



Installation Guide

Rational Rose RealTime

IMPORTANT NOTICE

COPYRIGHT

Copyright ©1993-2001, Rational Software Corporation. All rights reserved.

Portions Copyright ©2000-2001, Compaq Computer Corporation. All rights reserved.

Portions Copyright ©1992-2000, Summit Software, Inc. All rights reserved.

Part Number: 800-024514-000

Version Number: 2001A.04.00

PERMITTED USAGE

THIS DOCUMENT CONTAINS PROPRIETARY INFORMATION WHICH IS THE PROPERTY OF RATIONAL SOFTWARE CORPORATION ("RATIONAL") AND IS FURNISHED FOR THE SOLE PURPOSE OF THE OPERATION AND THE MAINTENANCE OF PRODUCTS OF RATIONAL. NO PART OF THIS PUBLICATION IS TO BE USED FOR ANY OTHER PURPOSE, AND IS NOT TO BE REPRODUCED, COPIED, ADAPTED, DISCLOSED, DISTRIBUTED, TRANSMITTED, STORED IN A RETRIEVAL SYSTEM OR TRANSLATED INTO ANY HUMAN OR COMPUTER LANGUAGE, IN ANY FORM, BY ANY MEANS, IN WHOLE OR IN PART, WITHOUT THE PRIOR EXPRESS WRITTEN CONSENT OF RATIONAL.

TRADEMARKS

Rational, Rational Software Corporation, Rational the e-development company, ClearCase, ClearCase Attache, ClearCase MultiSite, ClearDDTS, ClearQuest, ClearQuest MultiSite, DDTS, Object Testing, Object-Oriented Recording, ObjecTime & Design, Objectory, PerformanceStudio, ProjectConsole, PureCoverage, PureDDTS, PureLink, Purify, Purify'd, Quantify, Rational, Rational Apex, Rational CRC, Rational Rose, Rational Suite, Rational Summit, Rational Visual Test, Requisite, RequisitePro, RUP, SiteCheck, SoDA, TestFactory, TestFoundation, TestMate, The Rational Watch, AnalystStudio, ClearGuide, ClearTrack, Connexis, e-Development Accelerators, ObjecTime, Rational Dashboard, Rational PerformanceArchitect, Rational Process Workbench, Rational Suite AnalystStudio, Rational Suite ContentStudio, Rational Suite Enterprise, Rational Suite ManagerStudio, Rational Unified Process, SiteLoad, TestStudio, VADS, among others, are either 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.

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, Virtual Basic, the Virtual 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, among others, 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.

FLEXIm and GLOBEtrotter are trademarks or registered trademarks of GLOBEtrotter Software, Inc. Licensee shall not incorporate any GLOBEtrotter 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.

Portions of this product incorporate the expat XML parser 1.0 under the Mozilla 1.1 license available at <http://www.mozilla.org/MPL/MPL-1.1.txt>. The source code version of the expat XML parser is available at <http://www.jclark.com/xml/expat.html>.

PATENT

Portions covered by U.S. Patent Nos. 5,193,180 and 5,335,344 and 5,535,329 and 5,835,701 and 5,574,898 and 5,649,200 and 5,675,802.

U.S. Patents Pending.

International Patents Pending.

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

GOVERNMENT RIGHTS LEGEND

Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in the applicable Rational Software Corporation license agreement and as provided in DFARS 277.7202-1(a) and 277.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (Oct. 1988), FAR 12.212(a) (1995), FAR 52.227-19, or FAR 227-14, as applicable.

WARRANTY DISCLAIMER

This document and its associated software may be used as stated in the underlying license agreement. 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 or fitness for a particular purpose or arising from a course of dealing, usage, or trade practice.





Contents

Chapter 1	Introduction	1
	Welcome to Rational Rose RealTime	1
	Release Notes	1
	Installation Guide Updates	1
	Overview of Rose RealTime Capabilities	2
	What's New?	3
	How to Get Help	3
	Contacting Rational Technical Support Through the Help Menu	3
	Contacting Rational Technical Support by Email or Telephone	4
	License Support Contact Information	5
	Evaluation and Ordering Information	7
	Rational Web Site	7
	Directory Contents	7
	Accessing the Online Help System	9
Chapter 2	Platform and Toolchain Requirements	11
	Platform Requirements — Windows NT	11
	Platform Requirements — Windows 2000	11
	Platform Requirements — UNIX	12
	Toolchain Requirements	12
	Help Viewer (Windows Platforms Only)	12

	Compiler	13
	Real-time Operating System	13
	Supported Host Platforms	13
	Creating Executables for Hosts without Toolset Support	16
	Generating an executable without a common file system	17
	Adding a Printer on UNIX	18
Chapter 3	Installing Rational Rose RealTime on Windows	21
	Upgrade Information	21
	Installation Instructions	21
	Installing on a Network Drive	24
	Testing your Environment	25
Chapter 4	Installing Rational Rose RealTime on UNIX	27
	Upgrade Information	27
	Installation Instructions	28
	To Install Rational Rose RealTime on UNIX:	28
	Setting Up a User Workstation	30
	Environment variables	30
	Additional settings	31
Chapter 5	Understanding Rose RealTime Licenses	33
	How Licenses Work	33
	FLEXlm License Server	34
	FLEXlm components	34
	License manager daemon (lmgrd)	34
	Vendor daemon	35
	License key file	35
	Application program	36
	License activation process	36
	Licensing on UNIX	37
	Running the LMGRD from a Command Prompt	37
	Example	37

Administration commands	38	
The License File	38	
Format	39	
Chapter 6	Installing License Keys	41
Installing a Startup or Permanent License on Windows		41
Installing a Permanent License on Windows		43
Installing the License Key		44
Installing a Floating License Key on a UNIX server		44
Installing a Startup or Permanent License on UNIX		45
Installing a Startup License on UNIX		45
Installing a Permanent License on UNIX		46
Installing the License Key		47
Integration With Rational Suites Licensing		48
Troubleshooting		48
Windows		48
UNIX server		49
UNIX		50
Chapter 7	Migration	53
Migrating from Rational Rose		53
User Interface Differences		53
New Modeling Language Elements		55
Code Generation, Building, and Running		56
Opening Models from Rational Rose		56
List of Importation Log Messages		57
Limitations and Restrictions		57
Importing Rational Rose Generated Code		58
Limitations and restrictions		59
Migrating from ObjecTime Developer 5.2/5.2.1		59
Terminology		59
User Interface Differences		61
Compilation		62

Migrating from Rose RealTime 6.0/6.0.1/6.0.2/6.1	62
File Format Changes	63
Source Control Migration	63
Migrating customized CM scripts	64
Language Add-in Changes	65
Running Two Different Releases of Rose RealTime	65
Workspace Files	65
RRTEI Changes	66
C Language Migration	68
Converting a C++ Model to C	68
ObjecTime Developer for C Migration	69
Importing models	69
Converting global signals to local signals	70
Timing service	71
C++ Language Migration	71
Backwards Compatibility Mode	71
Migrating in two steps	72
What does backwards compatibility do?	72
Compiler will find all errors	73
Building a model in backwards compatibility mode	73
Full migration	76
Changes	76
C++ UML Services Library	76
Code generation and compilation	76
New classes for protocols, signals, and ports	77
Type safety explained	77
How has this been changed?	77
API changes summary	78
Asynchronous sends	79
Synchronous sends	80
Message reply	80
Defer, recall, and purge	81
Port indexes	82
Discriminating in code the signal of a received message	82
Forwarding	83
RTPortRef operations	85
RTTimespec parameters	86
RTSignalNames	87
Macros	87

External Layer Service (ELS)	87	
Code Generation	87	
Components	88	
Directory structure	88	
Parameters available in transition code	88	
Port cardinality cannot be unspecified	89	
Makefile overrides changes	89	
Model Properties	89	
Advanced property editors	90	
Chapter 8	Integration Notes	91
Configuration Management (CM) Tools Integration	91	
ClearCase on a UNIX Server and Clients on both NT and UNIX	92	
Migrating from Rational Rose and ObjecTime Developer	92	
Requirements Management Tools Integration	92	
Rational SoDA for Word	93	
Rational RequisitePro	93	
Unit Testing Tools Integration	93	
Rational Purify	93	
Adding options to Purify on UNIX	94	
Microsoft Development Environment	94	
Integration with Rational Robot	94	
Naming Directories	95	
Chapter 9	Starting Rational Rose RealTime	97
Starting Rose RealTime under Windows	97	
Starting Rose RealTime on UNIX	97	
Start-up options for UNIX	98	
Rose RealTime for UNIX and the X Window System	99	
X clients	99	
X servers	99	
X window managers	99	
Input focus (active window) policy	100	
Window order policy	100	

Automating Rose RealTime 101

Command Line Options 101

Chapter 10 Add-ins 103

Web Publisher 103

Description 103

Suggested workflow 104

Limitations 104

Model Integrator 105

Description 105

Suggested workflow 106

Rose C++ Analyzer 107

Description 107

Suggested workflow 107

Limitations 109

Chapter 11 Uninstalling Rational Rose RealTime 111

Windows 111

UNIX 111



Chapter 1

Introduction

Welcome to Rational Rose RealTime

Rational Rose RealTime is a comprehensive visual development environment that delivers a powerful combination of notation, processes, and tools to meet the challenges of real-time software development. Through the industry-standard Unified Modeling Language (UML), real-time design constructs, code generation, and model execution capabilities, Rational Rose RealTime addresses the complete lifecycle of a project: from early use case analysis, through to design, implementation, and testing.

Rational Rose RealTime is designed for simple insertion into your software development environment, processes, and workflows. Rose RealTime includes seamless integration with other Rational products and support for a variety of commercial real-time operating systems.

This guide provides the necessary information to install Rational Rose RealTime in your environment.

Release Notes

See the *Rational Rose RealTime Release Notes* for information on system requirements, known limitations, documentation updates, and troubleshooting information.

Installation Guide Updates

For the latest documentation updates, please refer to the Rational Rose RealTime web site:

<http://www.rational.com/support/>

Navigate to the [Documentation](#) link.

Overview of Rose RealTime Capabilities

Modeling:

- Use Case Modeling
- Class Modeling
- Collaboration (role) Modeling
- Interaction Modeling (sequence diagrams)
- Component Modeling
- Deployment Modeling

Application Generation:

- C++ Language Support
- Java Language Support
- C Language Support
- Data Class Code Generation

Visual Execution:

- Host Execution
- Target Execution
- Model Visualization (Animation)
- Model Debugging (Tracing, Injection, Inspection)

Tools Interworking:

- Rational ClearCase
- Microsoft Visual SourceSafe (Windows only)
- SCCS (UNIX only)
- RCS (UNIX only)
- Rational SoDA (requires Rational Rose RealTime domain)
- Rational RequisitePro
- Rational Purify

Model Documentation:

- Report Generation (Windows only)
- Web Publisher

What's New?

These are the new features included in this release of Rose RealTime.

