTS**. The code generator will generate a **make** file and expand the **\$ROSERT_HOME** environment variable to the actual value (for example, **C:\Program Files\Rational\Rose Realtime**). This value may not be valid on the computer doing the build. You must modify the environment variable so that it is not expanded by inserting round brackets around the environment variable name. For example:

`$(ROSERT_HOME)/C++/TargetRTS`

Path in Eclipse end

Specifies the path mapping to the location of the file system where the model and code reside for the computer that runs Eclipse.

Path in Rose RealTime end

Specifies the path mapping to the location of the file system where the model and code reside for the computer that runs Rational Rose RealTime.

Specifying Path Mappings

For a successful integration between Eclipse and Rational Rose RealTime, both applications must have to access the file system where the model and code reside. Because Eclipse and Rational Rose RealTime can run on different computers, as well as on different configurations, the Eclipse plug-in and the Rational Rose RealTime add-in must be aware of these path mapping differences in the file system. The following examples show the path mappings required for different configuration options:

- **Eclipse and Rational Rose RealTime are on two computers with different configurations (Windows and UNIX)**

If Rational Rose RealTime runs on a computer with a Microsoft Windows XP configuration, and the model file **mymodel.rtmdl** is stored on a network drive mapped to **W:** in the following directory:

\user\data

the path to the model file is as follows:

W:\user\data\mymodel.rtmdl

Note: On Windows configurations, you must map the path to a drive letter.

And, if Eclipse runs on a computer with a UNIX configuration, its fully qualified file location is as follows:

//mystorage/user/data/mymodel.rtmdl

In this example, Eclipse and Rational Rose RealTime will reference the same location if the following path mappings are specified:

Path in Eclipse end: //mystorage/

Path in Rational Rose RealTime end: W:

Note: You can use "/" or "\" interchangeably when specifying path mappings.

- **Eclipse and Rational Rose RealTime are on the different computers with the same configuration (Windows)**

If Eclipse and Rational Rose RealTime are on a Windows configuration, you want to specify the path differences (in this example, the drive letter) between the two computers. If Rational Rose RealTime runs on a computer with a Windows XP configuration, and the fully qualified location for the model is as follows:

K:\shared\models\mymodel.rtmdl

And, if Eclipse runs on a different computer with a Windows XP configuration where the same networked drive (**K:**) is mapped to **S:**, the fully qualified file location for the model is as follows:

S:\shared\models\mymodel.rtmdl

In this example, Eclipse and Rational Rose RealTime know that they reference the same location if the following path mappings are specified:

Path in Eclipse end: S:

Path in Rational Rose RealTime end: K:

- **Eclipse and Rational Rose RealTime are on the same computer**

In this example, Rational Rose RealTime and Eclipse will not require any path mappings to the file storage location because there will be no differences.

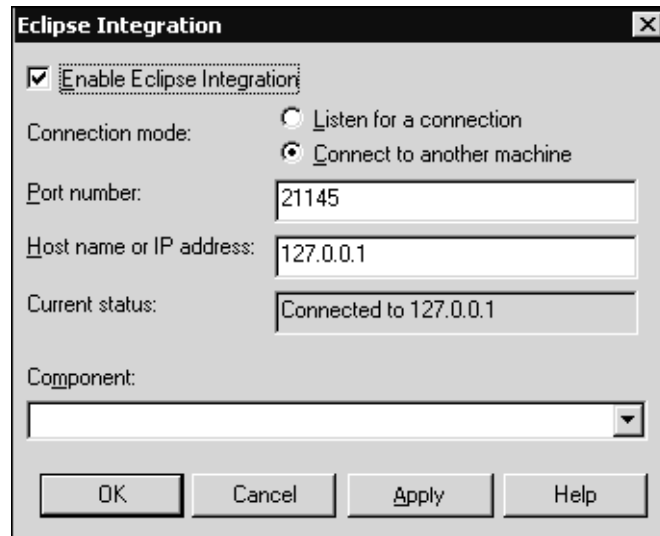
Note: If an existing project is refreshed after the storage location specified on the **Path Mappings** tab changes, Eclipse prompts you to change to the new location. Changes in location can occur if you modify the path mappings in Eclipse, or if you click **File > Save Model As** in Rational Rose RealTime.

Configuring Preferences in Rational Rose RealTime

Although the Eclipse plug-in and the Rational Rose RealTime add-in have basic connectivity configured, you may want to modify these settings from their default values, or verify the current status of the connection.

To configure the integrations settings for Eclipse in Rational Rose RealTime, click **Tools > Eclipse Integration Settings**.

Figure 6 Rational Rose RealTime - Eclipse Integration Setting Dialog Box



Enable Eclipse Integration

This check box determines whether or not integration occurs. When this check box is selected, Rational Rose RealTime is able to establish a connection to Eclipse. When this check box is cleared, no integration functionality will be available.

Connection Mode

Specifies whether Rational Rose RealTime listens for a connection, or connects to another computer. Either Eclipse or Rational Rose RealTime can act as the listener; however, if one application is the listener, the other must be the connector. By default, **Connect to another machine** is selected, and Eclipse is set as the listener.

Port Number

Specifies the port number that Rational Rose RealTime listens on, or connects to. The valid range for the port number is between 1024 and 65535. The default value is 21145.

Note: Only one listener can use any specified port number; multiple listeners on the same port are not permitted.

Host name or IP address

Specifies the computer name or the IP address that Rational Rose RealTime should connect to. By default, the address of the local host (127.0.0.1) is specified to ensure that if Rational Rose RealTime and Eclipse run on the same computer, the connectivity is automatic at start-up. When Rational Rose RealTime is designated the listener, this option is not available.

Current status

Specifies the current state of the connection from Rational Rose RealTime to Eclipse. The state can be one of the following options:

- **Connected to** *<hostname_or_IP_address>* - A successful connection was made between Eclipse and Rational Rose RealTime.
- **Connecting to** *<hostname_or_IP_address>:<port_number>* - Rational Rose RealTime is attempting to connect to the other end (Eclipse).
- **Can't connect to** *<hostname or IP address>:<port_number>* - Rational Rose RealTime could not connect to the other end because Eclipse was not running, or it is not configured to listen to the specified port. For additional information about connectivity issues, see the topic *Troubleshooting* on page 131.
- **Can't listen to port** *<port_number>* - Rational Rose RealTime cannot listen on the specified port number because that port is already in use. Choose a different port number. For additional information about connectivity issues, see the topic *Troubleshooting* on page 131.
- **Feature not enabled** - The **Enable Eclipse Integration** option is currently disabled.
- **Host** *<hostname or IP address>* **can't be resolved** - The host cannot be found, or the IP address is invalid. For additional information about connectivity issues, see the topic *Troubleshooting* on page 131.
- **Listening on port** *<port_number>* - Rational Rose RealTime is currently listening for a connection on the specified port number, but a connection has not yet been made. Rational Rose RealTime will continue to listen until a connection is established.

Component

This field contains a list of all of the components available in the current model, sorted alphabetically. In Eclipse, a component is referred to as a project. If you select a new component from the drop-down list, the Eclipse project also updates to use the new component.

Generating Code from Rational Rose RealTime

Before any navigation can occur between Eclipse and Rational Rose RealTime, generated code must exist for the desired Rational Rose RealTime component. Without generated code, navigation to and from both applications is not possible.

Note: The component used must have settings for the **C++ Generation** tab in Rational Rose RealTime that allows the code generator to function.

To generate code for a Rational Rose RealTime component:

- 1 In Rational Rose RealTime, right-click on a component.
- 2 Do one of the following:
 - For a model that was previously built, you can click **Build > Clean**, and then click **Build > Build**, and select **Generate**. All the generated code files are removed from the project in Eclipse and from the common file storage location, and then new files are generated.
 - For a model that was not previously built, click **Build > Build**, and then select **Generate**. The updates and any new generated code files display in the Eclipse project, and in the common file storage location.

Note: Allow the code generation process to complete before you proceed. Ensure that the last entry on the **Build Log** tab in the **Output** window in Rational Rose RealTime indicates a successful build.

After the generation process completes in Rational Rose RealTime, the corresponding project in Eclipse refreshes to show the generated code files.

Removing Generated Code from Rational Rose RealTime

It may be necessary to remove generated code for a component. Removing generated code updates the **C/C++ Project** view in Eclipse by removing all generated files, excluding the Eclipse file **Makefile.eric** which will continue to be available for the Eclipse project.

To remove all the generated code for a particular component in Rational Rose RealTime:

- 1 In Rational Rose RealTime, right-click on a component.
- 2 Click **Build > Clean**.
- 3 Click **OK**.

Refreshing an Eclipse Project

To refresh an Eclipse project from Rational Rose RealTime:

- 1 In Rational Rose RealTime, right-click on a component.
- 2 Select **Refresh Eclipse**.

In Eclipse, the corresponding project refreshes to show the current elements in the project.

Synchronizing Code Between Rational Rose RealTime and Eclipse

By default, when you save your code in Eclipse, a codesync is automatically performed. You can disable this feature by clearing the **Do a codesync on save** option on the **Features** tab in the **Preferences** dialog box in Eclipse.



Note: In Eclipse, the **Do a codesync on save** option is successful only when saving one file at a time. This means that if you make changes to two or more source files, selecting **File > Save All** in Eclipse does not perform a codesync on all files. We strongly recommend that you save files individually.

The code synchronization process in Eclipse is silent. This means that no confirmation dialog displays in Rational Rose RealTime. The changes are unconditionally synchronized into the model.

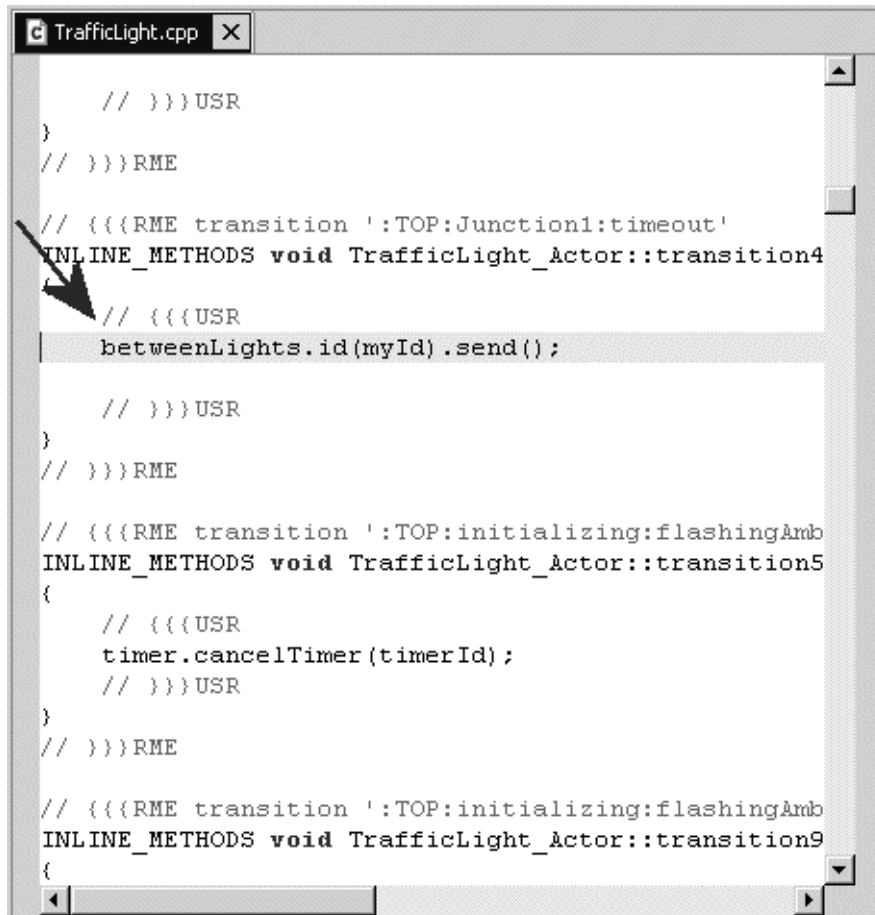


Note: Because Eclipse and Rational Rose RealTime share a common file system, you must use caution when modifying code and building. For example, you can modify code for a model element in Rational Rose RealTime, and then modify the same method in Eclipse for that corresponding model element. Saving the changes in Eclipse (which will cause a codesync to be performed in Rational Rose RealTime) will overwrite the initial changes you made to the model element in Rational Rose RealTime. In addition, if you modify a method in Eclipse without saving, generating the component in Rational Rose RealTime overwrites the modifications you made in Eclipse. We strongly recommend that you save and build your changes as often as possible.

Editing Code in Eclipse

In Eclipse, you can edit user-defined operations, choice point, state (entry and exit), and transition code. To edit method code in Eclipse, the method must contain the `// {{{USR` and `// }}}USR` tags. This means that the typical rules for a codesync apply; that is, changes in the code must be between `// {{{USR` and `// }}}USR` tags for the code associated with the element.

Figure 7 Location for Insertion Code



Note: You cannot edit a non-user operation generated by Rational Rose RealTime, or methods that do not contain **USR** tags. Additionally, you cannot modify attributes. Modifications are only supported for those elements in a `.cpp` or `.c` file.

From Eclipse, you can edit the following code from Rational Rose RealTime:

- *Editing Choice Point, State (Entry and Exit), and Transition Code in Eclipse* on page 96
- *Editing Operation Code in Eclipse* on page 96
- *Editing Capsule and Class Code in Eclipse* on page 97

After editing code in Eclipse, you can synchronize those changes into a Rational Rose RealTime model. When you save code in Eclipse, Rational Rose RealTime automatically performs a codesync. For information on synchronizing changes made to code, see *Synchronizing Code Between Rational Rose RealTime and Eclipse* on page 93.

Note: In Eclipse, if you select **Save > Save All**, a codesync will not be performed if more than one file was saved; codesync occurs only when you save one file at a time. You can disable the **Do a codesync on save** feature on the **Features** tab in the **Preferences** dialog box in Eclipse.

Before You Edit

You can only edit code after Eclipse has created a project containing the generated files from the initial Rational Rose RealTime build.

Note: If the **CodeSyncEnabled** option on the **C++ Generation** tab is disabled in Rational Rose RealTime, **USR** tags are not created when the component is generated. Although there are no **USR** tags, you can navigate to that corresponding element in Eclipse. Instead of positioning the cursor after the `// {{{USR` tag and before the `// }}}USR` tag, the cursor is positioned at the top of the file.

When you open a model in Rational Rose RealTime, and that model has been used previously with an established connection to Eclipse, a connection will automatically be established. Rational Rose RealTime will send the model name and the component name to Eclipse after the following items occur:

- The appropriate path maps were configured on the **Path Mappings** tab in the **Preferences** dialog in Eclipse.
- A successful connection is made between Eclipse and Rational Rose RealTime.
- A component is selected in the **Eclipse Integration Settings** dialog in Rational Rose RealTime.

Upon receiving the model name and the component name, Eclipse creates a C/C++ project with the same name as the component.

Note: If generated files exist from a previous build, in Rational Rose RealTime, click **Build > Clean** before performing a build. Cleaning the generated files removes all generated files in the common files system excluding the file **Makefile.eric**.