- Support for Automated Testing
 - Rational Quality Architect - RealTime
- Support for Additional platforms and languages
 - now includes Java language support
 - new host support
 - Solaris 2.8
- Team Development Improvements
 - Model Integrator improvements
 - enhanced model sharing with shared interface packages
- Usability Improvements
 - improved start-up with a customizable frameworks wizard
 - runs in English, supports non-English data on German, French, Italian, Swedish, and Japanese operating systems

How to Get Help

This section describes procedures for interacting with Rational Software Corporation's technical support services.

Contacting Rational Technical Support Through the Help Menu

With Rational Rose RealTime, you can email problem reports, feature requests, or support requests to the Rational Software Technical Support department that services your location, directly from the Rose RealTime application's Help menu.

For details on how to use this feature, see the Technical Support chapter of the *Rational Rose RealTime Release Notes*.

Contacting Rational Technical Support by Email or Telephone

When contacting Rational Technical Support by email or by telephone, please be prepared to supply the following information:

- Name, telephone number, and company name
- Product name and version number
- Operating system and version number (for example, Windows NT 4.0, Windows 2000, Solaris 2.6/2.7/2.8, or HP-UX 10.20)
- Computer make and model
- Your case id (if you're calling about a previously reported problem)
- A summary description of the problem, related errors, and how it was made to occur

If your organization has a designated, on-site support person, please try to contact that person before contacting Rational Technical Support.

You can obtain technical assistance by sending electronic mail to the appropriate e-mail address. Electronic mail is acknowledged immediately and is usually answered within one working day of its arrival at Rational. When sending an email place "Rational Rose RealTime" in the subject line, and in the body of your message include a description of your problem.

When sending email concerning a previously-reported problem, please include in the subject field: "CaseID: v0XXXXXX", where XXXXXX is the caseid number of the issue. For example:

```
CaseID: v0176528 New data on rational rose realtime install issue
```

Sometimes Rational technical support engineers will ask you to fax information to help them diagnose problems. You can also report a technical problem by fax if you prefer. Please mark faxes "Attention: Technical Support" and add your fax number to the information requested above.

Telephone and fax numbers for Rational Technical Support are contained in the following table. If you have problems or questions regarding licensing, please see "License Support Contact Information" on page 5.

Table 1 Support Telephone and Fax

Region	Telephone Number	Fax Number
Americas	800-433-5444	408-863-4300
Asia Pacific (includes support for Japan, China, India, Korea, Taiwan)	+61-2-9419-0111	+61 2 9419 0123
Europe, Middle East, and Africa (includes support for Israel)	31 (0)20 4546 200	+31 23 569 4302
Other worldwide locations	408-863-5000	

Email addresses for Rational Technical Support are listed in the following table.

Table 2 Support Email

Region	Email Address
Americas and other worldwide locations	support@rational.com
Asia Pacific (includes support for Japan, China, India, Korea, Taiwan)	support@apac.rational.com
Europe, Middle East, Africa (includes support for Israel), and Scandinavia	support@europe.rational.com

License Support Contact Information

If you have a problem or questions regarding the licensing of your Rational Software products, please contact the Licensing Support office nearest you.

Telephone numbers for license support are listed in the following table. Ask for, or select, Licensing Support.

Table 3 License Support Telephone and Fax

Region	Telephone Number	Fax Number
Americas	800-433-5444	781-676-2510
Europe, Israel, and Africa	+31 23 554 10 62	+31 23 554 10 69
North Asia Pacific (Mainland China, Hong Kong, Taiwan)	+852 2143 6382	+852 2143 6018
Korea	+82 2 556 9420	+82 2 556 9426
South Asia Pacific Australia, New Zealand, Malaysia, Singapore, Indonesia, Thailand, The Philippines, Vietnam, Guam and India	+612 9419 0100	+612 9419 0160
Japan	+81 3 5423 3611	+81 3 5423 3622

Email addresses for license support are listed in the following table.

Table 4 License Support Email

Region	Email Address
Americas	lic_americas@rational.com
Europe, Israel, and Africa	lic_europe@rational.com
North Asia Pacific Mainland China, Hong Kong, Taiwan, and Korea	lic_apac@rational.com
South Asia Pacific Australia, New Zealand, Malaysia, Singapore, Indonesia, Thailand, The Philippines, Vietnam, Guam and India	lic_apac@rational.com
Japan	lic_japan@rational.com

Evaluation and Ordering Information

United States and Canada

Rosebud@rational.com

1-800-728-1212

Other Worldwide locations

Rosebud@rational.com

+1-408-863-9900

Rational Web Site

You can contact technical support and obtain the latest product information through our web site at:

<http://www.rational.com/support>

Directory Contents

After installation of the main Rose RealTime files has been completed, the directory structure should be as follows. Please ensure that the installation directory \$ROSSERT_HOME on UNIX and %ROSSERT_HOME% on Windows NT and all its associated files are readable, and not writable, by all users of Rose RealTime. The directory and its sub-directories contain all the individual files that comprise the particular release. Some of the files and directories are:

ROSSERT_HOME

This is the top level directory.

Help

This directory and its subdirectories contains the online Help.

bin

This directory contains the Rose RealTime executable and various scripts. The bin directory also contains subdirectories for each of the supported workstation platforms ROSSERT_HOST.

`license`

This directory contains various files containing encrypted information and is used by the License Manager in order to ensure that Rose RealTime is being executed according to the End-User License Agreement.

`Scripts`

This directory contains various Rose RealTime scripts.

`C++ or C`

This directory contains the libraries, header files, scripts relating to code generation, and source files for the Services Library. For more information regarding the Services Library see the *Toolset Guide* and the programmer's guides.

`RTJava`

This directory contains the classes and scripts relating to code generation in Java, See the *Java Reference* for more information.

`AddIns`

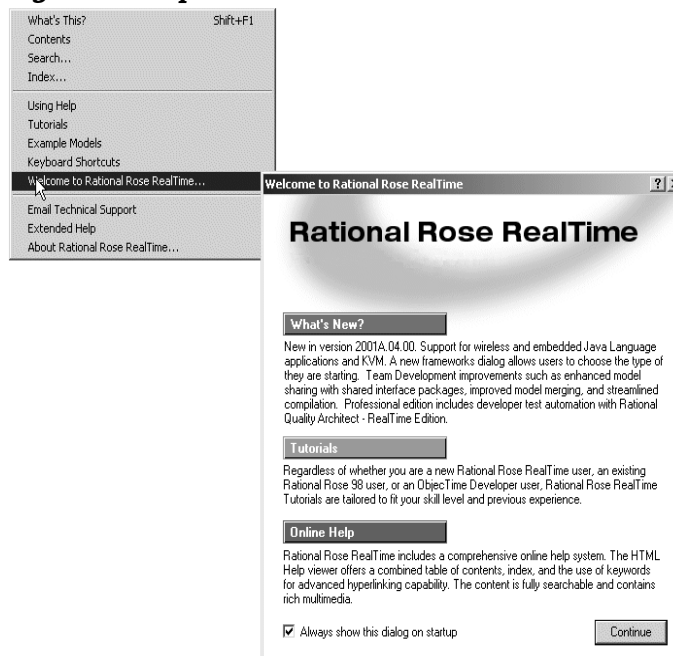
This directory contains the configuration information required by Rose RealTime Add-ins.

Note: For the latest integration information and line-up details, please consult the *Rational Rose RealTime Release Notes*.

Accessing the Online Help System

Online Help and documentation for Rational Rose RealTime is provided in Microsoft HTML Help format. You can load the online Help Viewer from the Rose RealTime toolset.

Figure 1 Help menu and Welcome screen



The Help Viewer requires that Microsoft Internet Explorer (version 3.02 or later) be set up on a user's computer. It is not required that Internet Explorer be used as the system's default browser, or that the Internet Explorer icon be visible on the user's desktop.

If you choose not to have Internet Explorer as the default browser, you will need to run Hhupd.exe (in redist). This file is the distribution executable that installs the run-time components needed for an HTML Help Project, such as Hh.exe, Hhctrl.ocx, Itss.dll, and Itircl.dll. Hhupd.exe is in the Redist folder of the HTML Help Workshop folder.

PDF versions of all the guides are available in the ROSERT_HOME/Help directory.



Chapter 2

Platform and Toolchain Requirements

This section describes the platform and toolchain requirements for running Rational Rose RealTime.

Note: *Rational Rose RealTime is not supported on Windows 95 or Windows 98.*

Platform Requirements — Windows NT

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

- Windows NT 4.0, with service pack 6a
- Minimum Pentium 150 MHz. We recommend 500 MHz or faster CPU
- Minimum 128 MB of RAM. We recommend 256 MB of RAM
- Minimum 325 MB of disk space for the Rose RealTime installation
- Minimum display 1024 X 768. We recommend 1280 X 1024 or better
- Postscript printer for printing
- Browser requirement — Internet Explorer 5.01 or 5.5 or Netscape Navigator 4.7 or 6.0. We recommend Internet Explorer 5.01 or 5.5

Platform Requirements — Windows 2000

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

- Windows 2000 Professional, with service pack 1

- Minimum Pentium 150 MHz. We recommend 500 MHz or faster CPU
- Minimum 128 MB of RAM. We recommend 256 MB of RAM
- Minimum 325 MB of disk space for the Rose RealTime installation
- Minimum display 1024 X 768. We recommend 1280 X 1024 or better
- Postscript printer for printing
- Browser requirement — Internet Explorer 5.01 or 5.5 or Netscape Navigator 4.7 or 6.0. We recommend Internet Explorer 5.01 or 5.5

Platform Requirements — UNIX

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

- Solaris 2.6, Solaris 2.7, Solaris 2.8, or HPUX 10.20
 - For Solaris operation, the minimum workstation is an UltraSparc 10 with 500 MB or RAM. We recommend an UltraSparc 60 with 600 MB or RAM. We recommend the Solaris 2.8 operating system.
 - For HPUX operation, we support installation of the HP 700 series architecture
 - Please see the Rational Rose RealTime website (<http://www.rational.com/support>) for a list of the required UNIX patches applicable to your operating system.
- The minimum is 500 MB or RAM. We recommend 600 MB of RAM
- Minimum 370 MB of disk space for the Rose RealTime installation
- Postscript printer for printing

Toolchain Requirements

Help Viewer (Windows Platforms Only)

The Help Viewer requires that Microsoft Internet Explorer (version 3.02 or later) be set up on your computer. For details, see “Accessing the Online Help System” on page 9.

Compiler

You must have a C++ compiler installed on your system to make use of the code generation and execution capabilities for Rose RealTime. Different compilers are required for host workstation and for embedded system targets. The list of supported compilers and targets is provided in “Supported Host Platforms” on page 13.

Real-time Operating System

If you are planning to deploy your model on a real-time operating system, your operating system, hardware and tool lineup must be one of the supported lineups listed in “Supported Host Platforms” on page 13. If you do not have a supported lineup, you may be able to get support for your lineup from a Rational RoseLink partner, or by customizing the Rose RealTime Services Library for your target. For instructions on customizing the Services Library and compiling for new target platforms, see the *C++, C, or Java Reference* .

Supported Host Platforms

Table 5 shows the supported host platforms for this release of Rational Rose RealTime.

Table 5 Host platforms

Toolset Host
Solaris 2.6
Solaris 2.7
Solaris 2.8
Windows NT 4.0
Windows 2000
HPUX 10.20

Note: *Java generation on HPUX is not supported.*

A pre-defined set of the Rose RealTime UML Services Libraries are delivered as part of the Rational Rose RealTime product. The UML Services Library is what allows standalone executable models to be executed on target operating systems. These ports are fully tested by Rational, and are covered by standard Rational support. A standard port can be used to facilitate a port to your environment of choice.

Note: For a more detailed description of the Services Library, refer to the programmer's guides, or online Help.

A port is based on the following specifications (often called the toolchain line-up):

- OS version
- Compiler version
- Processor type

If you are using a line-up other than the one tested by Rational and listed in this guide, standard support will cover problems encountered by customers only to the extent that the problem is reproducible on the line-up listed in this guide.

Table 6 shows the supported platforms and targets.

Table 6 Supported platforms and targets

Host Platform(s)	Target RTOS	Compiler/Processor	RTS	DCS
Solaris	Same	Gnu 2.95.1, sparc	C++	C++
		Gnu 2.8.1, sparc	C & C++	C++
		Gnu 2.7.2.3, sparc	C++	-
		Sun C++ 5.0, sparc	C++	C++
		Sun C 5.0, sparc	C	-
HPUX	Same	Gnu 2.8.1, hppa	C & C++	C++
		HP C++ 10.11, hppa	C++	-
Windows	Same	Visual C++ 5.0, x86	C & C++	C+
		Visual C++ 6.0, x86	C & C++	+C++
Solaris	pSOS 2.5	Diab 4.2b, ppc	C++	C++
Solaris, HPUX	C++	Microtec 1.3C, ppc	C++	n/a
Windows	VRTX 4.Baa	Microtec 1.4, ppc	C++	n/a
Solaris, Windows	OSE 4.1.1	Diab 4.3f, ppc	C & C++	C++
		GreenHills 1.8.9, ppc	C	-
		GreenHills 2.0, ppc	C	-
Solaris	OSE 4.1.1 SoftKernel	Gnu 2.95.1, sparc	C & C++	n/a
Windows	OSE 4.1.1 SoftKernel	Visual C++ 6.0, x86	C	-
Solaris, Windows, HPUX	Tornado 1.0.1 (VxWorks 5.3.1)	Cygnus 2.7.2.960126, M68040,	C & C++	-
		Cygnus 2.7.2.960126, ppc,	C & C++	C++
		Cygnus 2.7.2.960126, x86	C & C++	-
Windows	Tornado 1.0.1	Cygnus 2.7.2.960126, I960	C++	-
Solaris, Windows, HPUX	Tornado 2.0(VxWorks 5.4)	Cygnus 2.7.2.960126, M68040	C & C++	-
		Cygnus 2.7.2.960126, ppc	C & C++	C++
		Cygnus 2.7.2.960126, x86	C & C++	-
		GreenHills 1.8.9, ppc	C & C++	C++
		GreenHills 2.0, ppc	C & C++	C++
Solaris	Tornado 2.0 Sim	Cygnus 2.7.2.960126, sparc	C++	C++
Windows	Tornado 2.0 Sim	egcs 2.90.29, x86	C++	C++
Solaris, Windows	LYNX 3.1.0a	gnupro-2.9-98r2, ppc	C++	C++

Table 6 Supported platforms and targets

Host Platform(s)	Target RTOS	Compiler/Processor	RTS	DCS
Solaris	LYNX 3.0.1	Cygnus 2.7.97r1, x86 Cygnus 2.7.97r1, ppc	C++ C++	C++ C++
Solaris	Chorus Classix 4.0	egcs-2.91.66, ppc	C++	-
Windows	Windows CE sh3	eMbedded Visual C++ 3.0, sh3	C++	C++
N/C - native compilation only	AIX 4.2.1	gnu 2.8.1	C++	-
N/C - native compilation only	IRIX 6.	gnu 2.8.1	C++	-
N/C - native compilation only	Nucleus 1.1	Diab 4.2b	C++	-
N/C - native compilation only	Red Hat Linux 6.1	Egcs 2.91.66	C++	C++
N/C - native compilation only	QNX 4.2.2	Watcom C++ 10.6	C++	-
N/C - native compilation only	UnixWare 7.0.1	SDK 3.0	C++	C++

Creating Executables for Hosts without Toolset Support

For hosts without toolset support, create an executable on the host target.

Note: *The following steps assume that you are using a common file system hierarchy and that paths are equivalent on both machines.*

To produce an executable for a host without toolset support:

1. Select **Tools > Options** and click the **C++ Compilation** tab. Click **Select** in the **TargetConfiguration** area.
2. In the **Target Configuration** dialog, select the appropriate target configuration and click **OK**.
3. On the **C++ Generation** tab, ensure that **CodeGenMakeType** and **CodeGenMakeCommand** are appropriately set for the toolset host.

4. On the **C++ Compilation** tab, ensure that `CompilationMakeCmd` and `CompilationMakeType` are appropriately set for the compilation host.
5. Build the component with a build level set to “Generate”. This creates the source files and makefiles, required for compilation on the target host.

Note: *If the computer that you are using to compile does not have a common file system with the generated host, see “Generating an executable without a common file system” on page 17.*

6. From the build directory on the target host, set the environment variables for the compilation line-up.
7. Invoke the appropriate make command for the line-up.

Note: *If you build the source files on Windows NT and compile on UNIX, see the steps below about converting Windows files to UNIX type.*

Generating an executable without a common file system

If you build the source files on Windows NT and are compiling on UNIX, you must convert your files to UNIX type before you compile and generate an executable.

To generate an executable without a common file system:

1. On the target host, a visible copy of the `TargetRTS` must be available.
2. Copy the component directory into the target host file system.
3. Edit the build/makefile so `RTS_HOME` is set to location of the `TargetRTS`.
4. If the source was generated on Windows NT, convert all files in the component directory to UNIX type, using a utility such as `dos2unix`.

This is especially important, if the target host does not support CRLF (Carriage Return Line Feed) line terminators.

Note: *It may be necessary to convert files in the `TargetRTS` directory, especially if some files were edited on Windows NT.*

5. From the build directory, set your environment variables appropriately for the compilation line-up.
6. Invoke the appropriate make command for this line-up.

Note: You can access a ClearCase server on UNIX with Rose RealTime clients running on both Windows NT and UNIX workstations.

Adding a Printer on UNIX

Rose RealTime on UNIX uses MainWin (a Mainsoft product that allows Windows applications to run in a UNIX environment). Special printer specification is necessary to support the PSCRIPT.

MainWin uses the PSCRIPT keyword in win.ini to specify PostScript support under UNIX, using syntax similar to the way one would use the PSCRIPT driver in Windows. Below is a typical printer-related section of a win.ini file. The win.ini entries are more or less the same for MainWin as they are for Windows. An explanation of each section follows the win.ini file lines.

[windows]

```
device=Apple LaserWriter II NT,PSCRIPT,LPT1
...
```

The device entry in this win.ini [windows] section defines the default printer. It takes the following syntax:

```
device=outputdevicename,devicedriver,portconnection
```

The keyword PSCRIPT is used in place of the devicedriver.

[ports]

```
LPT1:=lp -c "%s"
LPT2:=lp -c -dps1700 "%s"
LPT3:=
...
```

The win.ini [ports] section lists available communication and printer ports. Under MainWin, the Windows LPTn keywords are mapped to UNIX commands. In this example, LPT1 and LPT2 are mapped to the print command lp. MainWin sends all print job output to a file. The output file is then sent to the printer. The term %s tells the system to substitute the name of the PostScript intermediate output file. The term -dps1700 in the example refers to a UNIX printer named ps1700. The printer should be defined in the UNIX printcap file.

[PrinterPorts]

```
Apple LaserWriter II NT=PSCRIPT,LPT1:,15,90
Postscript Printer QMS=PSCRIPT,LPT2:,15,90
```

The win.ini [PrinterPorts] section is included for compatibility with applications that require this section. Entries are similar to those for the [Devices] block listed below. In [PrinterPorts], PostScript timeout values are appended after the device name. The timeout values are not used by MainWin.

[Devices]

```
Apple LaserWriter II NT=PSCRIPT,LPT1:
Postscript Printer QMS=PSCRIPT,LPT2:
```

The [Devices] block lists the active and inactive output devices that can be accessed by device drivers and specifies the ports to which these devices are connected. In this example, Apple LaserWriter II NT=PSCRIPT,LPT1: specifies that the printer is connected to the PSCRIPT queue connected to LPT1.



Chapter 3

Installing Rational Rose RealTime on Windows

Upgrade Information

Ensure that past releases of Rational Rose RealTime are removed from your system prior to installation. For details on your specific platform, see “Uninstalling Rational Rose RealTime” on page 111 for your specific platform.

Models created in earlier versions of Rose RealTime can be loaded directly into version 6.3. Rational Rose and ObjecTime Developer models should be converted as described in “Migrating from ObjecTime Developer 5.2/5.2.1” on page 59.

Note: *Do not attempt to load workspaces created in earlier versions of Rose RealTime, as they are not compatible with the new release.*

License keys installed with Rose RealTime releases 6.0, 6.0.1 and 6.0.2 are not valid for release 6.3. However, license keys installed with Rose RealTime release 6.1, 6.1.1, or 6.2 are valid for 6.3.

Installation Instructions

Before you install Rose RealTime, make sure that you have a supported system configuration. The system requirements are listed in “Platform and Toolchain Requirements” on page 11. A setup program is provided to facilitate installing Rational Rose RealTime on Windows NT or Windows 2000. You must have administrator privileges to install this software.

In order to perform a network install, you must have the Rational Suite installed. Refer to the *Installing Rational Suite* guide for instructions on the network install.

To install Rational Rose RealTime, follow these steps:

1. Load the Rational Rose RealTime CD into your CD-ROM drive. The **Setup** dialog box appears, followed by the Welcome dialog.
2. The **Choose Product** screen appears. Choose **Rational Rose RealTime** as the product to install.

Note: *If you are planning on using floating licenses, the FlexLM license manager needs to be installed before you install Rational. If you do not have FlexLm installed, select **Install FlexLm** and follow the prompts.*

3. Click **Next**.

Note: *If you have not configured a license for Rational Rose RealTime, a dialog box appears. To install Rose RealTime without licensing configured, click **OK**. To configure the licensing, click **Cancel** and then click **Configure Licenses**. You can configure licenses after installing Rose RealTime. For details, see “Installing License Keys” on page 41.*

The license agreement appears.