Editing Choice Point, State (Entry and Exit), and Transition Code in Eclipse

To edit choice point, state (entry and exit), and transition code:

- 1 In Rational Rose RealTime, open a State Diagram for a classifier (capsule or class) with any combination of choice points, states, or transitions.
- 2 Right-click on the element and do one of the following:
 - For choice point code, select **Edit Choice Point code in Eclipse**.
 - For state entry code, select **Edit Entry code in Eclipse**. Additionally, you can select any state with entry code from the State Diagram browser.
 - For state exit code, select **Edit Exit code in Eclipse**. Additionally, you can select any state with exit code from the State Diagram browser.
 - For transition code, select **Edit Transition code in Eclipse**.

If not currently open, an editor shows the source code for the classifier (capsule or class) associated with the selected choice point, state, or transition, and the cursor moves to the beginning of the first line of code in the method following the `// {{{USR` tag.

Editing Operation Code in Eclipse

To edit operation code:

- 1 In Rational Rose RealTime:
 - Open a State Diagram for a classifier (capsule or class), and right-click on an operation in the State Diagram browser.

Or,

 - From the **Model View** tab in the global browser, right-click on an operation.
- 2 Select **Edit Operation code in Eclipse**.

If not currently open, an editor shows the source code for the operation, and the cursor moves to the beginning of the first line of code in the method following the `// {{{USR` tag.

3 Edit the code after the `// {{{USR` tag and before the `// }}}USR` tag.

Note: Overloaded operations (same name, different signatures) are supported, and the cursor will be positioned in the correct version of the operation.

Editing Capsule and Class Code in Eclipse

To edit code for a capsule or class, from the **Model View** tab in the global browser, right-click on a capsule and select **Edit Capsule code in Eclipse**, or right-click on a class and select **Edit Class code in Eclipse**.

If not currently open, an editor shows the source code for the selected capsule or class, and the cursor moves to the top of the file.

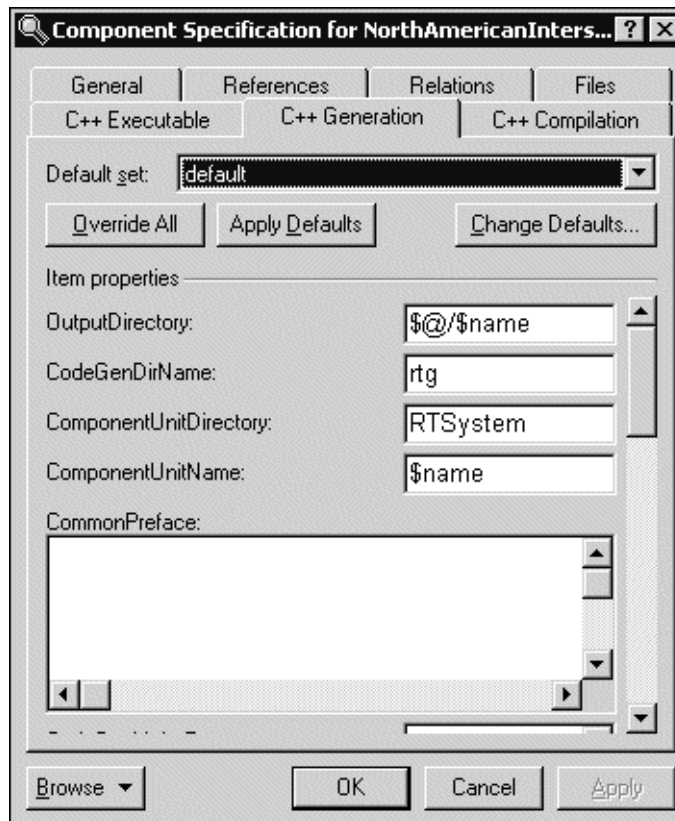
Note: While it is possible to navigate from a classifier (capsule or class) to its source, implementation, or body file by right-clicking on the classifier and selecting **Edit Capsule code in Eclipse** or **Edit Class code in Eclipse**, it is not possible to navigate to its corresponding header file. This includes all the **HeaderPreface**, **HeaderEnding**, **PublicDeclarations**, **ProtectedDeclarations** and **PrivateDeclarations** items. For a classifier, it is not possible to navigate directly to its **ImplementationPreface** or **ImplementationEnding** code segments; however, you can navigate to the top of the file.

Configuring Build Settings in Rational Rose RealTime

In Rational Rose RealTime, you want to ensure the configuration of the basic build settings before you attempt to generate code.

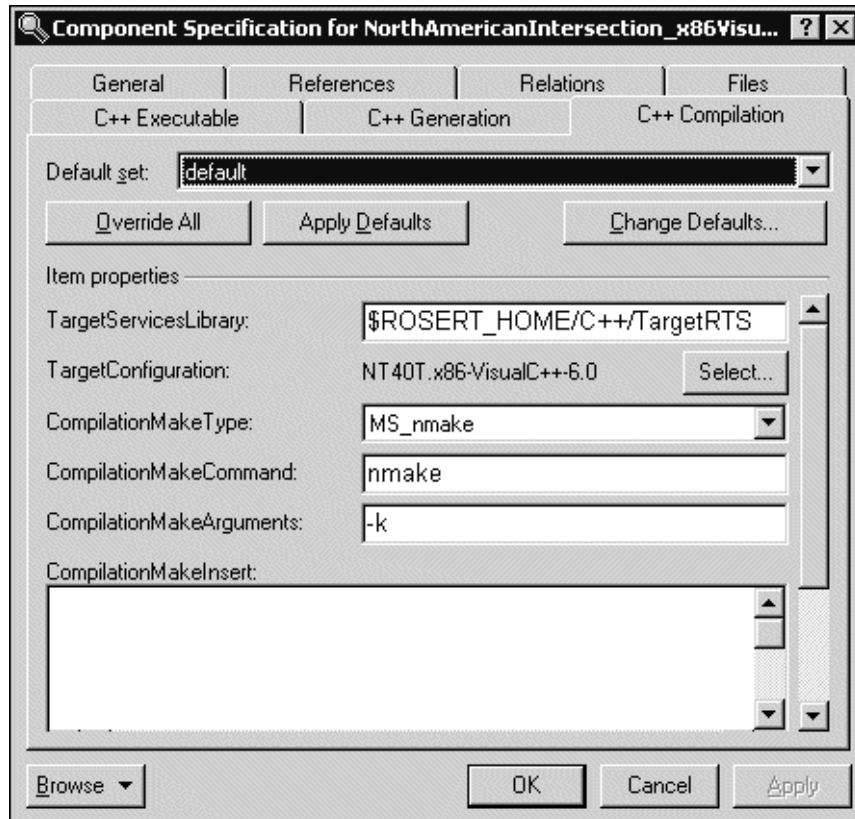
To verify your build settings:

- 1 Right-click on a component and select **Open Specification**.
- 2 Click the **C++ Generation** tab (C **Generation** tab for the C language).



- 3 In the **OutputDirectory** box, you can change the default to set another directory into which the generated files resulting from a component build are written. By default, this property is set to **\$@/\$name**, where **\$@** is the model file directory, and **\$name** is the name of the component.

- 4 Optional: Although this field can be blank, in the **CodeGenDirName** box, specify the name of the directory that the build created to store the generated source code for the component elements. This directory is generated as a subdirectory of *<output directory>/src*.
- 5 Click the **C++ Compilation** tab (C Compilation tab for the C language).



The following modifications to the **C++ Compilation** tab are for a Cygwin on a Windows configuration only. For additional information about modifying these settings, see *Example Workflow* on page 113.

- 6 In the **TargetConfiguration** box, type **NT40CygwinT.x86-cygwin-gnu-3.2**.
- 7 In the **CompilationMakeType** box, select **Gnu_make**.
- 8 In the **CompilationMakeCommand** box, type **make**.

Configuring Build Settings in Eclipse

Before a build can occur in Eclipse, there are activities which must occur in both Rational Rose RealTime and Eclipse. For additional information on the build settings in Rational Rose RealTime, see *Configuring Build Settings in Rational Rose RealTime* on page 98. To build from Eclipse, both Cygwin and Rational Rose RealTime must be installed on the computer that runs Eclipse.



Note: For a Windows configuration, after you install Cygwin, you must add `<CygwinInstallLocation>\bin` to your **Path** environment variable. Rational Rose RealTime must be installed on the computer running Eclipse, but it does not have to be running.

In Rational Rose RealTime, the settings on the **C++ Generation** tab (**C Generation** tab for the C language), are only for the computer running Rational Rose RealTime. Additionally, the settings in the **C++ Compilation** tab (**C Compilation** tab for the C language) must be configured for the computer building in Eclipse. This means that if you select a component in Rational Rose RealTime for a specific configuration (for example, a Windows-specific component) and you want to build for a different configuration (for example Linux) you must modify the compilation settings on the **C++ Generation** tab in Rational Rose RealTime. Additionally, you must modify the `$ROSE_HOME` environment variable in the **TargetServicesLibrary** box by inserting round brackets around the environment variable name. For example:

Note: `$(ROSE_HOME)/C++/TargetRTS`

For additional information about configuring these settings, see *Example Workflow* on page 113.

Building in Eclipse

In Eclipse, right-click on a project and select **Build Project**. The **C/C++ Project** perspective changes focus to the **C-Build** console view which shows the build output. Any compilation errors or warnings display in the **Tasks** list.

Note: When a project is created in Eclipse, Eclipse modifies the **C/C++ Build** settings to facilitate the building of Rational Rose RealTime code from Eclipse. Eclipse generates a new makefile, **Makefile.eric**. This makefile enables the re-use of the makefiles generated by Rational Rose RealTime.

For information about building in Eclipse, see the Eclipse online Help.

Building in Eclipse

To perform a build in Eclipse, Rational Rose RealTime must be installed on the computer that runs Eclipse, but it does not have to be running. Rational Rose RealTime must be installed because Eclipse must have access to the libraries and header files in Rational Rose RealTime. The build process requires access to the Rational Rose RealTime Runtime system because it must be able to compile for all of the targets.

Note: To use the **Build Project** option in Eclipse, you must first clear the generated files from the existing project.

To build clean code in Eclipse:

- 1 From the **C/C++ Project** view, right-click on a project (known as a component in Rational Rose RealTime).
- 2 Click **Delete**, and select the **Also delete contents** option.
- 3 Click **OK**.
- 4 In Rational Rose RealTime, right-click on the component, and click **Build > Build**.
- 5 Select **Generate**.

Eclipse generates a clean set of project files for the selected Rational Rose RealTime component.

Note: When Eclipse creates a project from building a Rational Rose RealTime component, Eclipse changes to the C/C++ Development perspective, and it changes the **make** build command to use the generated code and the generated **makefile** (**makefile.eric**). Now, when you perform a build in Eclipse, the build uses the **make** file **makefile.eric**.

Now, you are ready to build in Eclipse.

- 6 In Eclipse, right-click to select the project for the component you generated in Rational Rose RealTime.
- 7 Select **Build Project**.

Compilation starts, using the special makefile **Makefile.eric**. While the code compiles, you can view the output in the **C-Build** output window. If there are any compilation errors or warnings, they appear in the **Tasks** list.

Note: If there are build errors in Eclipse caused by syntax errors in your code (the code between the `/{USR` and `// }USR` tags), you can easily navigate to that corresponding element in Rational Rose RealTime. For more information on navigating from build errors in Eclipse, see *Navigating to Build Errors* on page 103.



Note: In Rational Rose RealTime, you will need to modify the value for the **TargetServicesLibrary** on the **C++ Compilation** tab for a component if you generate code on one computer and build it on another, and the computers are on different configurations. For example, if you generate a Rational Rose RealTime model on a Windows computer, and attempt to build it on a computer with a Linux configuration, the build will not be successful. To successfully build when doing cross-platform development, open the **Component Specification** dialog and click the **C++ compilation** tab. Typically, the value for **TargetServicesLibrary** is `$ROSSERT_HOME/C++/TargetRTS`. The code generator will generate a **make** file and expand the `$ROSSERT_HOME` environment variable to the actual value (for example, `C:\Program Files\Rational\Rose Realtime`). This value may not be valid on the computer doing the build. You must modify the environment variable so that it is not expanded by inserting round brackets around the environment variable name. For example:

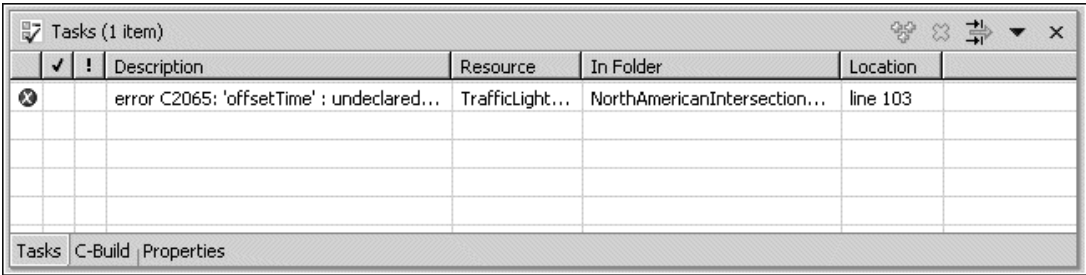
`$(ROSSERT_HOME)/C++/TargetRTS`

Navigating to Build Errors

After you build a component in Rational Rose RealTime or Eclipse, you may encounter compilation errors. If there are errors, you can easily navigate to that code and investigate.

To navigate from build errors in Rational Rose RealTime to the corresponding code in Eclipse:

- 1 In Rational Rose RealTime, right-click on any build error in the **Build Errors** list in the **Output** window.



- 2 Select **Edit Code in Eclipse**.

The corresponding source file opens in an Eclipse editor, with the cursor positioned on the line where the error occurred.

To navigate from build errors in Eclipse to the corresponding element in Rational Rose RealTime for items on the State Diagram:

- 1 In Eclipse, build a project. For instructions on building a project see *Building in Eclipse* on page 101.

If the compilation process encountered any errors, they appear in the **Task** list.

- 2 From the **Task** list, right-click on a build error and select **Navigate to Rose RealTime**.

If not currently open, a diagram for the capsule or class corresponding to the source file opens, and that model element is selected.

Navigating From Rational Rose RealTime to Eclipse

In Rational Rose RealTime, you can navigate from classifiers (capsule and class), user-defined operations, choice points, state (entry and exit), and transitions.

Note: It is not possible to navigate to any source, header, or generated code for a component. It is also not possible to navigate directly to any documentation generated in the source code.

You can easily navigate methods in Eclipse in the following ways:

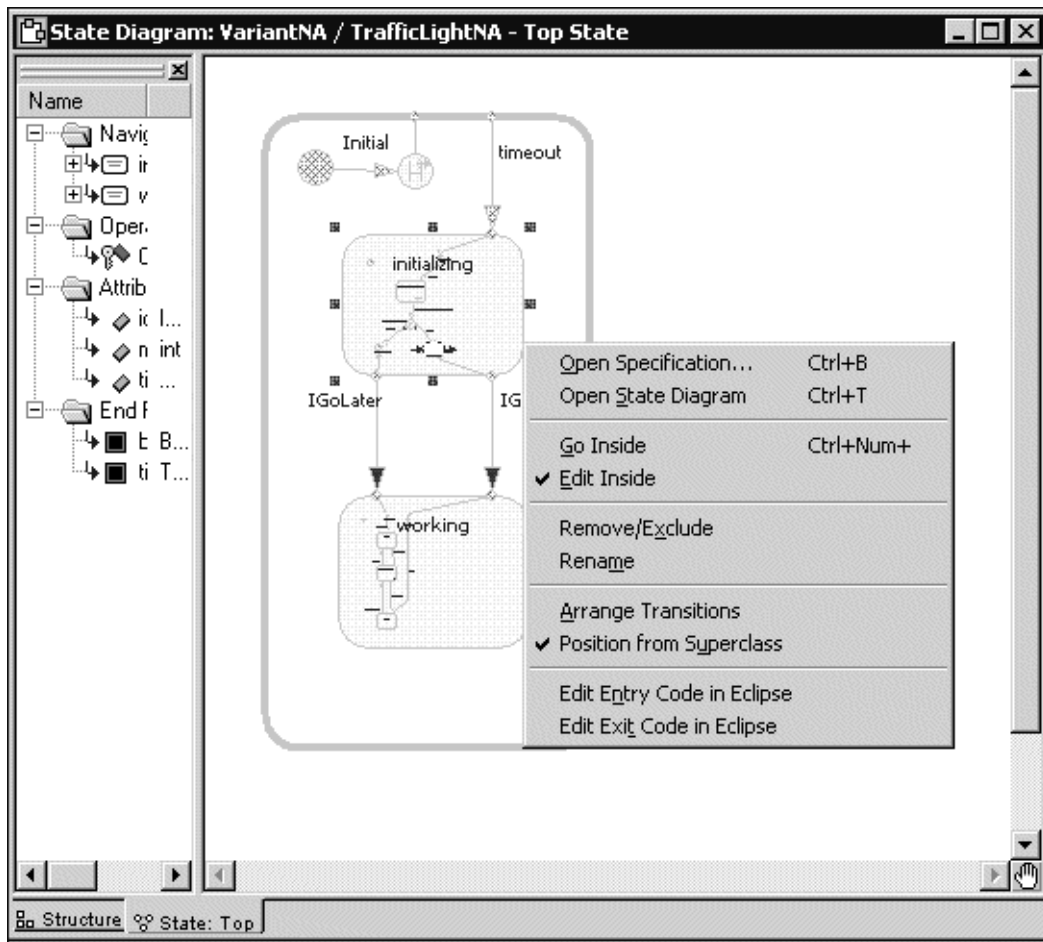
- *Navigating from a State Diagram* on page 104
- *Navigating from a State Diagram or State Diagram Browser* on page 106
- *Navigating from the Model View Tab in the Browser* on page 107

Navigating from a State Diagram

From Rational Rose RealTime, you can navigate to Eclipse by selecting a choice point, state, or transition in the State Diagram dialog box.

Right-click on an element from a State Diagram (a transition, state, or choice point) that currently has user code generated for that element, and then select **Edit <element> code in Eclipse**.

Figure 8 Navigating from a State Diagram



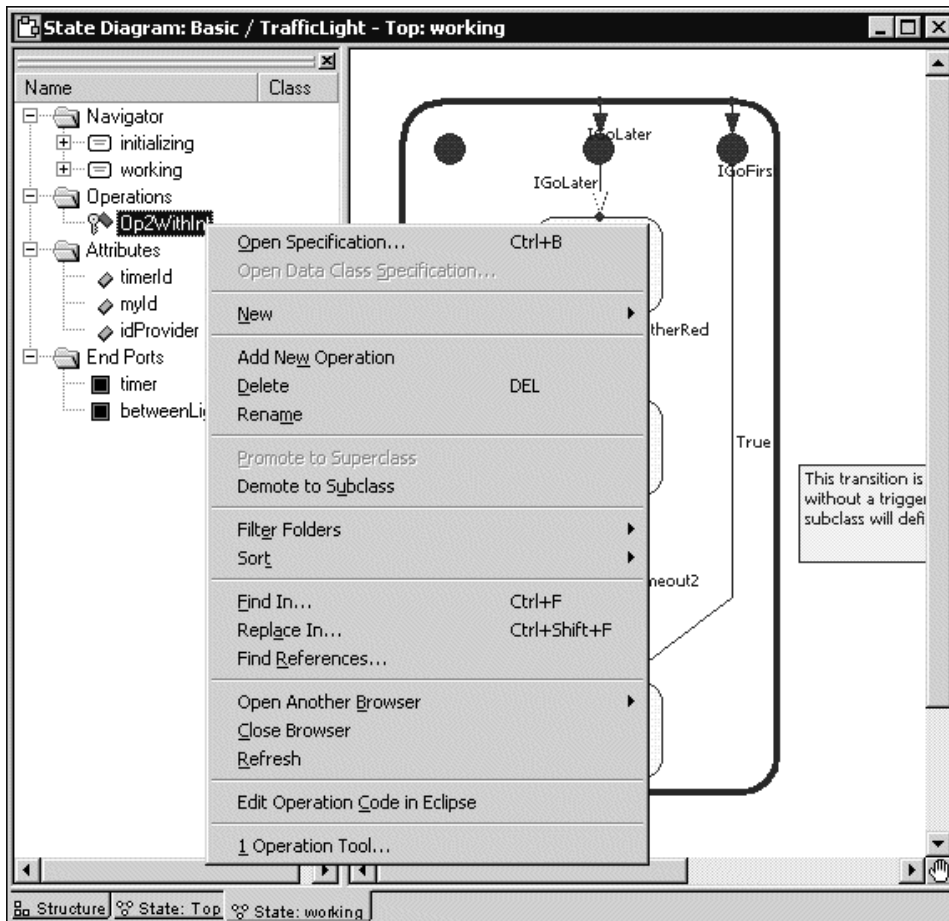
The source file for the corresponding element opens in an Eclipse editor, with the cursor positioned on the first line of the user code for the model element.

Note: If there is no existing code to edit for the selected element, the corresponding **Edit <element> code in Eclipse** menu item is grayed out. If the **CodeSyncEnabled** option on the **C++ Generation** tab is disabled in Rational Rose RealTime, **USR** tags are not created when the component is generated. Although there are no **USR** tags, you can navigate to that corresponding element in Eclipse. Instead of positioning the cursor after the `// {{{USR` tag and before the `// }}}USR` tag, the cursor is positioned at the top of the file.

Navigating from a State Diagram or State Diagram Browser

In a State Diagram browser, you can navigate to Eclipse by selecting a user-defined operation. In Rational Rose RealTime, right-click on any operation and select **Edit Operation Code in Eclipse**, **Edit Entry Code in Eclipse**, **Edit Entry Code in Eclipse**, or **Edit Choice Point Code in Eclipse**.

Figure 9 Rational Rose RealTime - Navigation to Eclipse from an Operation



The source file opens in an Eclipse editor, with the cursor positioned on the first line of the operation.

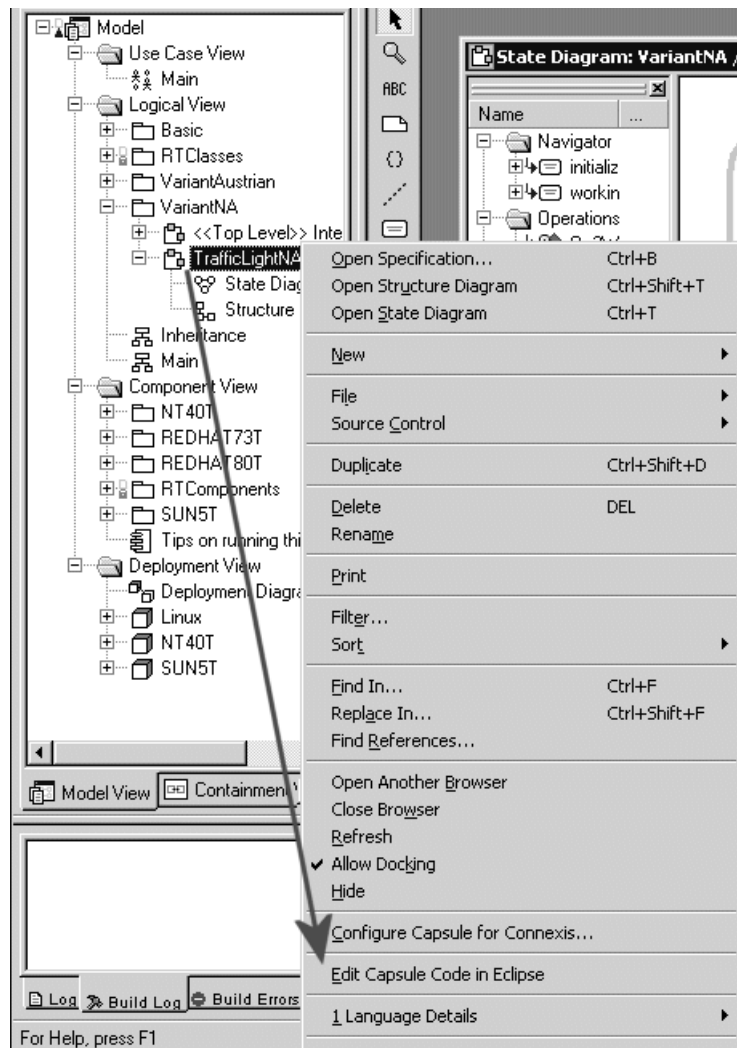
Note: Overloaded operations are supported.

Navigating from the Model View Tab in the Browser

On the **Model View** tab, you can navigate to Eclipse by selecting a user-defined operation, class, or capsule. To navigate from an operation, in Rational Rose RealTime, right-click on any operation and select **Edit Operation Code in Eclipse**. The source file for the selected operation opens in an Eclipse editor, with the cursor positioned on the first line of the operation. Overloaded operations are supported.

To navigate from a class or capsule, right-click any class or capsule and select **Edit <classifier> Code in Eclipse**.

Figure 10 Rational Rose RealTime - Navigating to Eclipse from a Capsule



The source file for the selected classifier opens in an Eclipse editor, with the cursor positioned on the first line.

Note: If the **CodeSyncEnabled** option on the **C++ Generation** tab is disabled in Rational Rose RealTime, **USR** tags are not created when the component is generated. Although there are no **USR** tags, you can navigate to that corresponding element in Eclipse. Instead of positioning the cursor after the `// {{{USR` tag and before the `// }}}USR` tag, the cursor is positioned at the top of the file.

Navigating from Eclipse to Rational Rose RealTime

From user-defined operations, choice points, state (entry and exit), and transition methods in Eclipse, you can easily navigate to the corresponding element in Rational Rose RealTime.

To navigate from an operation, choice point, state (entry and exit), or transition method:

- 1 In Eclipse, click **Window > Preferences**.
- 2 Click **Rose RealTime Integration** and select the **Features** tab.
- 3 Ensure that the **Select elements from C/C++ Projects View** option is selected.
- 4 Optional: To automatically select an element from the Rational Rose RealTime browser, select the **Also select element in browsers** option.
- 5 In the **C/C++ Projects** view, navigate through the generated source code for any capsule or class with an operation, choice point, state (entry and exit), or transition.

The generated methods are in the directory you specified in the **CodeGenDirName** box on the **C++ Generation** tab specified in Rational Rose RealTime.

- 6 Select any method for a user-defined operation, choice point, state (entry and exit), or transition.



Note: You can recognize valid methods by the following patterns:

- For a choice point method, the method name pattern is:
 - In the **C/C++ Projects** view, `<class>::choicePoint<number>_<name>`
 - In the **Outline** view, `<class>::choicePoint<number>_<name>(<arguments>)`
- For a state entry method, the method name pattern is:
 - In the **C/C++ Projects** view, `<class>::enter<number>_<name>(<arguments>)`
 - In the **Outline** view, `<class>::enter<number>_<name>(<arguments>)`
- For a state exit method, the method name pattern is:
 - In the **C/C++ Projects** view, `<class>::exit<number>_<name>`
 - In the **Outline** view, `<class>::exit<number>_<name>(<arguments>)`
- For a transition method, the method name pattern is:
 - In the **C/C++ Projects** view, `<class>::transition<number>_<name>`
 - In the **Outline** view, `<class>::transition<number>_<name>(<arguments>)`



Note: Because the names of operations are specified by the user, there is no method name pattern specified.

If not currently open, the diagram containing the classifier (capsule or class) associated with the source file for the selected operation, choice point, state (entry and exit), or transition method displays in Rational Rose RealTime. The corresponding operation, choice point, state (entry or exit), or transition is also selected in the main browser.

Getting Started

The following example will help you get started with the Rational Rose RealTime and Eclipse integration. This example guides you through the basic connectivity features of the integration.

To start using the Eclipse and Rational Rose RealTime integration:

- 1 Start Rational Rose RealTime.
- 2 In Rational Rose RealTime, do one of the following:
 - Create a new model and configure its Eclipse Integration settings in Rational Rose RealTime (**Tools > Eclipse Integration Settings**). For information on configuring the Eclipse Integration settings, see the topic *Configuring Preferences in Rational Rose RealTime* on page 89.

Note: When you click **File > New** in Rational Rose RealTime, the Rational Rose RealTime settings are restored to their default values.
 - Load a model that has not had a previous connection to Eclipse, and modify its Eclipse Integration settings in Rational Rose RealTime (**Tools > Eclipse Integration Settings**).
 - Load a model that has had a previously established connection to Eclipse.

Note: When you load a model that has had a previous connection to Eclipse, that connection will automatically be established after the model loads. Additionally, after a successful connection is established between Rational Rose RealTime and Eclipse, the previously built component in Rational Rose RealTime displays as a project in Eclipse, along with any generated source files.
- 3 In Rational Rose RealTime, ensure that the settings on the **Configuration** tab and **Generation** tab for the selected component are set correctly. For additional information about configuring these settings in Rational Rose RealTime, see *Configuring Build Settings in Rational Rose RealTime* on page 98, and *Configuring Build Settings in Eclipse* on page 100.
- 4 Start Eclipse.
- 5 Ensure that the Rational Rose RealTime Integration settings allow Rational Rose RealTime and Eclipse to connect. For information on configuring the Eclipse Integration settings, see the topic *Configuring Preferences in Eclipse* on page 81.

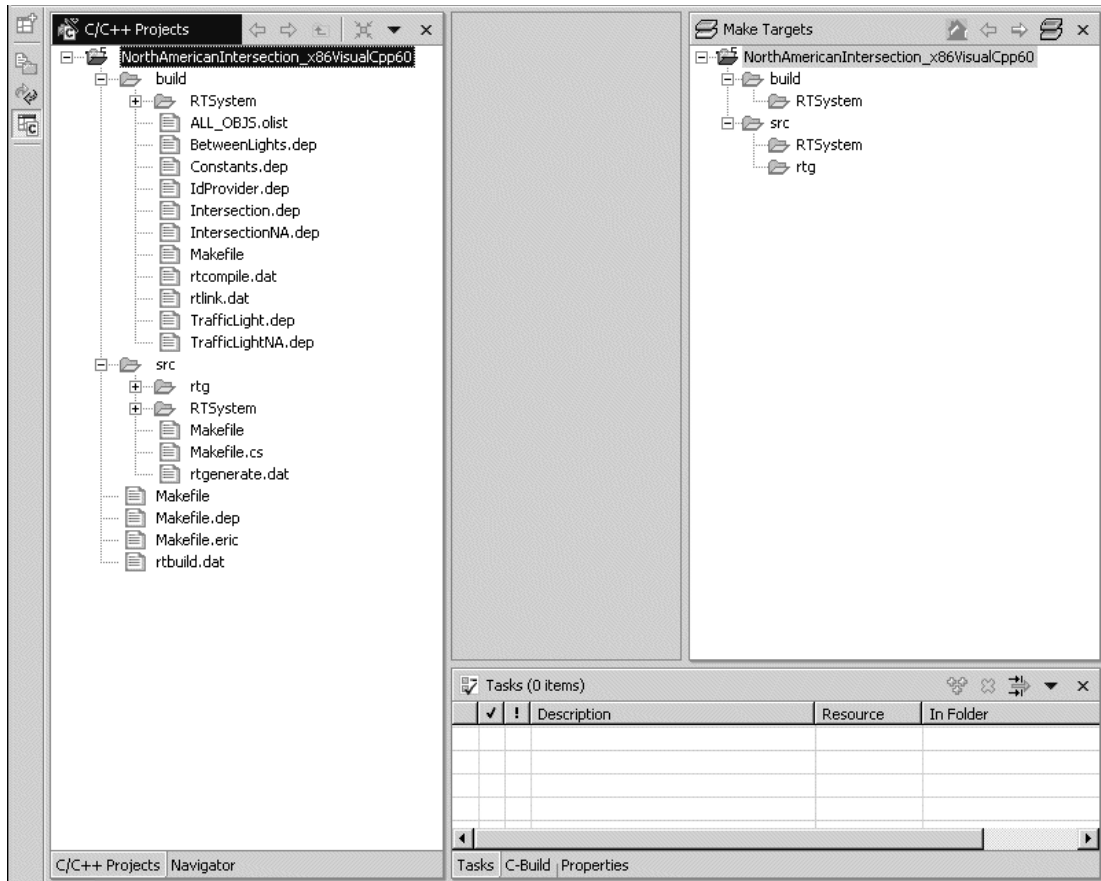
In Eclipse, if not currently open, the **C/C++ Development** perspective opens and the selected component for the model displays as a C/C++ project.