4. Click **Yes** to accept the terms and conditions of the license agreement,

Note: *You must accept the license agreement to proceed. If you do not agree with the terms of the license agreement, the installation should be aborted. All software and documentation should be returned to Rational Software.*

5. Choose either **Typical**, **Custom**, or **Compact** install.

Table 3-1 Installation Types

Type	Description
Typical	<ul style="list-style-type: none"> ■ Installs the most commonly used features for a product. ■ Use this option for standard installations.
Custom	<ul style="list-style-type: none"> ■ Allows you to add or remove product features for installation. ■ Defaults to all features in a Typical installation.
Compact	<ul style="list-style-type: none"> ■ Installs a subset of the standard configuration. May omit optional files, including online documentation or online Help. To find out which files will be installed, read the product's release notes. ■ Use this option for installations on systems with limited disk space.
Network	<ul style="list-style-type: none"> ■ Installs the required files on your system to run the program from a network location. ■ Use this option to run a program from a centrally managed location. ■ This installation type is available to users when you copy and decompress the product files to a target directory. To set up a target location for network installations, see Overview of Network Installations in the <i>Installing Rational Suite</i> manual.

Note: The network configuration option is not offered with the Rose RealTime point product installation.

6. Click **Next**.

An information summary of the shared components appears.

Update Shared Components

The **Update Shared Components** dialog box appears if the Setup program needs to update shared files or components on your system. Click **Next** to have the Setup program install these files for you or **Cancel** to install these files yourself. For additional information, see the online Help.

The Setup wizard does not recalculate the disk space required for your updated selections.

Upgrade Compatibility

The **Upgrade Compatibility** dialog box appears if you have older Rational products installed on your system. For each of the older products listed, we strongly recommend that you do one of the following:

- Upgrade it: Complete this installation, and then restart the Setup program to upgrade the listed products.
- Remove it: Complete this installation, and then remove the listed products from the system.

7. Click **Next**.

A confirmation dialog box appears, listing your selected settings and options.

8. Click **Next**.

The installation begins.

Note: *If any errors occur during installation, such as insufficient disk space or inadequate permissions, an error summary dialog is displayed.*

9. When the installation is finished, the **Setup Complete** dialog box appears, prompting you to restart your computer. We strongly recommend you click **Yes, I want to restart my computer now**.

10. Click **Restart**.

11. The installation dialog appears, indicating that your computer has been restarted.

12. Click **Finish**.

If you have not configured your product license, the License Key Administrator appears.

13. Install the License Key if required. For details, see “Installing License Keys” on page 41.

Installing on a Network Drive

If you want to install on a network drive (mounted not using UNC names), just select it as though it is a local drive.

Testing your Environment

Note: *If you only want to construct UML models and not execute them, you do not need to read the remainder of this chapter.*

You must have Microsoft Visual C++ 5.0 or 6.0 installed on your system and configured to be run from the DOS prompt to make use of the code generation and execution capabilities of Rose RealTime.

The following instructions will help you to determine whether you have Visual C++ properly installed and configured on your system.

To perform testing on your environment:

1. From the Windows **Start** menu:
 - In Windows NT, choose **Start > Programs > Command Prompt**
 - In Windows 2000, choose **Start > Programs > Accessories > Command Prompt**
2. Type **nmake** and press **ENTER**.
3. Type **cl** and press **ENTER**.

If your environment is correct, then you should see the following report errors:

```
Command Prompt
Microsoft © Windows NT ™
© Copyright 1985-1996 Microsoft Corp.

C:\>nmake

Microsoft © Program Maintenance Utility Version 6.00.8168.0
Copyright © Microsoft Corp 1988-1998. All rights reserved.

NMAKE = fatal error B1864: MAKEFILE not found and no target
specified
Stop.

C:\>cl
Microsoft © 32-bit C/C++ Optimizing Compiler Version 12.00.8168
for 80x86
Copyright © Microsoft Corp 1984-1998. All rights reserved.

Usage = cl { option... } filename... { /link linkoption... }
```

If your environment is NOT properly configured, then you will see an error similar to this one:

```
Command Prompt
```

```
C:\> nmake
```

```
The name specified is not recognized as an internal or external  
command, operable program or batch file.
```

Note: *If you get this error message, your compiler environment setup is improperly configured. There is a **vcvars32.bat** file located in the installation directory for Microsoft Visual Studio (for example, **\\Program Files\\Microsoft Visual Studio\\VC98\\Bin\\vcvars32.bat**) that lists the environment variables that need to be configured.*



Chapter 4

Installing Rational Rose RealTime on UNIX

Upgrade Information

Ensure that past releases of Rational Rose RealTime are removed from your system prior to installation. Please see “Uninstalling Rational Rose RealTime” on page 111 for your specific platform.

Models created in earlier versions of Rose RealTime can be loaded directly into 6.3. Rational Rose and ObjecTime Developer models should be converted as per “Migrating from ObjecTime Developer 5.2/5.2.1” on page 59.

Note: *Do not attempt to load workspaces created in earlier versions of Rose RealTime, as they are not compatible with the new release.*

License keys installed with Rose RealTime releases 6.0, 6.0.1 and 6.0.2 are not valid for release 6.3. However, license keys installed with Rose RealTime release 6.1, 6.1.1, or 6.2 are valid for 6.3.

If you are upgrading Rose RealTime on any of the UNIX platforms, you must do one of the following:

- Manually delete your ~/.registry directory before you run the new version for the first time
- or
- Add the “-recreate_registry” command line option the first time you run the new version.

Installation Instructions

The procedure for installing Rational Rose RealTime on UNIX is described in the following sections.

Note: *Unless specified otherwise, your system administrator will generally carry out these steps.*

For environments where there are more than one user of Rose RealTime, we strongly recommend that the main Rose RealTime files be installed on a centralized file server.

To Install Rational Rose RealTime on UNIX:

1. Insert the Rose RealTime CD into your CD-ROM drive.
2. Mount the CD-ROM device.

You are usually required to be a system administrator (root or super-user) to be able to do this. See the instructions for your particular CD-ROM drive and operating system for details.

HP-UX:

```
mount -rt cdfs /dev/dsk/c201d511 /cdrom
```

Solaris:

```
mount -rF hsfs /dev/sr1 /cdrom
```

where

```
/dev/sr1
```

is the CD-ROM device.

Note: *In Solaris the CD-ROM may be automatically mounted.*

3. From a shell window, change directory to the mounted CD-ROM device.

For example:

```
cd /cdrom
```

4. Type `cd /cdrom0` and press **ENTER**
5. Run the setup script.

```
./setup.sh
```

On HP-UX, it may be necessary to use the following command (including the quotes):

```
sh './SETUP.SH;1'
```

The Welcome Script appears.

6. Press **Enter**.

The license agreement appears and you are prompted to accept or reject the license agreement. You must accept the license agreement to proceed:

```
"Enter Y<ENTER> to Accept, R<ENTER> to Read again, or
Q<ENTER> to Quit:" Y<ENTER>
```

If you do not agree with the terms of the license, the installation should be aborted. All software and documentation should be returned to Rational Software.

7. Specify the platforms to be supported by the Rose RealTime installation.

Select all platforms to be supported by this installation. The default is no and in the example, only SUN5 was selected by typing "Y<ENTER>" at the SUN5 prompt.

```
"Which platforms would you like to be supported?
HP10          Y/N [n]?
SUN5          Y/N [n]? Y<ENTER>
Platforms to be supported:
SUN5"
```

8. Confirm the platform settings.

You are asked to confirm the platforms selected.

```
Type M <ENTER> to modify platform settings or Y<ENTER> to
accept platform settings:" Y<ENTER>
```

9. Specify the installation directory.

The script prompts you for a directory that it will copy the Rose RealTime files into. The directory name must be specified as an absolute path name. A `roseRT` sub-directory will be appended in the directory that you specify. You must have write permissions for the installation directory. If the directory does not exist, you will be asked if you would like to create it.

```
"Enter absolute installation directory path:"
/testing<ENTER>
```

10. Confirm the Rose RealTime Packages to Install.

You are asked to confirm the packages and installation directory.

```
The following 5 packages are selected for installation in
the directory '/home/tester/myname/RoseRT':
```

Package description	Size in kB	I
===== Rose RealTime Platform Independent Code	11590	
Generic On-line Documentation and HELP	43225	
Rose RealTime Unix Platform Independent Code	2245	
HP-UX 10 Toolset binaries	204194	
Solaris Toolset binaries	179834	
===== Selected size:	441088	kB
Free disk space:	2786616	kB

Type M<ENTER> to Modify installation directory path, or
Y<ENTER> to Begin installing the selected packages: Y

The installation takes several minutes. You will be prompted with
the following message:

Installation complete

11. Install the License Key. For details, see “Installing License Keys” on page 41.

Setting Up a User Workstation

Environment variables

With Rose RealTime, you need to specify environment variables. Set the environment variable \$ROSERT_HOME to the new installation directory and add \$ROSERT_HOME/bin to your path.

These lines can be added to your shell initialization file, so that they are available each time you log on.

Bourne shell (sh or ksh):

```
ROSERT_HOME=/disk/apps/Rational/RoseRT
export ROSERT_HOME //for hpux10 host replace sun5 with hpux10
ROSERT_HOST=sun5
export ROSERT_HOST
PATH=$PATH:$ROSERT_HOME/bin
export PATH
```


C shell (csh):

```
setenv ROSERT_HOME /disk/apps/Rational/RoseRT
setenv ROSERT_HOST sun5 //for hpux10 host replace sun5 with
                        //hpux10
set path=($path $ROSERT_HOME/bin)
```

Either logout and then login again, or perform the rest of the upgrade from a new command shell.

Additional settings

For additional environment variables and startup options, see the *Rational Rose RealTime Toolset Guide* .



Chapter 5

Understanding Rose RealTime Licenses

This chapter describes:

- How Licenses Work
- FLEXlm License Server
- Licensing on UNIX
- The License File

When you buy Rational Rose RealTime, you purchase a number of node-locked and/or floating licenses. A node-locked license allows you to use Rose RealTime on a specific workstation. Floating licenses allow anyone on your network to use Rose RealTime as long as a floating license is available. Thus, the number of licenses that you purchase determines the maximum number of users who can use Rose RealTime simultaneously.

For example, if you purchased five licenses and three users are currently using Rose RealTime, then two more users can use Rose RealTime.

How Licenses Work

Licenses are controlled by a license manager FLEXlm (software delivered as part of Rose RealTime) that runs on a license server (one of your workstations or a dedicated machine depending on configuration and license types purchased). The license manager monitors license access.

When you start Rose RealTime, you are initially unlicensed. If a license is available, the license manager gives you a license. You retain the license as long as you are using Rose RealTime. When you exit Rose RealTime, your license is returned to the license manager and becomes available for another user.

If no license is available, you are unable to use Rose RealTime until a license is returned by another user. An “Unable to obtain a license” message is displayed.

Note: *The inability to obtain a license may also be caused by a corrupted license file, a change to the host id (network card, IP address) or a hard disk drive replacement when a node-locked license is used on NT. Please ensure you are able to communicate with the license server through a simple ping command.*

FLEXlm License Server

The following sections provide information about the FLEXlm license server, including descriptions of the license daemons running on the server systems.