6 Do one of the following:

- If the model is new and has not been built previously in Rational Rose RealTime, click **Build > Build**, and then select **Generate**. The updates and any new generated code files display in the Eclipse project, and in the common file storage location.
- For a model that was built previously, you can click **Build > Clean**, and then click **Build > Build**, and select **Generate**. All the generated code files are removed from the project in Eclipse, and from the common file storage location, and then new files are generated.

Note: Allow the code generation process to complete. Ensure that the last entry on the **Log** tab in the **Output** window indicates a successful build.

After the generation process completes in Rational Rose RealTime, the corresponding project in Eclipse refreshes to show the generated code.



Now, you can navigate to and from Rational Rose RealTime and Eclipse by selecting user-defined operations, choice points, state (entry and exit), transitions, classes and capsules in Rational Rose RealTime, or by selecting user-defined operations, choice point, state (entry and exit), and transition methods in Eclipse.



Note: Because Eclipse and Rational Rose RealTime share a common file system, you must use caution when modifying code and building. Do not modify the same method in Rational Rose RealTime and Eclipse. Saving in one application will overwrite any change you made in the other application. We strongly recommend that you save and build you changes as often as possible.

Example Workflow

The following example will help you get started with the Rational Rose RealTime and Eclipse integration. This example guides you through the features of the integration using the **TrafficLights.rtmdl** model file included with Rational Rose RealTime.

This example, makes use of the following assumptions:

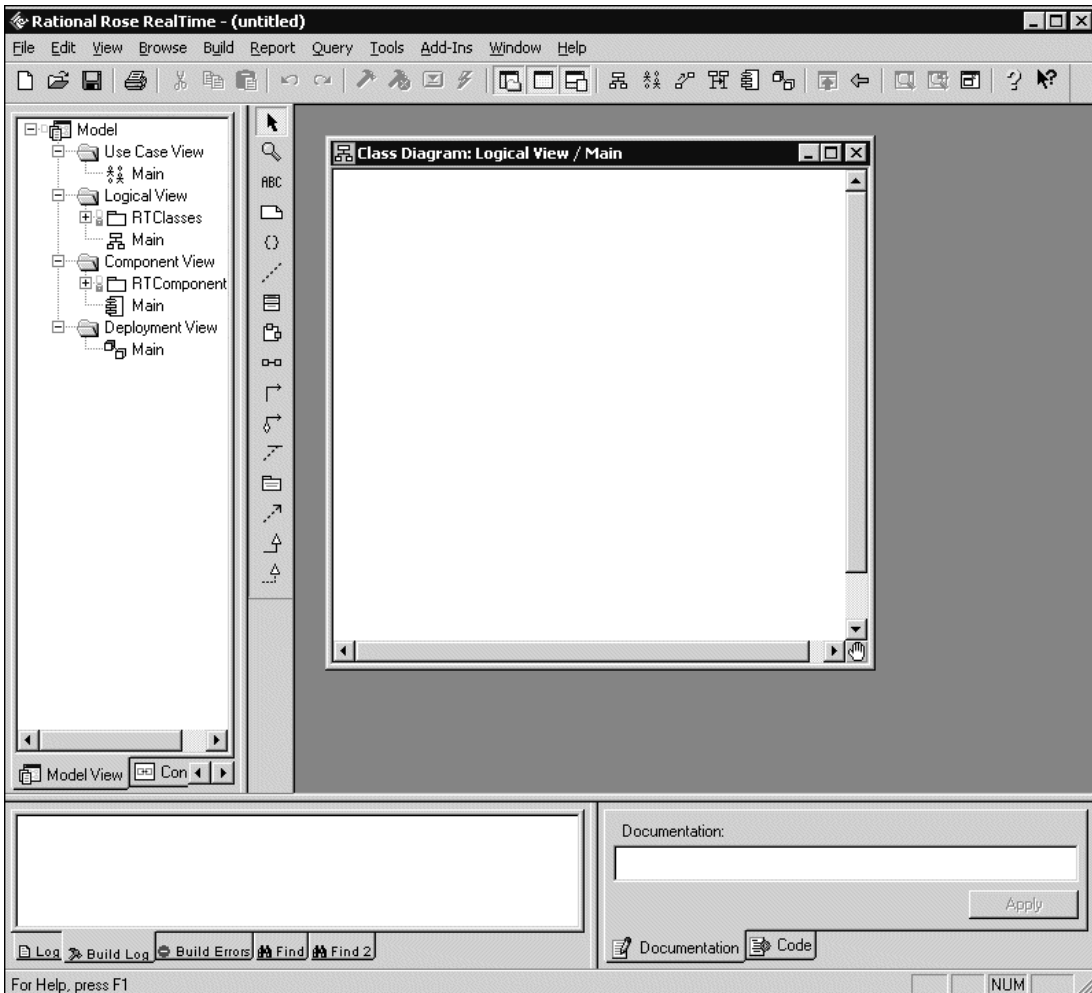
- Rational Rose RealTime is running on a Windows configuration.
- Eclipse is running a UNIX configuration.
- The computer running Rational Rose RealTime is the listener, and Eclipse is the connector.
- The **TrafficLights.rtmdl** model has not been built previously or used in an Eclipse and Rational Rose RealTime integration.

To observe a simple example of the Eclipse and Rational Rose RealTime Integration:

- 1 Start Rational Rose RealTime.
- 2 Click **File > New**.
- 3 Click the **RTC++** framework.

The default setting for a C or C++ model is to connect to Eclipse.

When Rational Rose RealTime starts, it loads a blank model. The default Rational Rose RealTime integration setting for a blank model is disabled.



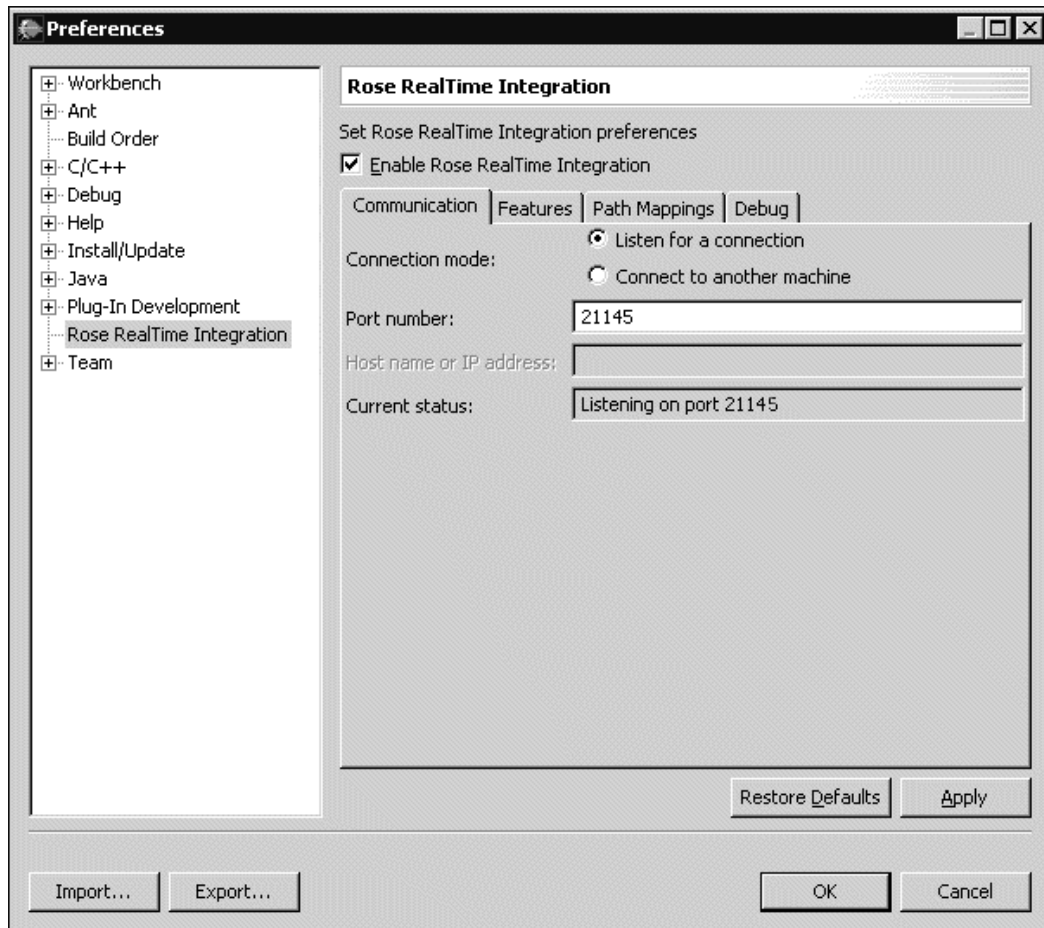
4 Start Eclipse.

When Eclipse starts, if the Rational Rose RealTime Integration is enabled, it automatically attempts to establish a connection. Because this example has Eclipse and Rational Rose RealTime on two different computers with different configurations (UNIX and Windows), the default settings must be modified to establish communication, and to specify path mapping for both configurations.

5 In Eclipse, click **Window > Preferences**.

6 Click **Rose RealTime Integration** and click the **Communication** tab.

By default, Eclipse is the listener and the port number is 21145.



7 Select **Connect to another machine**.

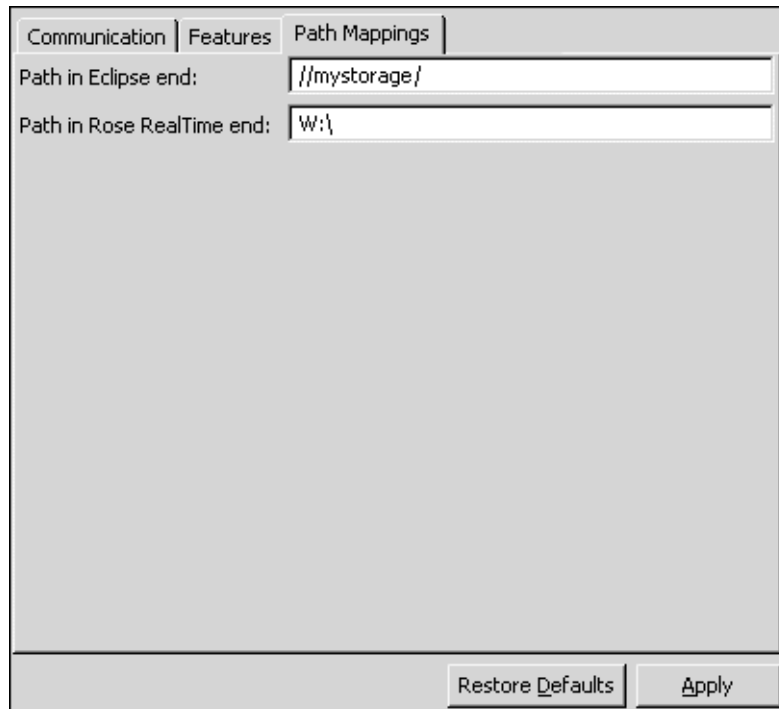
8 Specify the host name or IP address of the computer running Rational Rose RealTime.

In this example, Rational Rose RealTime is on a Windows XP computer and the model files are stored in the directory `\user\data\` on a network drive mapped to `S:`. The fully qualified path to the model file is `W:\user\data\mymodel.rtm1`. Eclipse is running on a computer with a UNIX configuration and the fully qualified file location is `//mystorage/user/data/mymodel.rtm1`. Eclipse and Rational Rose RealTime need to reference the same location.

9 Click the **Path Mappings** tab.

10 Set **Path in Eclipse end** to `//mystorage/`.

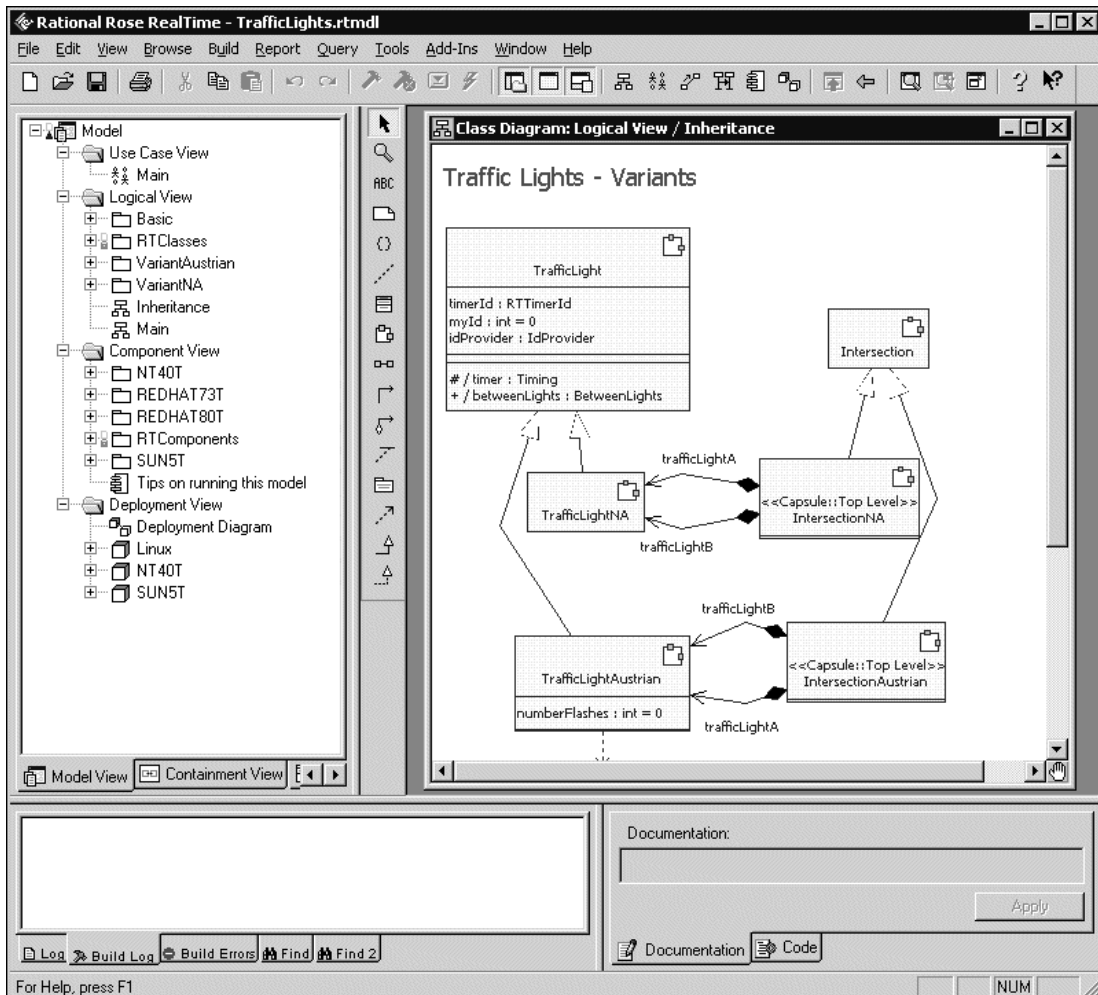
11 Set **Path in Rational Rose RealTime end** to `W:\`.



Note: You can use "/" or "\" interchangeably when specifying path mappings.

12 Click **Apply**.

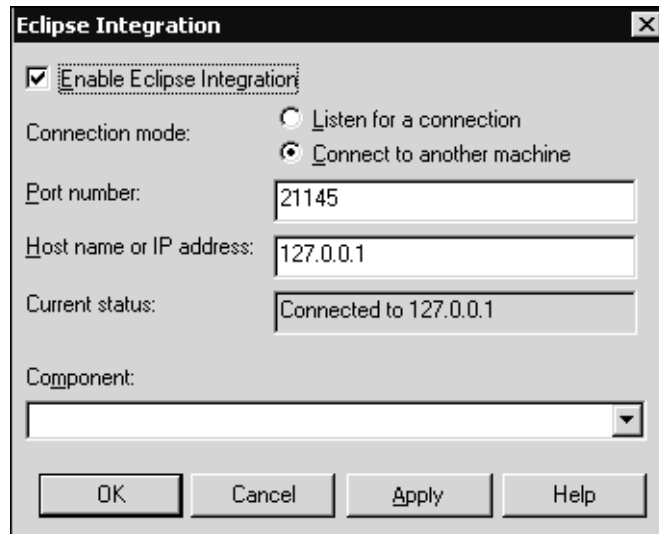
- 13 In Rational Rose RealTime, open the sample model file
<\$ROSERT_HOME>/examples/models/C++/TrafficLights.rtm1.



Next, you will create a duplicate

- 14 Right-click the **NorthAmericanIntersection_sparcgnu281** component from the **SUN5T Component View** package in the **Model View** tab in the main browser and click **Duplicate**.
- 15 Right-click on the new component and click **Rename**.
- 16 Type **NorthAmericanIntersection_sparcgnu281example**.

17 Click **Tools > Eclipse Integration Settings**.



By default, Rational Rose RealTime is the connector.

18 In the **Connection mode** area, select **Connect to another machine**.

19 In the **Host name or IP address** box, specify the host name or IP address of the UNIX computer running Eclipse.

20 Click **Apply**.

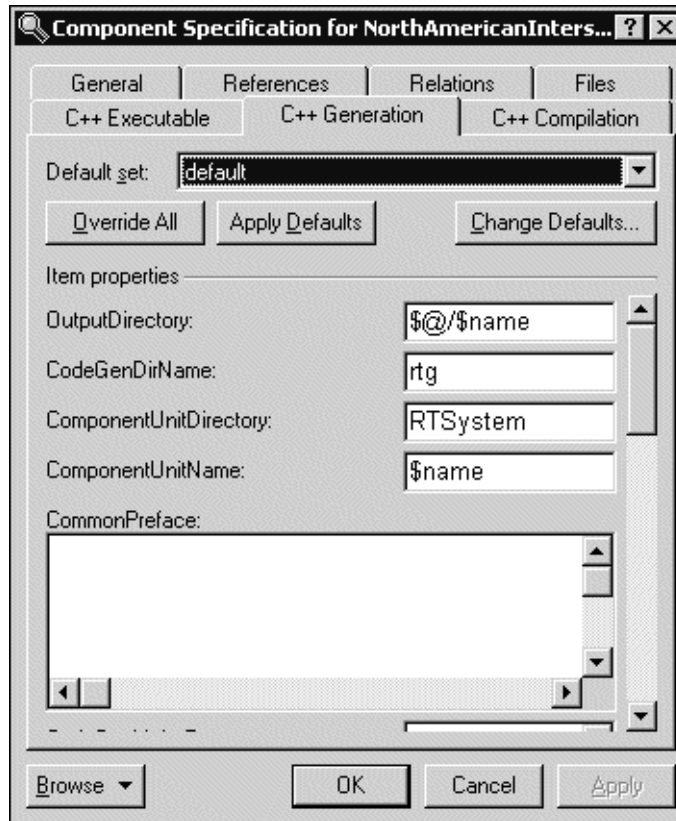
21 In the **Current Status** box, verify that the connection to Eclipse is established. For information about troubleshooting connectivity errors, see *Troubleshooting* on page 131.

22 In the **Component** drop-down list, select the component called **NorthAmericanIntersection_sparcgnu281example**.

23 Click **OK**.

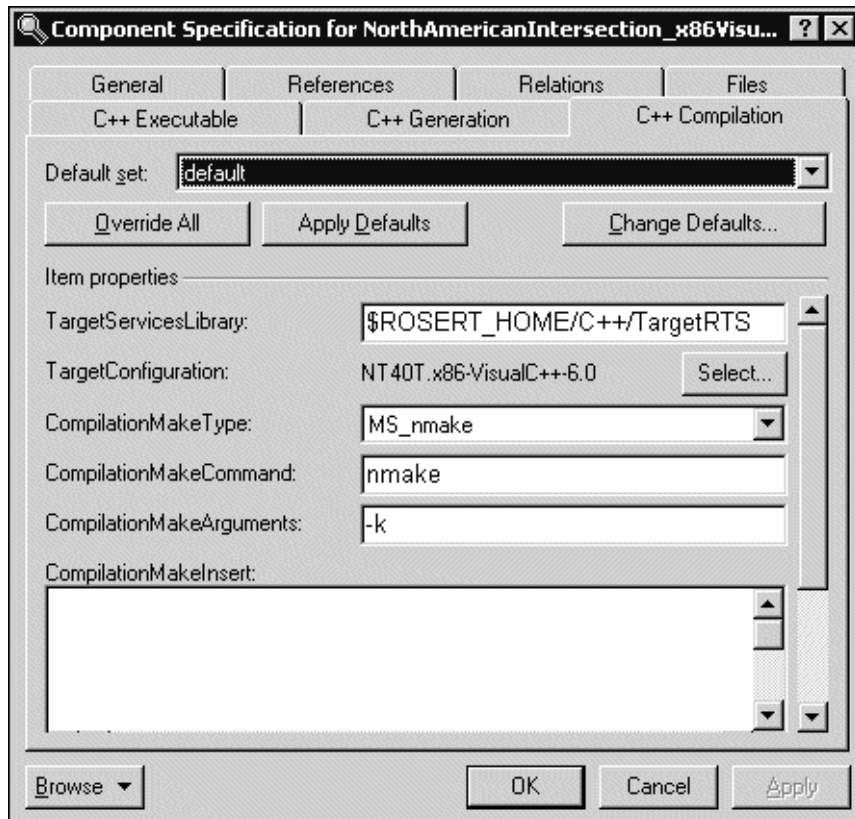
24 Right-click the **NorthAmericanIntersection_sparcgnu281example** component from the SUN5T **Component View** package in the **Model View** tab in the main browser and select **Open Specification**.

25 Click the **C++ Generation** tab.



- 26 In the **OutputDirectory** box, you can change the default to set another directory into which the generated files resulting from a component build are written. By default, this property is set to **\$@/\$name**, where **\$@** is the model file directory, and **\$name** is the name of the component.
- 27 Optional. In the **CodeGenDirName** box, specify the name for the directory to store the generated source code for the component elements. This directory is generated as a subdirectory of *<output directory>/src*.

28 Click the **C++ Compilation** tab.



Earlier, you opened the Windows component for which the settings on the **C++ Generation** tab are correct. Because your compilation settings are going to be different, you will need to modify the settings on the **C++ Compilation** tab.

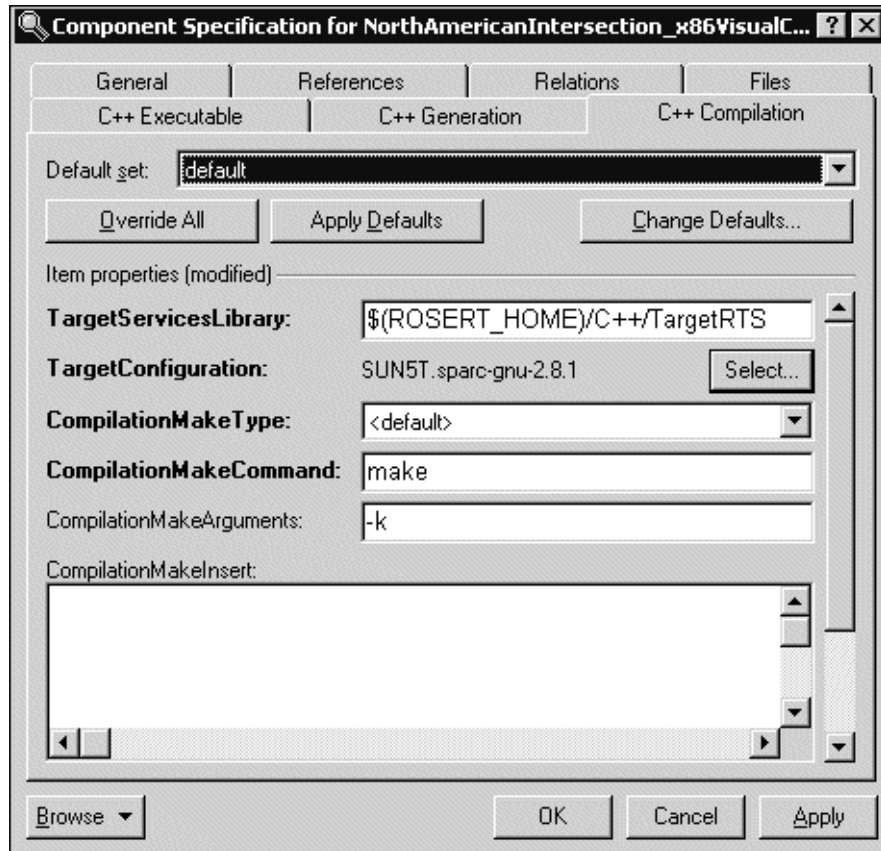
29 In the **TargetServicesLibrary** box, add round brackets around the **ROSERT_HOME** environment variable.

30 In the **CompilationMakeCommand** box, type **make**.

31 In the **CompilationMakeType** box, type **<default>**.

32 In the **TargetConfiguration** box, select **SUN5T.sparc-gnu-2.8.1**.

Your C++ **Compilation** tab should look like the following:



33 Click **OK**.

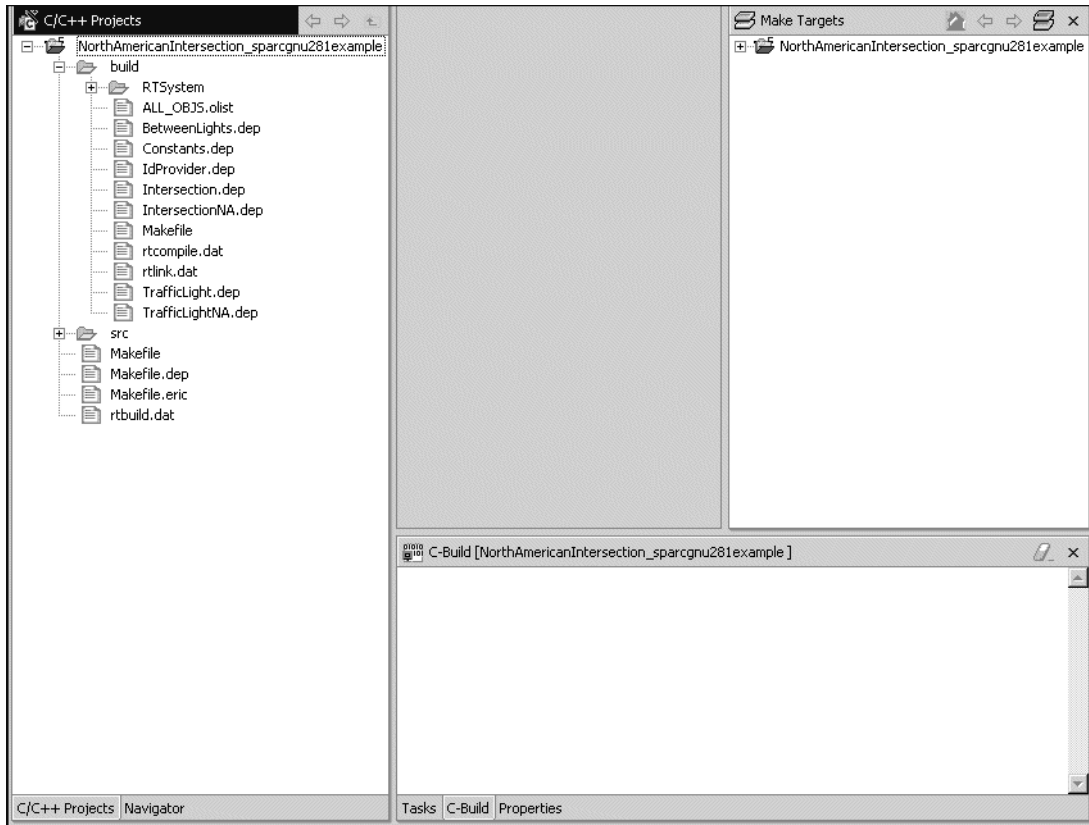
34 In Rational Rose RealTime, right-click the **NorthAmericanIntersection_sparcgnu281example** component, click **Build > Clean**, and then click **OK**.

All previously generated code files for the **TrafficLights.rtmidl** model (if any) are removed from the output directory (our common file storage location).