FLEXlm components

The FLEXlm license configuration includes these major components, which are described in the following sections:

- License manager daemon
- Vendor daemon
- License key file
- Application program

License manager daemon (lmgrd)

The *license manager daemon* (`lmgrd`) handles the initial contact with the client application programs, passing the connection on to the appropriate vendor daemon. It also starts, stops, and restarts the vendor daemons.

Vendor daemon

In FLEXlm, licenses are granted by running processes. There is one process for each vendor who has a FLEXlm-licensed product on the network. This process is called the *vendor daemon*. The vendor daemon keeps track of how many licenses are checked out, and who has them. If the vendor daemon terminates for any reason, all users lose their licenses. (This does not mean that the applications suddenly stop running. Users can save their work and exit safely.) Users normally regain their license automatically when `lmgrd` restarts the vendor daemon, although the applications may exit if the vendor daemon remains unavailable.

Client programs communicate with the vendor daemon usually through TCP/IP network communications. The client application and the daemon processes (the license server) can run on separate nodes on your network across any size wide-area network. Also, the format of the traffic between the client and the vendor daemon is machine independent allowing for heterogeneous networks. This means that the license server and the computer running an application can be on different hardware platforms or even different operating systems (for example, Windows NT as a server system and UNIX as a client or UNIX as a server and Windows NT as a client).

License key file

Licensing data is stored in a text file called the *license key file*. The license key file is created by the software vendor and is edited and installed by the License Key Administrator. It contains information about the server nodes and vendor daemons, and at least one line of data (called FEATURE or INCREMENT lines) for each licensed product. Each FEATURE line contains a license key based on the data in that line, the *hostids* specified in the SERVER lines, and other vendor specific data.

In some environments, you can combine the licensing information for several vendors into a single license key file. The FLEXlm default location is:

```
/usr/local/flexlm/licenses/license.dat (Unix)
```

You will not typically need to set the `LM_LICENSE_FILE` variable in order to use Rational software products. We strongly recommend that you keep a copy of the license key file in the vendor's 'default' location, or that users do not need to set `LM_LICENSE_FILE` to run their applications. The `LM_LICENSE_FILE` variable is used on UNIX systems and on systems with multiple license key files.

Application program

The application program using FLEXlm is linked with the program module (called the FLEXlm client library) that provides communication with the license server. On Windows, this module is called `LMGRxxx.DLL`, where `xxx` indicates the FLEXlm version. During execution, the application program communicates with the vendor daemon to request a license.

License activation process

When you run a 'counted' FLEXlm-licensed application, such as a Rational Suite product that uses a floating license, the following occurs:

1. The license module in the client application finds the license key file, which includes the host name of the license server node and port number of the license manager daemon, `lmgrd`.
2. The client establishes a connection with the license manager daemon (`lmgrd`) and specifies the appropriate vendor daemon.
3. `lmgrd` determines which machine and port correspond to the master vendor daemon and returns that information to the client.
4. The client establishes a connection with the specified vendor daemon and sends its license request.
5. The vendor daemon checks in its memory to see if any licenses are available and sends a grant or denial back to the client.
6. The license module in the application grants or denies use of the feature, as appropriate.

'Uncounted' features, where the number of licenses is '0' (zero), do not require a server and the FLEXlm client library routines in the application grant or deny usage based solely upon the license contents. Node-locked licenses, for example, set the license number to 0 (zero).

Licensing on UNIX

Running the LMGRD from a Command Prompt

From a command prompt execute:

```
lmgrd -c <licenseFileList> -l <logfile>
```

Note: *lmgrd* can be found in `$ROSERT_HOME/bin/<arch>`, where `<arch>` is the host that Rose RealTime is installed on (*sun5* or *hpux10*).

- **licenseFileList** is the path to the license file or a list of license files. If the FLEXlm daemon is only being used to provide Rose RealTime licenses, use `-c $ROSERT_LICENSE_FILE`. Otherwise, include the `$ROSERT_LICENSE_FILE` environment variable in a semicolon (“;”) separated list.
- **logfile** is the path to a log file. `$ROSERT_HOME/license/log` is recommended if *lmgrd* is only providing Rose RealTime licenses.

For convenience, you will probably want to augment a system initialization script on your license server to automatically start the license daemon each time the license server boots.

The names, locations, organization, and contents of system initialization scripts varies from UNIX system to UNIX system. You might begin by looking at the following files:

- HP-UX: `/sbin/init.d/SlmRational.sh`
- Solaris: `/etc/rc2.d/SlmRational.sh`

To verify that your license manager is operational, you can enter these commands on your license server to see if its daemons are running:

```
% ps axw | grep -v grep | egrep "lmgrd|rational"
```

or

```
% ps -e | grep -v grep | egrep "lmgrd|rational"
```

Example

```
setenv ROSERT_LICENSE_FILE /apps/licenses/rrt6.0/license.dat
lmgrd -c $ROSERT_LICENSE_FILE -l /apps/logs/logRRT
```

or

```
lmgrd -c $ROSERT_LICENSE_FILE;$LM_LICENSE_FILE -l
/apps/logs/current_log
```

Administration commands

The license manager supports several system-administration commands.

Command	Description
lmdiag	Allows you to diagnose problems when you cannot checkout a license.
lmdown	Shuts down license and vendor daemons.
lmhostid	Reports license manager host ID of workstation
lmremove	Returns specific licenses to license pool (for example, after a workstation crashes).
lmreread	Rereads license file, starts new vendor daemons.
lmstat	Reports status on daemons and feature usage.
exinstal	Reports on licenses in license file you specify on the command line.

Note: These commands can be found in `$ROSSERT_HOME/bin/<arch>`, where `<arch>` is the host that Rose RealTime is installed on (`sun5` or `hpux10`).

The License File

The FLEXlm license files are the same format for UNIX and Windows.

The default UNIX Rose RealTime license file is:

```
$ROSSERT_HOME/license/license.dat
```

When users start Rose RealTime, the startup script automatically defines the environment variable `ROSSERT_LICENSE_FILE` for them.

However, as you install Rational products, you can merge the Rational license data into another license file that you have already set up for another product, as long as you change the `ROSSERT_LICENSE_FILE` environment variable to point to the new file.

FLEXlm uses this variable to locate the license file.

Format

The license file is a text file that you can edit with any text editor. Your license file will contain lines similar to:

```
SERVER garcon 1874350 1706
DAEMON rational
FBE669014E142A4CF37 " "
```

In general, one or three server lines are followed by one or more vendor daemon lines, which are followed by one or more feature lines. Rose RealTime requires only one of each, but your license file may include data for other products.

Each server line contains:

- Keyword SERVER
- Host name of the license server, from hostname
- License manager host ID of the license server, from lmhostid
- TCP port number to use

Each vendor daemon line contains:

- Keyword DAEMON
- Name of the vendor daemon (always rational for Rose RealTime)
- Pathname to the directory that contains the executable code for this daemon
- Pathname to your options files for this daemon (optional)

Each feature line contains:

- Keyword FEATURE
- Name of the feature
- Name of the vendor daemon, previously defined on a DAEMON line, that serves this feature (always rational for Rational products)
- Latest (that is, highest number) version of this feature that is supported (5.000) for the current release of Rose RealTime
- Expiration date. This is specified as 'dd-mmm-yy' or as 'dd-mmm-yyyy', where 'yy' is the last 2 digits of the year and 'yyyy' is the unabbreviated year. You must specify 4 digits for the year 2000 and beyond. You must specify '00' to indicate a license which does not expire.
- Number of licenses

- Encryption code (obtained from Rational for Rose RealTime)
- Vendor string, enclosed in double quotes, contains node-locked information when licensing Rose RealTime as node-locked
- License manager host ID, supplied only when this feature is bound to a specific host (that is, node-locked)

Note: *You cannot combine floating and node-locked licenses for the same product in a single license file.*

The tokens on each line can be separated by any amount of white space (spaces or tabs). You can edit only four kinds of tokens in the license file:

- Host names on SERVER lines
- TCP port numbers on SERVER lines
- Pathnames to vendor daemons on DAEMON lines
- Pathnames to options files on DAEMON lines

All other tokens are included as input to the encryption algorithm that produces the encryption codes on the FEATURE lines.

Note: *A DEMO FEATURE Line (includes “DEMO” at the end of the FEATURE Line) is a special temporary license which does not require running lmgrd or start_lm. Licensing is activated when the DEMO FEATURE Line is placed in the license file.*



Chapter 6

Installing License Keys

For specific information on license keys please refer to the Installation Instructions and License Certificate that accompany the product shipment. If either of these two documents is missing, please contact Rational License Support for replacement information. See “License Support Contact Information” on page 5.

Note: *If you are installing licenses on a UNIX platform, **do not** follow the instructions on the Rational Start-up License Certificate or on the envelope in which the certificate is delivered to you.*

Before you begin, ensure that you know the name of your license server. You will be prompted for the server name during the installation.

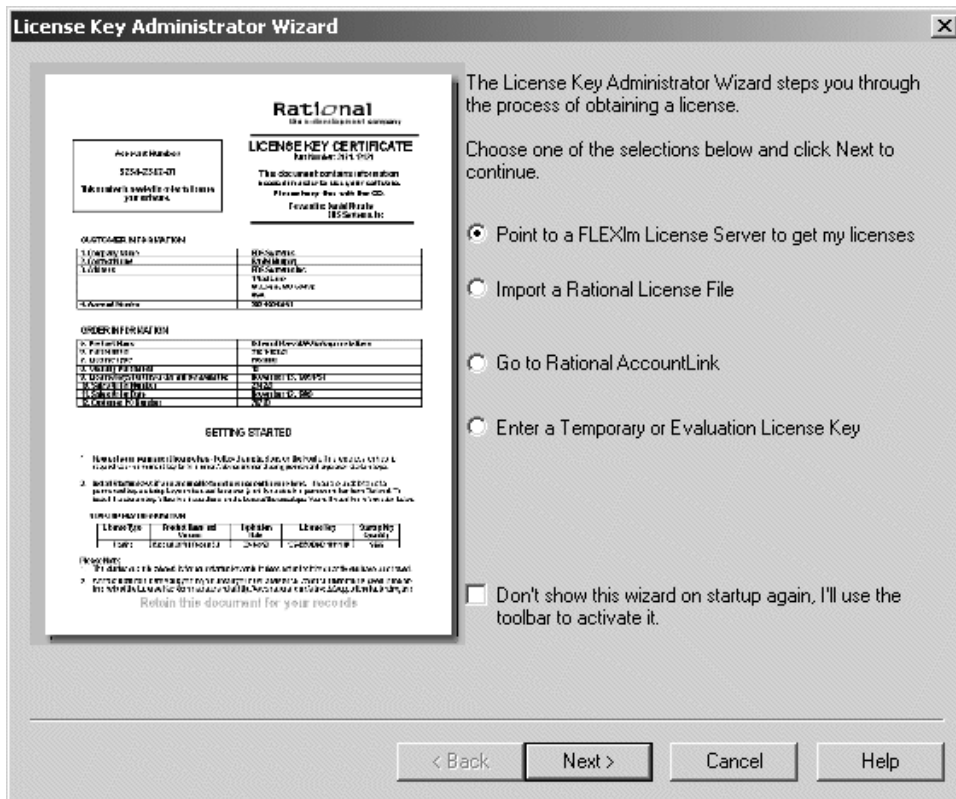
You can install Rational license keys before or after you install a Rational product. If you want to install a license key before you install a Rational product, open the Rational License Key Administrator by clicking the **Configure Licenses** button in the **Choose Product** dialog box. Use the Rational License Key Administrator Help or see the *Administering Licenses for Rational Software* manual for information about requesting and installing license keys.

Installing a Startup or Permanent License on Windows

The License Key Administrator (LKAD) lets you install startup or permanent license keys, as required. The startup license keys are time-limited and allow you to start using Rational Rose RealTime immediately.

After the Rational Rose RealTime product installation is complete, the LKAD wizard appears.

Figure 2 License Key Administrator (LKAD) Wizard



To obtain a license key:

1. Do one of the following:
 - To install a temporary license key, select the **Enter a Temporary or Evaluation License Key** option.
 - To obtain a permanent license key, select one of the other options.
2. Follow the prompts in the wizard after you have chosen your option.

If you choose **Request a license using Rational AccountLink on the World Wide Web**, your web browser opens and takes you to the AccountLink web site:

<http://www.rational.com/accountlink>

We recommend that you bookmark this site. You will need to access AccountLink when you are ready to obtain a permanent license.

Installing a Permanent License on Windows

To install a permanent license key:

1. Open the Rational Rose RealTime AccountLink web site:

www.rational.com/accountlink

2. Click **Get License Key(s)**.

AccountLink prompts you to enter your account information.

3. View your company's License Key Certificate and enter your Rational account number found on this certificate.

Note: *If you are unable to find your Rational account number, contact Rational License Support. See "License Support Contact Information" on page 5*

4. Click **Next**.

AccountLink prompts you to specify the license type.

5. To Select a license type, do one of the following:

- Click **NodeLocked** to obtain a license for a client install.
- Click **Floating** to obtain a license for a server install.

6. Select the product line **Rose RealTime**.

7. Select the product name **Rational Rose RealTime for Windows**.

8. Enter the required quantity of licenses.

Note: *For node-locked licenses, the quantity can only be "1".*

9. Click **Next**.

AccountLink prompts you to enter your Host Name and Host ID.

10. Enter your Host Name and Host ID.

11. If you do not know this information, you can request it from AccountLink:

- Select **Windows operating system** from the scroll down list.
- Click **Download**.

The **File Download** dialog box appears, prompting you to open the file from its current location or to save the file. We recommend that you open the file, to import it to disk automatically.

- Click **OK**.
 - A dialog box appears containing the Host Name and Host ID.
 - Copy the Host Name and Host ID from the dialog box.
 - Paste the contents into the Host Name and Host ID fields.
12. Select the platform on which the toolset will be running.
 13. Click **Next**.
 14. Enter the contact information.
 15. Click **Next**.
 16. Verify the information:
 - If the information is correct, click **Submit**.
 - If the information is NOT correct, click **Modify email**. Correct the information as required, then click **Submit**.

Note: An email message will be sent to the inbox for the email address which you submitted.

Installing the License Key

To install the license key:

1. Double-click the attached .upd file.
A dialog box appears prompting you to save the file to disk or open the file.
2. Click **Open** and then click **OK**.
The LKAD **Confirm Import** dialog box appears.
3. Click **Import**, then click **OK**.

Installing a Floating License Key on a UNIX server

To install a floating license key on a UNIX server:

1. Obtain the license key as outlined in “Installing a Permanent License on Windows” on page 43.
2. Set the **Host Name** and **Host ID** to be the **UNIX license server**.
3. Save the .upd file as **license.dat** in a directory on your UNIX system where you would like to maintain your licensing information.

We recommend you save this file to the \$ROSSERT_HOME/license directory.

Note: Do not overwrite any existing **license.dat** files you may currently have in this directory.

4. FLEXlm v6.0i or greater and the rational daemon are both required on the UNIX machine. If either of these is not available, they can be downloaded from our ftp site at:
ftp://ftp.rational.com/public/tools/flexlm
5. Activate the new licenses with the FLEXlm software. For information about the FLEXlm license manager, see “FLEXlm License Server” on page 34, or refer to the FLEXlm documentation.
6. Using the License Key Administrator, set your license server using the **Settings - Service Configuration** menu.

Installing a Startup or Permanent License on UNIX

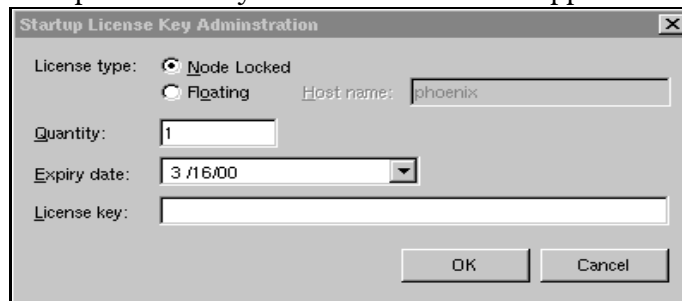
The startup license keys are time-limited and allow you to start using Rose RealTime immediately.

Installing a Startup License on UNIX

To install a startup license on UNIX:

1. Go to the \$ROSSERT_HOME/bin directory.
2. Type **RoseRT -startuplicense**.

The Startup License Key Administration form appears.



The screenshot shows a dialog box titled "Startup License Key Administration". It contains the following fields and controls:

- License type:** Two radio buttons are present: "Node Locked" (which is selected) and "Floating".
- Host name:** A text input field containing the value "phoenix".
- Quantity:** A text input field containing the value "1".
- Expiry date:** A dropdown menu showing "3 /16/00".
- License key:** A large empty text input field.
- Buttons:** "OK" and "Cancel" buttons are located at the bottom right of the dialog.

Locate the Startup License Key certificate that accompanied your product shipment.

3. Based on the license type and product name indicated on this certificate, copy the appropriate information into the Startup License Key Administration form, and click **OK**.

Note: A floating license requires you to start the license server. See “Understanding Rose RealTime Licenses” on page 33.

Your startup license is created. Remember that your Startup license will expire on the date listed on the certificate. You will have to request and install permanent license keys before this expiry date.

Now you are ready to start Rose RealTime.

Installing a Permanent License on UNIX

Licenses are obtained from the Rational website, using AccountLink. After obtaining the license(s), they need to be installed on Rational Rose RealTime.

To install a permanent license on UNIX:

1. Visit the Rational Rose RealTime AccountLink web site:
www.rational.com/support/accountlink
2. Click **Get License Key(s)**.
AccountLink prompts you to enter your account information.
3. View your company’s License Key Certificate and enter your Rational account number found on this certificate.

Note: If you are unable to find your Rational account number, contact Rational License Support. See “License Support Contact Information” on page 5.

4. Click **Next**.
AccountLink prompts you to specify the license type.
5. To select a license type, do one of the following:
 - Click **NodeLocked** to obtain a license for a client install
 - Click **Floating** to obtain a license for a server install
6. Select the product line **Rose RealTime**.
7. Select the product name **Rational Rose RealTime for UNIX**.
8. Enter the required quantity of licenses.

Note: For node-locked licenses, the quantity can only be “1”.

9. Click **Next**.
AccountLink prompts you to enter your Host Name and Host ID.
10. Enter your Host Name and Host ID.
11. If you do not know this information, you can request it from AccountLink:
 - Select **UNIX operating system** from the scroll down list.
 - Click **Download**.
The **File Download** dialog box appears, prompting you to open the file from its current location or to save the file. We recommend that you open the file, to import it to disk automatically.
 - Click **OK**.
 - A dialog box appears containing the Host Name and Host ID.
 - Copy the Host Name and Host ID from the dialog box.
 - Paste the contents into the Host Name and Host ID fields.
12. Select the platform on which the toolset will be running.
13. Click **Next**.
14. Enter the contact information.
15. Click **Next**.
16. Verify the information:
 - Click **Submit** if the information is correct.
 - Click **Modify email** if the information is NOT correct, . Correct the information as required and then click **Submit**.

***Note:** An email message will be sent to the inbox for the email address which you submitted.*

Installing the License Key

To install the License Key:

1. Save the attached .upd file as:
\$ROSSERT_HOME/license/license.dat
2. Do one of the following:
 - To integrate Rose RealTime with other Rational products, see “Integration With Rational Suites Licensing” on page 48.

- To not integrate Rose RealTime with any other Rational products, see “FLEXlm License Server” on page 34, to initially set up FLEXlm and activate your new keys.

Integration With Rational Suites Licensing

If you are using other Rational products with Rose RealTime, the license.upd file that you receive from Rational in response to a license request will contain the keys for all the Rational products. If you are using floating licenses, you will already be using the FlexLM lmgrd daemon and the rational vendor daemon.

Rose RealTime assumes that the ROSERT_LICENSE_FILE variable points to a valid FlexLM license file that contains a valid Rose RealTime license. If you follow the instructions provided, the existence of the additional license keys will not cause any problems.

Note: *Only one instance of the rational daemon can be executed at any given time for floating licenses. Your project’s license administrator should ensure that only one instance of the rational command exists and/or all paths are set correctly so that only one instance of the rational command is used.*

For additional information on integration with Rational Suites Licensing, see the *Installing Rational Suite Guide*.

Troubleshooting

Windows

Problem 1

If a FlexLM License Manager dialog appears indicating that “Your application was unable to obtain a license because...”, do the following:

1. Click **Cancel**.
You will get a Rose RealTime message stating “Unable to obtain a license”.
2. Click **OK**.
3. Run the **LMTools** application, located in:
C:/Program Files/Rational/CommonLM

4. Verify that **FlexLM** is pointing to the correct license file.

Problem 2

If you receive an “Unable to obtain a license message” message after the splash screen is displayed, check the expiration date of your license.

Problem 3

If you have received a floating license file and are unable to obtain a license, verify that the license daemon is running. See “Installing a Floating License Key on a UNIX server” on page 44.

UNIX server

Note: *This section applies only if you are installing a floating license on a UNIX server.*

Problem 1

If a FLEXlm License Manager dialog appears indicating that “Your application was unable to obtain a license because...”:

1. Ensure that your Windows setup is correct.
2. Ensure that your UNIX server is set up correctly. For information on setting up your UNIX server, see “Understanding Rose RealTime Licenses” on page 33, or refer to the FLEXlm documentation.

Problem 2

If you receive an “Unable to obtain a license message” message after the splash screen is displayed, check the expiration date of your license.