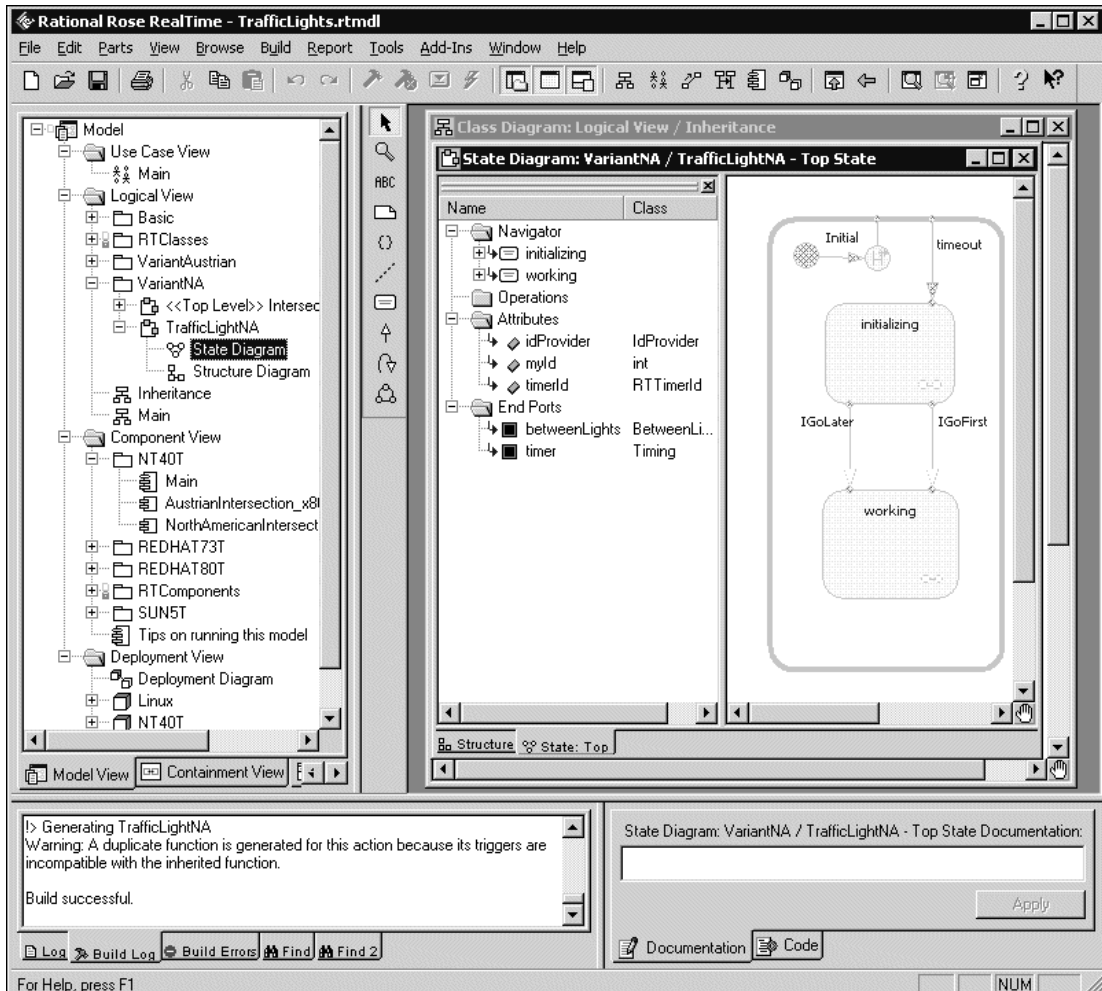
35 Click **Build > Build**, click **Generate**, and then click **OK**.

Note: While Rational Rose RealTime builds the selected component, there will be a delay in Eclipse proportionate to the amount of code being generated.

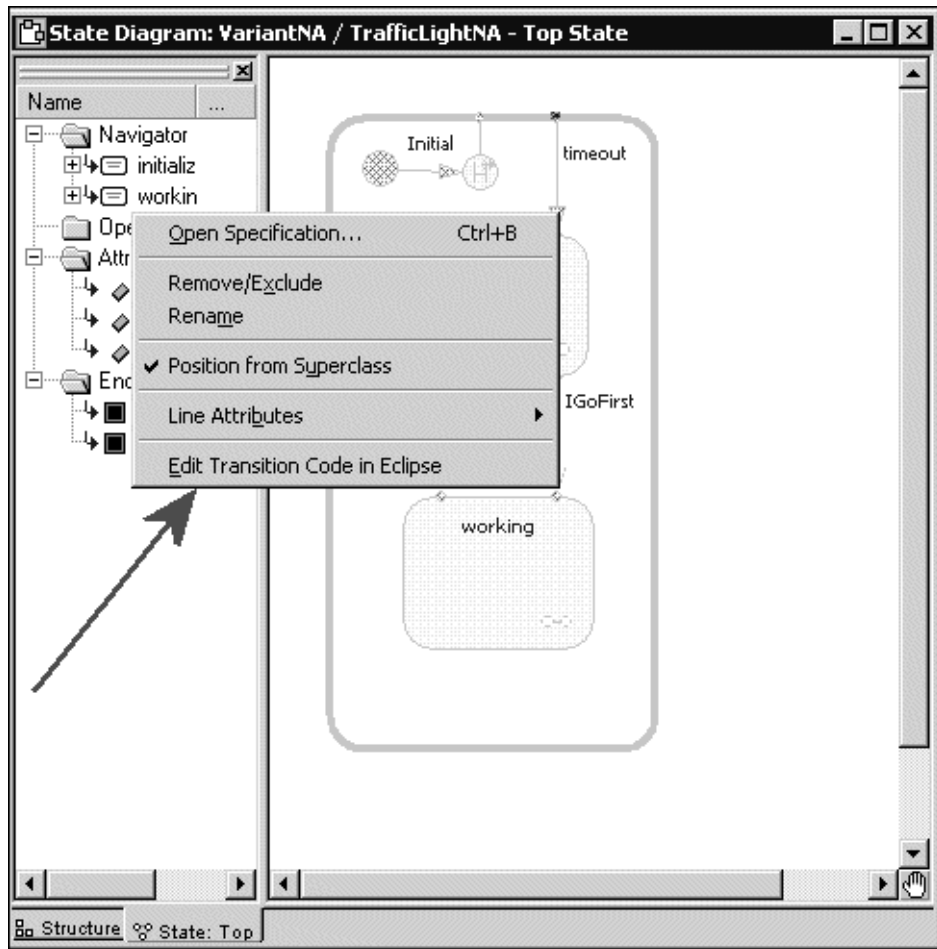
The updated files list, and any new generated code files, display in the Eclipse project and the common file storage location.



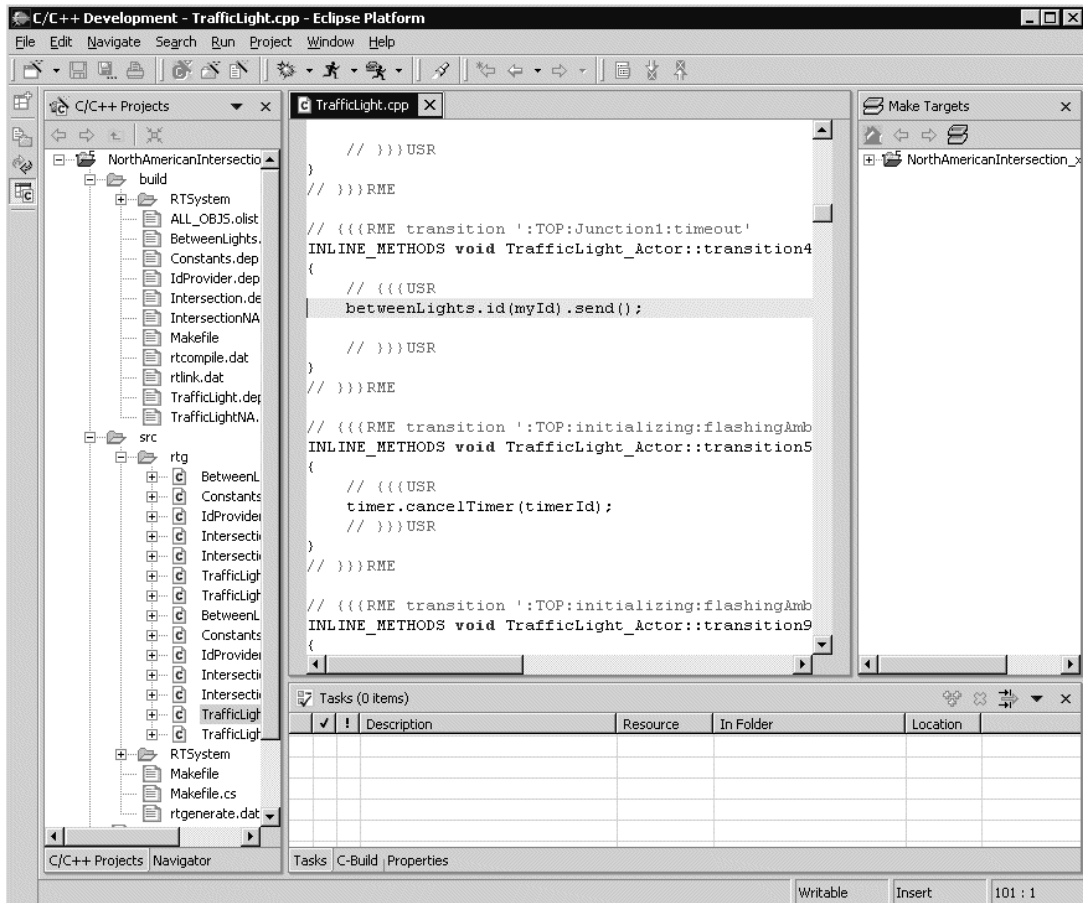
- 36 In Rational Rose RealTime, on the **Model View** tab in the main browser, open the State Diagram for the capsule **TrafficLightNA** from the **VariantNA** package.



37 Right-click the **timeout** transition line and select **Edit Transition Code in Eclipse**.

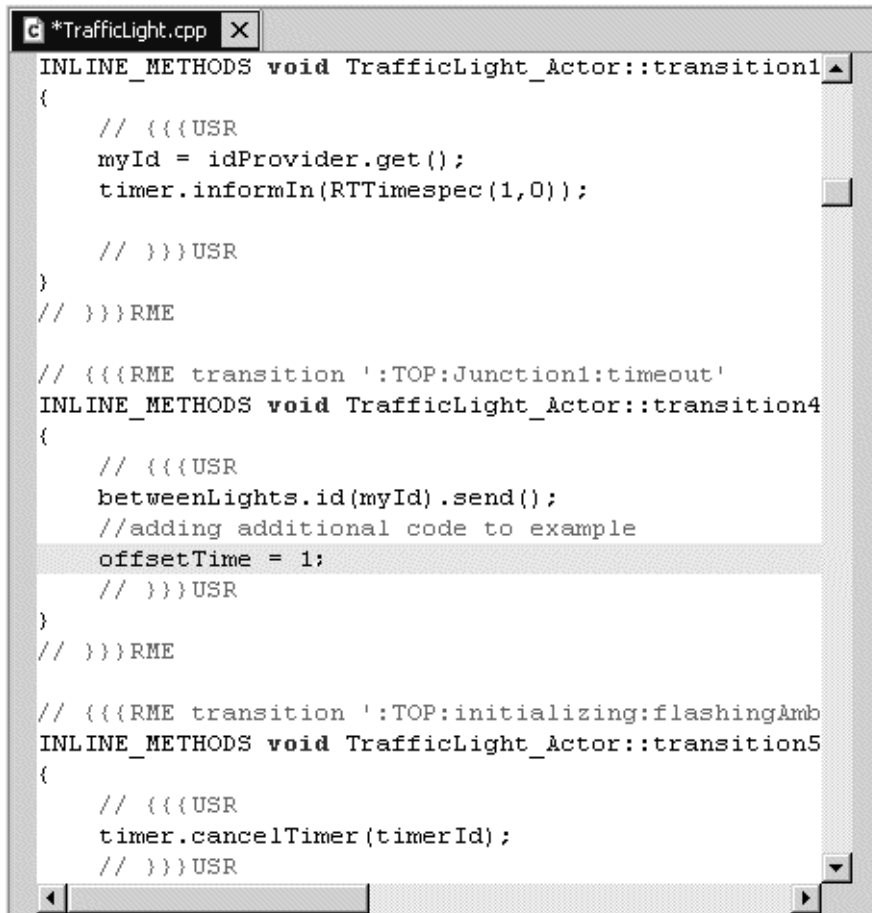


In Eclipse, the source file for the corresponding transition opens in an Eclipse editor, with the cursor positioned on the first line of the user code after the **USR** tag for the model element.



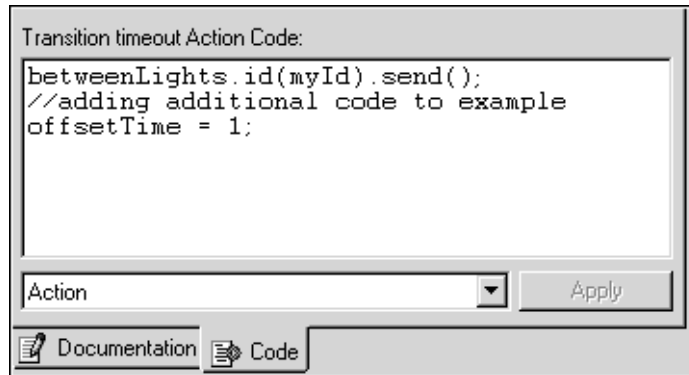
38 Between the **USR** tags, the following comment and code is added.

```
//adding additional code to example  
offsetTime = 1;
```



39 In Eclipse, click **File > Save**.

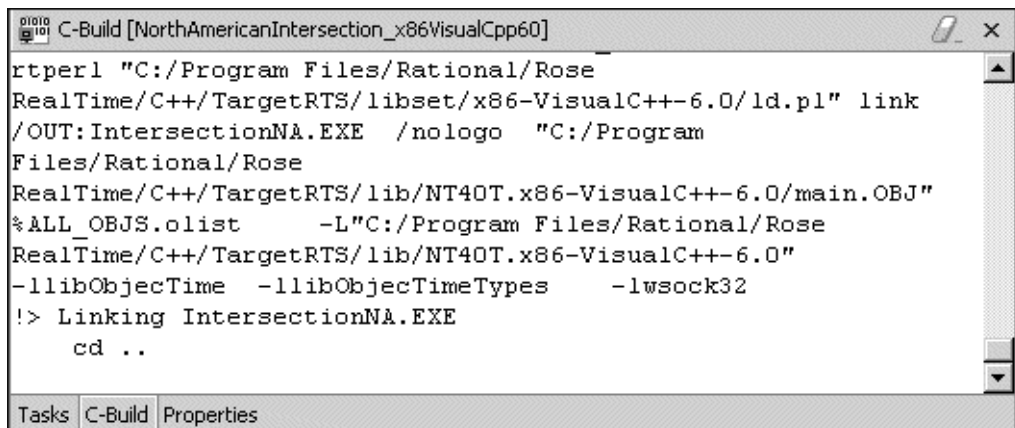
Eclipse saves the file containing the changes and a codesync is performed with Rational Rose RealTime. Although the last line of the **Build Log** tab in the **Output** window indicates a successful build, only a codesync was performed. If you select the **timeout** transition in Rational Rose RealTime, the **Code** tab in the **Documentation** window includes the code added earlier.



Next, you will build the project.