Problem 3

If you have received a floating license file and are unable to obtain a license, verify that the license daemon is running. “Installing a Floating License Key on a UNIX server” on page 44.

UNIX

Problem 1

If a FlexLM License Manager dialog appears indicating that “Your application was unable to obtain a license because...”, do the following:

1. Click **Cancel**.
You will get a Rose RealTime message stating “Unable to obtain a license”.
2. Click **OK**.
3. Verify the location and naming of the license file:
 - If you do have the **ROSERT_LICENSE_FILE** variable set, verify that the variable set matches the actual location and file name, by typing the following in a command prompt:

```
echo $ROSERT_LICENSE_FILE
```
 - If you do NOT have the **ROSERT_LICENSE_FILE** variable set, the default location of the license file is **\$ROSERT_HOME/license/license.dat**. Ensure that you have named this file correctly.
 - If you are incorporating this file into an existing FLEXlm license file, see “Understanding Rose RealTime Licenses” on page 33, or refer to the FLEXlm documentation, to ensure that the setup and key activation was done correctly.
4. If both the name and location are correct, verify that the install process has set the **ROSERT_LICENSE_FILE** environment variable to the location of the file. This variable points to the **license.dat** file, not just the directory in which it is located.

To verify that the environment variable is correctly set, type **echo \$ROSERT_LICENSE_FILE** at a command prompt.

If the environment variable is not set or set incorrectly, add or modify as appropriate.

Problem 2

If you receive an “Unable to obtain a license” message after the splash screen is displayed, check the expiration date of your license.

Problem 3

If you have received a floating license file and are unable to obtain a license, verify that the license daemon is running. See the “Installing a Floating License Key on a UNIX server” on page 44.



Chapter 7

Migration

This section provides help for users wanting to migrate models from Rational Rose, ObjecTime Developer, or previous releases of Rational Rose RealTime.

- “Migrating from Rational Rose” on page 53
- “Migrating from ObjecTime Developer 5.2/5.2.1” on page 59
- “Migrating from Rose RealTime 6.0/6.0.1/6.0.2/6.1” on page 62

Migrating from Rational Rose

The Rose RealTime interface is similar to Rose; however, there are some subtle differences that Rose users should understand before using Rose RealTime.

User Interface Differences

If you are familiar with Rose, you should not have too much trouble understanding the Rose RealTime user interface. Rose RealTime has maintained the same architecture as Rose and has preserved the main toolset features: a model browser, diagrams, model properties, add-ins, and an extensibility interface (RRTEL).

Note: *Some of the icons have been modified but they have remained intuitive.*

However, to support modeling real-time systems, to allow full code generation, and to provide an executable interface, you will notice the following main changes to the Rose RealTime interface. (For a complete description of the Rose RealTime user interface please refer the *Rational Rose RealTime Toolset Guide* available from the online Help.)

Multiple model browsers

The model browsers in Rose RealTime have three views: the Model View, the Containment View, and the Inheritance View. Each view displays the elements in your model from different perspectives.

In addition, you can create multiple model browser windows by selecting **View > Browsers > Create New Browser**.

Output windows

In Rose, the log is in an undockable window that cannot be dragged onto another section or window. In Rose RealTime, the output window is dockable, and contains a set of windows that show different kinds of output from the toolset.

Code editors

In Rose, code is added to operations outside the toolset; in Rose RealTime, code is added in the tool. Code is added to model elements through their specification dialogs. For example, the **Details** tab of an Operation specification contains a Code window in which you can write the body source code of the operation.

Code can also be added to capsule state diagrams.

Code browser

During the development of a model, you spend considerable time writing source code. In Rose RealTime, you can edit the code for the currently selected element in the code window, rather than having to open the element's specification dialog.

Layout tools and line styles

Rose RealTime allows you to perform advanced layout operations on diagrams. For example, you can align, change the view spread, and make elements the same size. You can also configure the way lines are drawn:

Figure 3 Layout menu — right-click on any diagram

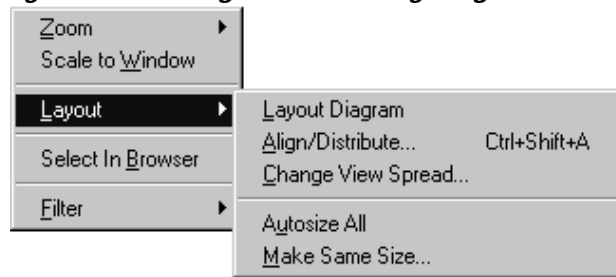
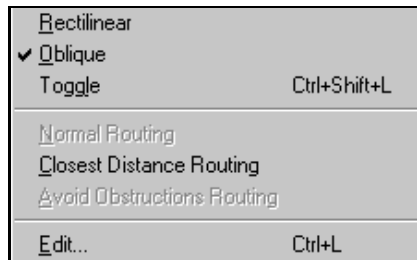


Figure 4 Line attributes menu - Edit > Line Attributes



New Modeling Language Elements

Rose RealTime introduces new modeling elements—capsules, protocols, and ports—and a new diagram—the structure diagram. The *Rational Rose RealTime Modeling Language Guide* contains information about the new modeling elements, as well as a summary of the real-time specializations to the UML.

You can also review the Concept Tutorials.

Code Generation, Building, and Running

An important difference between Rose and Rose RealTime is the support for building and executing models from within the toolset. Note the following:

- Rose RealTime is not meant to be used in a round trip process. The model contains all the information required to generate, build, and run elements in the model.
- Rose RealTime does not ship with a compiler for your target environment. You must install and configure a compiler for your target. Rose RealTime will use that compiler to build the model.

For more information, see the *Rational Rose RealTime Toolset Guide*, available through the online Help.

Opening Models from Rational Rose

Rose RealTime can open files saved with Rational Rose 98, 98i, and Rose 2000 (.mdl files).

Fixing unresolved references

When importing a model from Rose 98, 98i, or Rose 2000 into Rose RealTime, you should fix any model errors in Rose (**Tools > Check Model**) before trying to import the model. In particular, it is important to resolve any unresolved references. Rose is not concerned with unresolved references; however, they are very important in Rose RealTime as they can result in incomplete code generation and compilation errors.

For more information, see “Model Validation” in the *Guide to Team Development*.

Tasks

To open a Rational Rose model in Rose RealTime:

1. Select **File > Open** and choose Rose Model (.mdl) from the **Files of Type** pull-down menu.
2. Select a file and click **Open**.

Files from Rose versions older than Rose 98 have to be opened in Rose 98 and saved first.

Note: *Opening a new model discards any existing model that you have. The tool prompts you to save changes first.*

List of Importation Log Messages

The following messages may appear in the Log after a Rose98 model has been imported.

Message: Warning: Renamed elementClass “oldElementName” to “newElementName”.

Description: A loaded model element has been renamed to conform with Rose RealTime's naming requirements. Double-clicking on the warning in the log, this may display the renamed element.

Message: Error: Unresolved reference from ... to ... by ...

Description: The toolset was unable to resolve a reference between two model elements. This is usually the result of loading an incomplete model, for example, when the user has updated only part of a model from CM. The rest of the model needs to be loaded in order for the reference to be resolved. However, in some cases, the unresolved model element is removed from the model and the deletion is recorded in the log window.

Message: Error: Error reading file fileName at line lineNumber or Error message detail.

Description: The error message detail may contain validation errors originating from the internal meta-model. Possible error message details that originate from the petal reader are listed below.

Message: Invalid syntax.

Description: The file contents cannot be read by the toolset. The user should send the file to customer support with a description of what they were doing when the file was created. For example, if you import a Rose98 model and make some changes to the Component View, the file will not reload in Rose RealTime.

Limitations and Restrictions

When a Rose model is opened in Rose RealTime, the following elements are not converted:

- State diagrams and Activity diagrams

The model opens, but the state diagrams and activity diagrams are not present in Rose RealTime.

- Importing Rose models containing controllable units is not supported.

If the Rose model file contains controllable units, you should export the model from Rose98 into a single .ptl petal file (**File > Export Model**), that can then be opened with Rose RealTime (**File > Open**, and select **All Files...** in the combo box to display .ptl files).

- Three-tier class diagrams are not supported in Rose RealTime.

If the Rose model file contains a three-tier class diagram, you should create a copy of the Rose model that does not contain a three-tier diagram to import into Rose RealTime.

Note: *The conversion of models is supported in one direction only: once models are brought into Rose RealTime, if they are converted back to Rose, the additional Rose RealTime functionality will not appear in Rose. Working in a mixed Rose RealTime/Rose environment is not supported. Generated code is not compatible between the two tools.*

Importing Rational Rose Generated Code

Source code that has been generated from a Rose model and has been edited within the preserved regions may be imported.

To import Rose generated code:

1. Verify that the Rose .mdl file is not newer than the generated code. If so, regenerate the code.
2. Open the Rose model.
For details, see “Opening Models from Rational Rose” on page 56.
3. Choose **Tools > Import Code...**

If code was generated from this model using Rational Rose and the model was saved after the code generation was performed, a “Rose Code Import” window appears. Otherwise, a “There are no cpp or h files available for import” message is displayed.

The Rose Code Import Window lists all the .cpp and .h files that were generated from the model and lets you select all or a subset of the files. It also displays the classes that will be affected by each file that is selected. Once a file has been imported it will not be listed if code importation is repeated.

4. After you have complete importation and are satisfied with the results, save the model.

Limitations and restrictions

- No action will be taken on empty preserved regions. As a result, constructors, destructors and operators that are generated by Rose and have empty preserved regions, will not be added to the model.
- Use of the Code Name properties for classes and operations can cause inconsistent naming in the generated code. The inconsistencies can cause compile time errors, which can be resolved manually.

Migrating from ObjecTime Developer 5.2/5.2.1

Users migrating from ObjecTime Developer can open their models in Rose RealTime. First, see the *ObjecTime Developer Conversion Guide* to get your ObjecTime Developer model loaded and built in Rose RealTime.

Terminology

The modeling language and toolset terminology in Rose RealTime is different than that used in ObjecTime. This section provides an overview of the changes.

Actor/binding/protocol class

Rose RealTime supports the UML modeling language. Therefore, certain modeling elements are referred to by UML standards differently than they are in ROOM (Real-Time Object-Oriented Modeling). For detailed information regarding the UML modeling elements supported in Rose RealTime, see the *Modeling Language Guide*.

Table 7 Terminology mappings from ROOM to UML

ROOM	UML
actor class	capsule
actor reference	capsule role
protocol class	protocol

ROOM	UML
port	port
SAP/SPP	unwired ports
binding	connector

Context/update

In ObjecTime Developer, contexts contain a group of related actors, protocols, and data classes. In Rose RealTime, models are stored in controlled units that can vary in granularity. For example, the whole model can be stored as a single controlled unit (default) or each element can be stored individually. If a model is stored as one controlled unit, then the model file (.rtmdl) contains all information about a model. If the model file is read-only, then when the model is opened in Rose RealTime it is also read-only.

Activation/passivation

These terms have been replaced by more commonly used open and save. You open a model into Rose RealTime, and save it to disk.