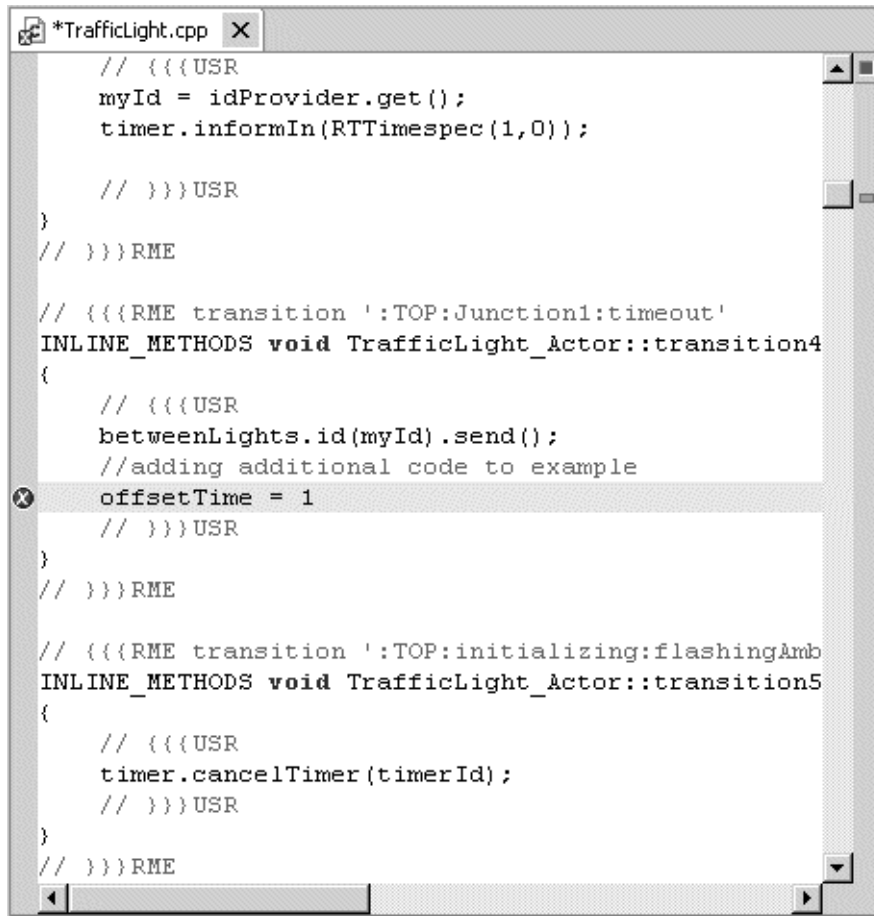
- 40 In Eclipse, right-click the **NorthAmericanIntersection_sparcgnu281** project and click **File > Build Project**.

Eclipse generates the executable **IntersectionNA.EXE** for the selected project.



Next, you will introduce an error in the code so that you can navigate to that error.

- 41 Remove the semicolon at the end of the code for **offsetTime = 1**;



```
// {{{USR
myId = idProvider.get();
timer.informIn(RTTimespec(1,0));

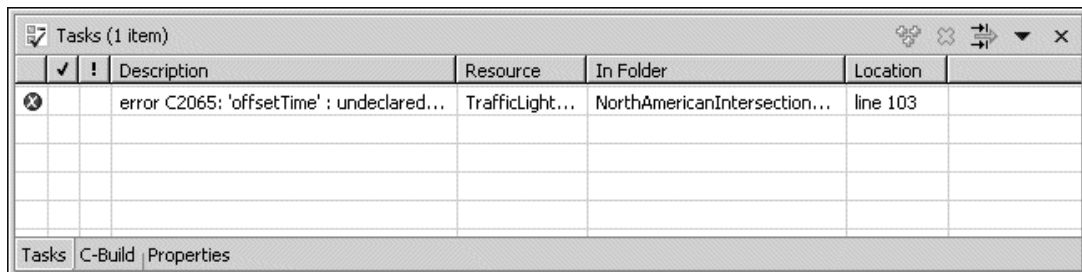
// }}}USR
}
// }}}RME

// {{{RME transition ':TOP:Junction1:timeout'
INLINE_METHODS void TrafficLight_Actor::transition4
{
    // {{{USR
    betweenLights.id(myId).send();
    //adding additional code to example
    offsetTime = 1
    // }}}USR
}
// }}}RME

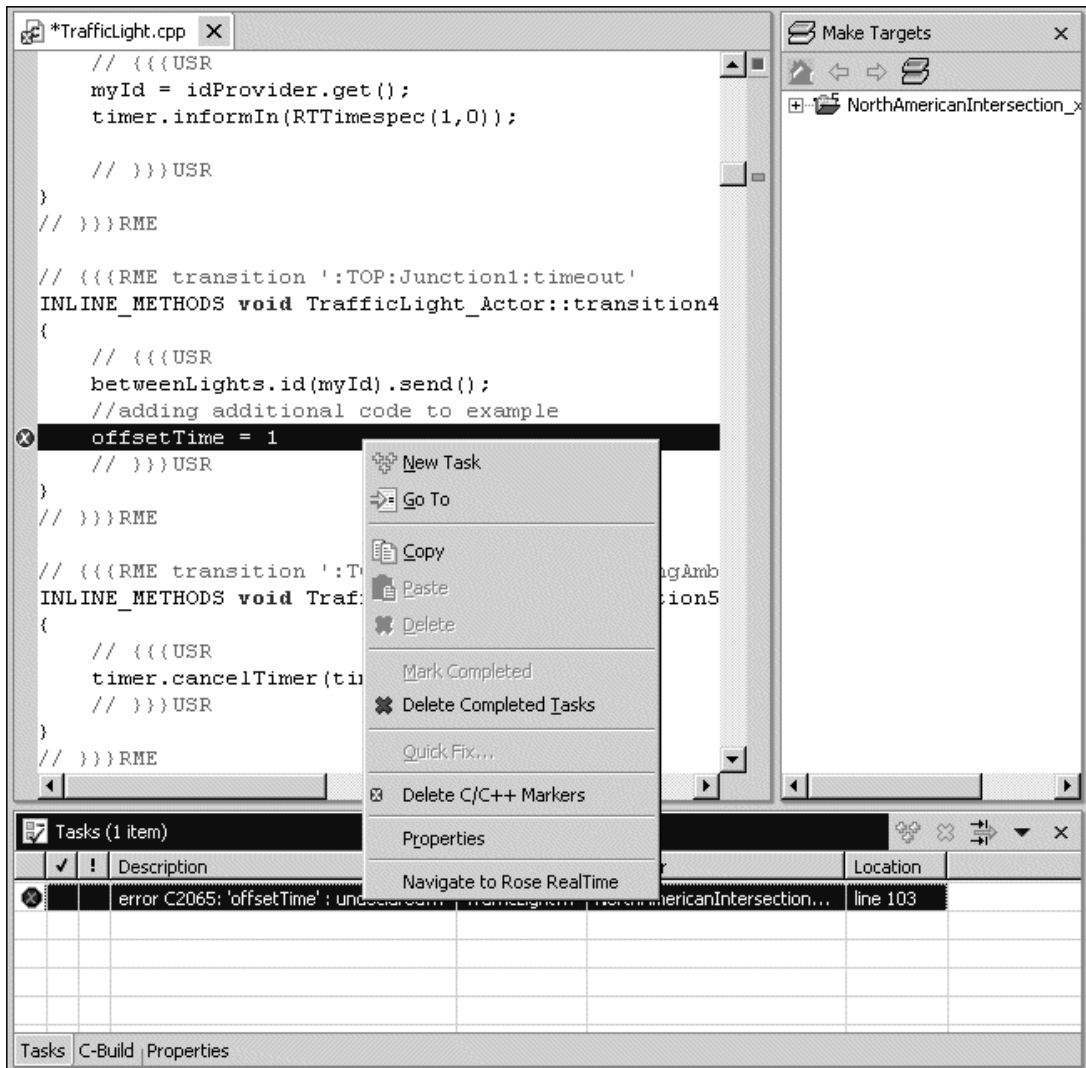
// {{{RME transition ':TOP:initializing:flashingAmb
INLINE_METHODS void TrafficLight_Actor::transition5
{
    // {{{USR
    timer.cancelTimer(timerId);
    // }}}USR
}
// }}}RME
```

- 42 In Eclipse, right-click the **NorthAmericanIntersection_sparcgnu281** project and click **File > Build Project**.

The **Tasks** lists contains the error that you introduced.



- 43 Double-click the error in the **Task** list to locate the error in Eclipse, or right-click the error and select **Navigate to Rose RealTime**.



If you navigate to Rational Rose RealTime, the element (**timeout** transition) containing the error is highlighted.

Next, you will navigate from an operation, choice point, state (entry or exit), or transition in Eclipse to the corresponding element in Rational Rose RealTime.

Earlier, we saw that the **CodeGenDirName** box on the **C++ Generation** tab in Rational Rose RealTime contained **rtg**. This is the location where the generated source files for the selected Rational Rose RealTime component appear in the corresponding Eclipse project.

44 In Eclipse, expand the **rtg** folder under **build**.

45 In the **C/C++ Projects** view, navigate through the generated source code to find the transition **TrafficLightNA_Actor::transition21_greenTimeout**.



Note: For information about valid pattern names, see *Navigating from Eclipse to Rational Rose RealTime* on page 108.

46 Single-click on the transition to navigate to Rational Rose RealTime.

In Rational Rose RealTime, the transition is highlighted.

Troubleshooting

Refer to the following table to help troubleshoot building, connectivity, and navigation problems that might occur.

Area	Problem Description	Possible Resolution
Building	In the C-Build window, you encounter the error message " Error launching builder (make -f Makefile.eric all) (Exec error:Launching failed) " when you attempt to build a C/C++ project from Eclipse.	Add C:\cygwin\bin (or wherever you installed Cygwin) to your Path environment variable.
Connectivity	In Eclipse, the Current Status box on the Communication tab shows Listening to port <port_number> for a long period of time without establishing a connection.	Ensure that the port numbers are the same.
	In Eclipse, the Current Status box on the Communication tab shows Can't connect to <hostname or IP address>:<port_number> .	Only one listener can use the specified port number; multiple listeners are not permitted. Ensure that another connection with the same port number has not been made.
	In Eclipse, the Current Status box on the Communication tab shows Feature is not enabled .	The Rational Rose RealTime Integration option is currently disabled. In Rational Rose RealTime, click Tools > Eclipse Integration Settings , and then set the Enable Eclipse Integration option.
	In Eclipse, the Current Status box on the Communication tab shows Can't connect to <hostname or IP address>:<port_number> .	Verify that the Port number specified in Eclipse and Rational Rose RealTime are the same.
Navigation	In Eclipse, if you select an element from the list and right-click and select the option to edit that code in Rational Rose RealTime, you do not navigate to that code segment.	Some generated code, and some elements may not have existing code to edit. If a code segment for a corresponding element does not contain any USR tags, you will not be able to navigate to that element.

Navigation	In Eclipse, if you select a choice point method, state (entry or exit) method, or transition method, but nothing is selected in Rational Rose RealTime.	<p>You can recognize valid methods by the following patterns:</p> <ul style="list-style-type: none"> ▪ For a choice point method, the method name pattern is <i><class>::choicePoint<number>_<name></i> ▪ For a state entry method, the method name pattern is <i><class>::enter<number>_<name>(<arguments>)</i> ▪ For a state exit method, the method name pattern is <i><class>::exit<number>_<name></i> ▪ For a transition method, the method name pattern is <i><class>::transition<number>_<name></i>
------------	---	--

Contents

This chapter is organized as follows:

- *Overview of OSEK/VDX Support in Rational Rose RealTime* on page 133
- *Introduction to the OSEK OIL Files* on page 134
- *Configuring the TargetRTS and Adapting it for a Particular OSEK/VDX Operating System* on page 135
- *Updating Tasks* on page 136

Overview of OSEK/VDX Support in Rational Rose RealTime

Rational Rose RealTime for the C language includes support for operating systems adhering to the OSEK/VDX standard. The basic conformance classes for OSEK are:

- BCC1 - (Basic Conformance Class 1) Supports basic tasks with one request per task, and one task per priority. Every task has a different priority.
- BCC2 - (Basic Conformance Class 2) Allows for more than one basic task per priority, as well as multiple task-activation requests.

In Rational Rose RealTime, the supported conformance class is BCC2; however, you can use BCC1 with some restrictions, described later. In addition, you can also use the extended conformance classes ECC1 (similar to BCC1, but it permits extended tasks) and ECC2 (follows BCC2 without the multiple-task-request facility), if required.

Note: To use an OSEK/VDX operating system with Rational Rose RealTime, we recommend that you become familiar with the OSEK/VDX standard, the TargetRTS in Rational Rose RealTime, and the book *Adapting Rational Rose RealTime for Target Environments* (<\$ROSERT_HOME>/Help/rosert_adapting_targets.pdf).

Rational Rose RealTime includes the following directories (located in `$RTS_HOME/src/target/`) for OSEK/VDX support:

- `OSEK_VDX_BCC1`
- `OSEK_VDX_BCC2`
- `OSEKWorks`

All source files necessary to support the BCC1 conformance class of OSEK/VDX standard v. 2.1.1 are located in `/OSEK_VDX_BCC1`. Additional files for the BCC2 conformance are located in `/OSEK_VDX_BCC2`. The example provided with the **TargetRTS** for C is for OSEKWorks 3.0 and is located in the `/OSEKWorks` directory. The files in the `OSEK_VDX_BCC1` and `OSEK_VDX_BCC2` directories are based on the OSEK/VDX standard and can be used for any operating system that follows this standard. Every operating system will require some additional files specific to that operating system.

Note: If you use OSEKWorks, you must rebuild the TargetRTS libraries included with Rational Rose RealTime.

Introduction to the OSEK OIL Files

The OSEKWorks target base directory contains two OIL files: **TargetRTS.oil** for multi-threaded targets, and **TargetRTS_single.oil** for single-threaded targets. You can modify these OIL files as required. Single-threaded configurations and multi-threaded configurations without the timing service can use the BCC1 conformance class. Multi-threaded configurations used with the Rational Rose RealTime timing service must use the BCC2 target base because the BCC1 conformance class guarantees only one resource (**RES_SCHEDULER**), and the default timing service makes use of a resource protected by another resource.

Note: Single-threaded configurations and multi-threaded configurations without the timing service can use the BCC1 conformance class; however, multi-threaded configurations that are used with the timing service must use the BCC2 target base.

Every OSEK application must have its system objects specified in an OIL file. In the TargetRTS for C, the OIL file must reside in the OS-specific target base directory. For OSEKWorks, this directory is `$RTS_HOME/src/target/OSEKWorks/`.

The OIL preprocessor is called by the command in the **\$postprocessor** variable in the configuration's **setup.pl** file. For example, **\$RTS_HOME/config/OSEKWorks30T.ppc555-Diab-4.4b/setup.pl** contains line:

```
$postprocessor = "rtperl ../target/OSEKWorks30T/oil.pl  
../src/target/OSEKWorks/TargetRTS.oil OSEKWorks30T.ppc555-Diab-4.4b  
$target_base";
```

In this example, the OIL preprocessor is called through the **oil.pl** script in the **OSEKWorks** target directory. The **setup.pl** file also contains the following **\$target_base** variable that specifies how the Target RTS uses source from two or more different target bases. For example, if the **setup.pl** file specifies the following:

```
$target_base = 'OSEKWorks, OSEK_VDX_BCC2, OSEK_VDX_BCC1';
```

this means that the TargetRTS configuration uses files from **\$RTS_HOME/src/target/OSEKWorks/** whenever possible, and then it will use files from **\$RTS_HOME/src/target/OSEK_VDX_BCC2/**, and then from **\$RTS_HOME/src/target/OSEK_VDX_BCC1**.

By default, the multi-threaded TargetRTS OIL file is configured with eight tasks. This is the maximum guaranteed by the BCC1 configuration class. The operating system you use determines the maximum number of tasks; however, you can modify the number of tasks, as required. For information about modifying these tasks, see *Configuring the TargetRTS and Adapting it for a Particular OSEK/VDX Operating System* on page 135.

Typically, every physical thread in a Rational Rose RealTime application corresponds to a thread (task) on the target OS. This is not the case with OSEK/VDX. If a model has two or more physical threads with the same priority, these TargetRTS threads will alternate running on the same OSEK/VDX OS task. In the TargetRTS code, the Rational Rose RealTime physical threads are represented by **RTThread**, whereas the real OS tasks are represented by **RTTask**. This arrangement is necessary because the BCC1 conformance class permits only one task per priority.

Configuring the TargetRTS and Adapting it for a Particular OSEK/VDX Operating System

To port the TargetRTS for C to other OSEK/VDX operating systems, use the OSEKWorks example provided with Rational Rose RealTime. The new configuration should use the **OSEK_VDX_BCC1** target base (and the **OSEK_VDX_BCC2**, if necessary), and provide any OS-specific changes and OIL files in a separate target base directory. The OIL files included with Rational Rose RealTime are for

OSEKWorks, and they will require modifications to work with different operating systems. In particular, any ISR definitions are not portable, and the definition for the **SystemTimer** counter is also specific to the hardware used.

Note: The **SystemTimer** counter is necessary if you want to use any timing service.

The multi-threaded version of the OIL file defines eight tasks: **RTTask0** through **RTTask7**. The single-threaded OIL file defines only one task: **RTTask0**. These tasks correspond to start-up task functions defined in **\$RTS_HOME/src/target/OSEK_VDX_BCC1/Main/main.c**. They are **RTTask0func** through **RTTask7func**. All of the task definitions have the following structure:

```
TASK RTTask0 {  
    SCHEDULE = FULL;  
    PRIORITY = 0;  
    ACTIVATION = 1;  
    AUTOSTART = TRUE;  
    STACKSIZE = 512;  
    RESOURCE = RES_SCHEDULER;  
    RESOURCE = RTUserRes;  
};
```

Note: The resource **RTUserRes** is required for multi-threaded applications that use the Rational Rose RealTime timing service. The **RTUserRes** resource is implemented in **\$RTS_HOME/src/target/OSEK_VDX_BCC2/UsrMutex/**. If used, it must be defined in the OIL file as:

```
RESOURCE RTUserRes;
```

Note: This additional resource is the only thing that breaks BCC1 conformance class compatibility.

Updating Tasks

If you want to use fewer tasks, you must delete the tasks with the highest IDs first. For example, if a system has N tasks, the tasks are numbered 0 to $N - 1$. You want to start by deleting task **RTTask($N-1$)** first.

Note: You must set the alarm **RTTimerAlarm** to activate the highest priority task.

The file **\$RTS_HOME/src/target/OSEK_VDX_BCC1/RTPPriv/OSEK_VDX.h** contains a macro called **RT_OSEK_NUM_TASKS**. If you change the number of tasks, you must redefine this number.

To delete (or add) tasks from the system:

- 1 Change the macro `RT_OSEK_NUM_TASKS` in `$RTS_HOME/src/target/OSEK_VDX_BCC1/RTPPriv/OSEK_VDX.h` to the correct number of tasks.

- 2 Delete (or add) tasks in the OIL file.

Note: Number the tasks sequentially. The first task must have a priority of 0 (the lowest priority). This means that if the application is to have N tasks, the tasks must have priorities 0 through $N - 1$.

Each task has a start-up function defined in

`$RTS_HOME/src/target/OSEK_VDX_BCC1/Main/main.c`. The start-up functions are called `RTTasks<task_number>func` (for example, `RTTask1func`). You can delete any unused startup functions. The `main.c` file also contains the function `initRTTasks` that is responsible for initializing the `RTTasks` array. If you modify the number of tasks, change the initialization to only initialize the elements in the array that are used.

- 3 Delete (or add) corresponding task start-up functions in `$RTS_HOME/src/target/OSEK_VDX_BCC1/Main/main.c`.
- 4 Change the function `initRTTasks` in `$RTS_HOME/src/target/OSEK_VDX_BCC1/Main/main.c` to initialize the `RTTasks` array for the appropriate number of tasks.
- 5 Take note of the entry for the alarm in the OIL file. For example:

```
ALARM RTTimerAlarm {  
    COUNTER = SystemTimer;  
    ACTION = ACTIVATETASK {  
        TASK = RTTask7;  
    };  
};
```

The `RTTimerAlarm` has to activate the highest priority task.

The example OIL files specify that the application is to use extended status:

```
OS _OS {  
    STATUS = EXTENDED;  
};
```

Specifying an **EXTENDED** status is useful during the development because the operating system will perform better error checking, but it will degrade performance. For the final version of the application, specify a **STANDARD** status to improve performance. For example:

```
OS _OS {  
    STATUS = STANDARD;  
};
```

Note: Task stack sizes are defined in the OIL files, so the thread stack sizes in component's **Physical Threads Specification** dialog box in the model are ignored.

Index

Symbols

\$& 52
\$target_base 135

A

Abort 62
Abort documentation 62
adding
 GUIDs 73
 OSEK/VDX tasks 137
Attribute tool
 Get method 54
 Set method 54

B

Basic Conformance Class 133
BasicTest server output 34
BCC1 133
BCC2 133
browser requirements 43
building
 component on Solaris 2.6 47
 Eclipse 101

C

C Add-ins 54
C++ Add-ins 54
C++ Analyzer
 using (Linux) 48
CD-ROM
 mounting instructions 25
 unmount 30
CLASSPATH 53
-cleanup 59

ClearCase
 command-line access 30
 element type 31
 repository setup 31
 workstation setup 30
cm_getcaps script 53
code generator 54
codesync 85
collision of GUIDs 71
color
 restoring defaults 52
 settings 52
command-line access to ClearCase 30
CompilationMakeCommand 99
CompilationMakeType 99
Component 91
configuration management on UNIX 46
configuration requirements
 UNIX 20
 Windows 2000 18
 Windows XP Pro 19
configuring
 build settings in Eclipse 100
 build settings in Rational Rose RealTime 98
Connection Mode 83, 90
Connector 80
Connexis
 verifying the installation 32
contacting Rational Customer Support xiv
Context Sensitive Help 61
cross-platform issues 49
Current status 83, 91
Cygwyn 100

D

Debugger-Tornado2 43
Debugger-xxgdb 53
debugging
 using Tornado 2.2 on UNIX 45

- deleting
 - OSEK/VDX tasks 137
- dependencies 40
- descriptors for nested classes 55
- directory names 42
- documentation updates 62

E

- ECC1 133
- ECC2 133
- Eclipse
 - building 101
 - Communication Overview 79
 - Communication settings 82
 - Configuring Build Settings 100
 - configuring build settings 100
 - configuring preferences 81
 - Connection tab
 - Connection Mode 83
 - Current status 83
 - Host name or IP address 83
 - Port number 83
 - Restore to Defaults 84
 - connector 80
 - editing code 94
 - Enable Rose RealTime Integration 82
 - establishing a connector 80
 - Features tab 84
 - Also select elements in browsers 86
 - Do a codesync on save 85
 - Select elements from C/C++ Projects View 85
 - Select elements from Outline View 86
 - Generating Code 92
 - Integration Overview 77
 - integration with Rational Rose RealTime 77
 - method name patterns 110
 - navigating from Rational Rose RealTime 104
 - navigating to Build Errors 103
 - Path Mappings tab 86
 - Path in Eclipse end 88
 - Path in Rose RealTime end 88

- preferences 81
- referenced configuration requirements 21
- removing generated code 92
- synchronizing code 93
- troubleshooting integration 131
- Eclipse integration
 - example workflow 113
 - getting started 110
- editing
 - code in Eclipse 94
- element type setup (Linux) 31
- Enable Eclipse Integration 90
- enabling
 - GUIDs 71
- example workflow (Eclipse integration) 113
- Exceed
 - refresh issues 45

F

- Favorites list 61
- Find command 58
- font issues 53
- Frameworks dialog 55

G

- generating
 - code in Eclipse 92
 - GUIDs 71
 - large models 54
- Get method 54
- GetSelected 56
- GetSelected functions 56
- GetSystemTime 41
- GetTickCount 41
- getting started
 - Eclipse and Rational Rose RealTime integration 110
- Globally Unique Identifiers
 - see GUIDs 69
- GNU libraries
 - updating 48

GUIDs

- adding 73
- advanced handling of 69
- check unique identifiers presence 74
- collisions 71
- enabling 71
- generating 71
- issues 75
- known issues 75
- managing 72
- minimizing impact of collisions 71
- regenerating 74
- removing 75
- setting 73
- turning off 75
- turning on 73
- using 69

H

- Help issues 60
- Host Configurations 66
- Host Name 83
- Host name or IP address 91
- host platform installation 32

I

- Index tab 62
- initRTTasks 137
- installation
 - disk space requirements 17, 24
 - host platform 32
 - restarting (Linux) 24
 - stopping (Linux) 24
- installing
 - Linux instructions 25
 - multiple versions 24
 - Rational Rose RealTime on Linux 25
- integration
 - Enable Eclipse Integration 90
 - Rational Rose RealTime and Eclipse 77
- IP address 83

issues

- cross-platform 49
- displaying the version tree (Linux) 48
- Linux 47
- online Help 60
- start-up 37
- uninstalling 38
- UNIX 43
- updating GNU libraries (Linux) 48
- using the C++ Analyzer (Linux) 48
- Windows 39

J

Java

- CLASSPATH 53
- compiling models 53
- no automatic unwired port registration 58
- no codesync support 56

L

- Launching Model Integrator 64
- LD_LIBRARY_PATH 29, 48

licenses

- expiration of 50
- licensing options (Linux) 28

Linux

- after you install 29
- before you install 23
- C++ Analyzer 48
- Connexis Viewer 48
- displaying the version tree 48
- installation instructions 25
- issues 47
- requirements 17, 24
- set Connexis Variable 32
- setup script 29
- shortcut keys 49
- starting Rational Rose RealTime 36
- unmount CD-ROM 30
- updating GNU libraries 48
- using shortcut keys 49

- using Web Publisher with Exceed 49
- viewing the online Help 49
- Listener 80

M

- Makefile.eric 100
- managing
 - GUIDs 72
 - unique identifiers 72
- method name patterns 110
- Model Integrator
 - launching 64
- models
 - generating 54
- mounting
 - CD-ROM 25
- Mutex 136

N

- netinet 42
- node -ocked licenses 50

O

- OIL files (OSEK) 134
- oil.pl 135
- online Help
 - navigating 60
 - viewing animated demonstrations 60
- Operation tool
 - Get method 54
 - Set method 54
- OSEK
 - OILfiles 134
 - using 133
- OSEK_VDX_BCC1 134
- OSEK_VDX_BCC2 134
- OSEK_VDX.h 136
- OSEK/VDX
 - add tasks 137
 - BCC1 133

- BCC2 133
- configuring the TargetRTS 135
- delete tasks 137
- EXTENDED status 138
- initRTTasks 137
- main.c 137
- oil.pl 135
- OSEK_VDX_BCC1 134
- OSEK_VDX_BCC2 134
- OSEK_VDX.h 136
- Physical Threads Specification 138
- RES_SCHEDULER 134
- RT_OSEK_NUM_TASKS 136
- RTTask 135
- RTTask0 136
- RTTask7func 136
- RTThread 135
- setup.pl 135
- STANDARD status 138
- SystemTimer 136
- TargetRTS_single.oil 134
- TargetRTS.oil 134
- tasks update 136
- OSEK/VDX Support 133
- OutPutDirectory 52

P

- path mappings 88
 - between Windows and UNIX 88
 - between Windows configurations 89
 - specifying (example) 88
- PathMap 52
- paths 46
- permissions
 - Rose RealTime 40
- Port Number 83, 90
- preferences
 - configuring 89
- printing diagrams 58
- printing on UNIX 43

R

- RAM requirements 17, 24
- Rational Customer Support
 - contacting xiv
- Rational Rose RealTime
 - Communication Overview 79
 - configuring preferences 89
 - Integration Overview 77
 - Integration with Eclipse 77
 - troubleshooting integration with Eclipse 131
- rational_dir 24
- Referenced Configuration for Nucleus 59
- referenced configuration requirements
 - Eclipse integration 21
 - Windows 2000 18
 - Windows NT 18
 - Windows UNIX 20
 - Windows XP Pro 19
- referenced configurations 18
- referenced host configurations 66
- regenerating
 - GUIDs 74
- removing
 - generated code 92
 - GUIDs 75
- ReplaceRoseHelpDir 61
- repository setup for ClearCase 31
- requirements
 - disk space 17, 24
 - Linux installation 17, 24
 - RAM 17, 24
 - referenced configurations 18, 19, 20
 - UNIX installation 20
 - updates 39
 - Windows NT installation 18
- RES_SCHEDULER 134
- restarting
 - installation 24
- restoring default colors 52
- ROBERT_NO_FEEDBACK 58
- ROBERT_TORNADO_TIMEOUT 59
- rs_install 26
- RT_OSEK_NUM_TASKS 136
- RTCapsule_context 65

- RTMessage_getData 64
- RTMessage_getPortIndex 63
- RTTask 135
- RTTask0 136
- RTTask0func 136
- RTTasks 137
- RTThread 135
- RTTimerAlarm 136
- RTUserRes 136
- run
 - install program 26

S

- saving
 - codesync 85
- script
 - check_rose_reqs 17, 20, 24
- scripts
 - file association 41
- Sequence Diagrams 53
- service pack updates 39
- Set method 54
- setting
 - GUIDs 73
- setup script 29
- setup.pl 135
- shortcut keys on Linux 49
- source control
 - command-line access to ClearCase 30
- stack space limit 47
- starting
 - Rational Rose RealTime (Linux) 36
 - Rose RealTime when one instance is running 37
 - Rose RealTime when using virus scanning software 38
 - toolset freezes on startup 38
 - vi editor from Rational Rose RealTime 46
- startup-issues 37
- stopping an Installation 24
- synchronizing code 93
- SystemTimer 136

T

- target
 - connecting problems 53
- TargetConfiguration 99
- TargetRTS
 - configuring for OSEK/VDX 135
 - symbolic links with 42
- TargetRTS_single.oil 134
- TargetRTS.oil file 134
- Timeout for wtx commands 59
- toolset
 - freezes on startup 38
- Troubleshooting
 - integration settings between Eclipse and Rose RealTime 131
 - toolset freezing on UNIX 44
- type descriptors
 - example model update 66
 - using 67
 - when to use 67
- type manager 31

U

- uninstall issues 38
- UNIX 46
 - browser requirements 43
 - case sensitivity within paths 46
 - configuration requirements 20
 - debugging using Tornado 2.2 45
 - editors 47
 - exceptions when using CM 46
 - file names containing spaces 45
 - issues 43
 - printing 43
 - recovering from toolset freeze 44
 - refresh problems with Exceed 45
 - requirements 20
 - stack space limit 47
 - starting vi editor 46
 - troubleshooting when the toolset freezes 44
- unmount CD-ROM 30

- updating
 - GNU libraries (Linux) 48
- UsrMutex 136

V

- verifying
 - Connexis installation 32
 - host platform installation 32
 - Linux installation 32
- version tree
 - displaying 48
- vi
 - editor 46
 - starting 46
- virus software affects startup 38

W

- Web Publisher 49
 - applet not loading 55
- window limitations 51
- window order policy 46
- Windows
 - building dependencies 40
 - file associations for compiled scripts 41
 - issues 39
 - links with TargetRTS 42
 - permissions 40
 - privileges 40
 - service pack requirements 39
 - service pack update 39
 - spaces in directory names 42
 - using Hummingbird Exceed 39
 - using Rose RealTime on Windows XP
 - configurations 39
- Windows 2000 18
- Windows CE
 - GetSystemTime function 41
 - GetTickCount 41
- Windows NT
 - configuration requirements 18

Windows XP Pro
 configuration requirements 19
wtx 59

X

-xxgdb debugger 53