For more information, see the *Toolset Guide*.

Workspace browser

In ObjecTime Developer, workspace browsers showed all activated contexts and updates. Since Rose RealTime only supports one model loaded at a time, there is no equivalent concept.

The workspace in Rose RealTime is associated with a specific model and is saved as such. The workspace can be stored under Configuration Management, if desired.

Model browser

Rose RealTime still has a model browser. You can, however, have more than one browser for a model, and each browser shows the model from three different views: the Model View, the Containment View, and the Inheritance View.

For more information, see the *Toolset Guide*.

Project files

Project files do not exist in Rose RealTime. An equivalent concept is the model file (.rtmdl) that contains references to a set of packages, but does not contain version information. Rose RealTime does not manage versions of files. Instead the model file loads the packages it finds on disk. It is up to the developer, through their configuration management process, to ensure that the files on disk are the correct version.

Library browser

Library browsers do not exist in Rose RealTime. Because of the changed underlying model representation, the configuration management integration has changed significantly in Rose RealTime.

It is highly recommended that you read the *Guide to Team Development* for a detailed introduction to using source control with Rose RealTime.

User Interface Differences

For a complete description of the Rose RealTime user interface, please refer to the *Toolset Guide*. Rose RealTime looks very different than ObjecTime Developer. Although you can accomplish almost everything you can in ObjecTime Developer, the steps and mechanics are very different. For this reason, it is recommended that you review the tutorials to become familiar with the interface.

Note: *When using Rose RealTime, everything is right-click-centric, meaning that you can right-click on every element in the toolset to show a context-menu that contains actions that you can perform.*

Property editors

Property editors have been replaced by specification dialogs. Every modeling element has a specification dialog that contains a non-graphical view of its properties. To access an element's specification, right-click on the element (in either the browser or on a diagram) and select **Open Specification**.

List headers

In ObjecTime Developer, every window has a list header in which you can access menu items specific to that window. In Rose RealTime, these have been replaced by right-click menus and the main application menu.

State and structure diagrams

To open a state or structure diagram, right-click a capsule, and click **Open Structure Diagram** or **Open State Diagram**. The state and structure diagram editors appear in the same window. You can switch between one and the other using the tabs at the bottom of the window. If you want to see the structure and state diagrams simultaneously, click and drag one of the tabs away from the window. This undocks the diagram and creates a new window containing only the selected diagram. You can redock the diagrams by dragging one of the tabs into the other.

For more information, see the *ObjecTime Developer Conversion Guide*.

Compilation

In ObjecTime Developer 5.2/5.2.1, data classes were compiled one package at a time. In Rose RealTime, data classes are compiled one class at a time.

Migrating from Rose RealTime 6.0/6.0.1/6.0.2/6.1

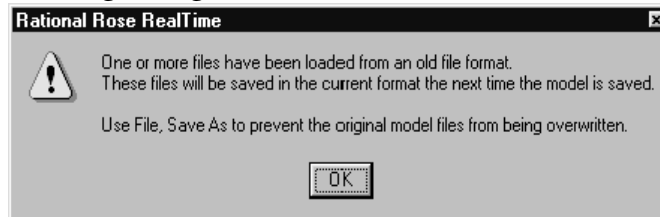
Models from these previous versions of Rose RealTime are compatible with this version. However, there are some changes in team development and language add-ins that require you to plan some changes to your model.

Note: *Beta customers must uninstall before installing the new release.*

File Format Changes

When opening a Rose RealTime 6.0 model, a dialog warns you that the next time the model is saved the files will be saved in the new file format. To prevent the original model from being overwritten, on the **File** menu, click **Save As**.

Figure 5 Warning dialog



For this reason, when working with a model under source control, you must check out all controlled units so that they can be saved in the new format.

Source Control Migration

If your model is in source control, you need to load it into the new release of Rose RealTime.

To save a file in the new file format:

1. In the 6.0 toolset, all files should be checked in, and the model should build and test successfully.
2. The source control administrator/model converter **checks out all files** from 6.0 toolset.
3. Install and start the new release of Rose RealTime.
4. Open the .rtmdl file in Rose RealTime.

Note: Do not open the workspace (.rtwks).

5. Save the model.
6. Configure the source control settings.
7. Save the Workspace.
8. Submit all changes.

Note: Migration from 6.0 is one-way. After you have migrated a model, you cannot successfully reload a controlled unit in 6.0 format. Although the toolset lets you attempt to reload a controlled unit, several errors will be reported. A mixed model is not supported.

ClearCase integration

Rose RealTime models currently stored in a ClearCase VOB should be converted to use the type manager in order to take advantage of the new integration features. A script, `cc_chtype.pl`, has been included to help in the conversion process. The script, located in `$(ROSET_HOME)/bin/$(ROSET_HOST)/cc`, produces a log of commands that will convert the existing model files from the default “text_file” type to the supplied “rosert_unit” type.

After following the setup directions detailed in the “Source Control Tools” chapter of the *Guide to Team Development*, use the following invocation from the root of your VOB to produce a batch file, which when executed will convert any Rose RealTime files to the `rosert_unit` type:

```
rtperl cc_chtype.pl -cmdfile chcmds.bat -recurse *
```

After examining the `chcmds.bat` file and verifying that the commands contained within it are the commands you want to perform, execute the batch file.

If you do not want to be queried to convert each file, add “-chargs -f” to the `cc_chtype.pl` command line before the `-recurse` argument.

```
rtperl cc_chtype.pl -cmdfile chcmds.bat -chargs -f -recurse *
```

This will generate commands that force the type change without querying.

For ClearCase users who want to use `clearmake`, there is a problem with filenames with spaces in them. For help with this, contact Technical support at:

<http://www.rational.com/support>

Migrating customized CM scripts

For complete information on library scripts and what scripts may require modification to meet your specialized CM needs, see the *Guide to Team Development*.

Language Add-in Changes

The C and C++ Language Add-Ins have changed, it is very important to read “C Language Migration” on page 68 and “C++ Language Migration” on page 71 for instructions on migrating existing models to either of these Language Add-Ins.

Note: *Rational Rose RealTime version 2001A.04.00 now also supports the Java language.*

Running Two Different Releases of Rose RealTime

Windows NT

If you need to run both versions of the tool while you are converting your models to the new release, you need to start the 6.02 release with a batchfile to reset your environment settings to the 6.02 defaults. This script is available on the Rose RealTime support website in the patches and updates section:

<http://www.rational.com/support>

Note: *Add-Ins and other product integrations may not work with the 6.0.2 release after you have installed the new release on your workstation because of the new registry settings. We recommend that you remove the 6.0.2 release from your workstation as soon as your 6.0.2 model has been converted to the new release.*

UNIX

You can set up your environments to run both releases of Rose RealTime, but do not run them from the same machine at the same time. This is a MainWin limitation.

Workspace Files

Version 6.0.x workspace files are not supported. You must open the model without the workspace. The unsupported workspace is backed up to a file.

RRTEI Changes

If you have previously used any of the following classes or functions in your scripts, they have to be removed in order for your scripts to be compatible with this new release:

- ComponentAggregationCollection class
- ComponentAggregation class
- Component::GetComponentAggregation()
- Component::AddComponentAggregation()
- Component::DeleteComponentAggregation()
- ComponentPackage::GetObject()
- RSSchedule enumeration
- Schedule rich type

If you have previously used any of the following classes or functions in your scripts, they have to be replaced in order for your scripts to be compatible with this new release. Use the model element's tool's properties. For example, The old Component::OutputPath property can now be retrieved by the "C++ Generation" OutputDirectory property from the component.

- Component::OutputPath
- Component::TopCapsule
- Component::RTSType
- Component::TargetLibrary
- Component::RTSDescription
- Component::CompilerName
- Component::CompilerLibrary
- Component::CompilerFlags
- Component::CompilerDescription
- Component::Inclusions
- Component::UserObjectFiles
- Component::InclusionPaths
- Component::LinkerName
- Component::LinkerFlags
- Component::LinkerDescription
- Component::ExecutableFileName

- Component::Platform
- Component::MultiThreaded
- Component::DefaultArgs
- Component::TargetDescription
- Component::CodeGenMakeName
- Component::CodeGenMakeFlags
- Component::CodeGenMakeOverridesFile
- Component::CodeGenMakeDescription
- Component::CompilationMakeName
- Component::CompilationMakeType
- Component::CompilationMakeFlags
- Component::CompilationMakeOverridesFile
- Component::CompilationMakeDescription
- Component::UserLibraries
- Component::UserSourceFiles
- Component::UserLibraryPaths
- Component::CodeGenMakeType
- Component::AddInclusion()
- Component::DeleteInclusion()
- Component::AddUserLibrary()
- Component::RemoveUserLibrary()
- Component::AddUserObjectFile()
- Component::DeleteUserObjectFile()
- Component::AddInclusionPath()
- Component::DeleteInclusionPath()
- Component::GetInclusionPathFlag()
- Component::AddUserLibraryPath()
- Component::DeleteUserLibraryPath()

C Language Migration

The following section provides details on migration issues specific to the C Language Add-in.

For more information on the C Language Add-in, refer to the *Rational Rose RealTime C Reference* .

Converting a C++ Model to C

You can convert a C++ model to C, however, the process is not as simple as changing the language of each model element. First, the C Services Library's API is different than that of the the C++ Services Library, meaning that all the Services Library references in the detail code must be changed. Secondly, the C Services Library does not support dynamic structure (import/deport), which may require you to re-design you model. In addition, all issues regarding conversion from regular C++ to C still apply to the conversion (for example, polymorphism is not supported in C, encapsulation is not enforced, all fields in a struct are public, and so on..).

You should decide early in the development cycle whether your project will be developed in C or C++ because changing languages in the middle of development requires a lot of work.

To convert an existing Rose RealTime model based on the C++ language:

1. Make a backup copy of the C++ model that you are trying to convert.
2. Change the language of each model element. The language setting is on the **General** tab of each element's specification dialog.

Note: *When model elements change languages, all the C++ language properties are replaced by C language properties. Therefore, any properties that have been modified are lost when the language is changed.*

3. Review the *Rational Rose RealTime C Reference* for descriptions of the new C properties and how these are to be used in your model.
4. All attribute and operations should be made public. The model will still build with them as private or protected, but the code generator will output many warnings in this regard.

5. If your C++ model depends on dynamic structure and importation, you can mimic this behavior in a C model by combining the static linkage of ports between capsules and the dynamic linkage of unwired ports. With some re-design, you can replace importation from your C++ model to use unwired ports and the `RTPort_registerAs()` and `RTPort_deregister()` functions to bind and unbind dynamically ports.
6. Convert all timing ports to C Timing, and then add a timing capsule to your model.
7. Remove all Log ports and all Exception ports.
8. When your design can be supported by C Services Library features, you can convert the syntax in your detail code.

***Note:** We recommend that you start converting a small set of capsules that can be built and tested separately before trying to convert the whole model. Iteratively modify detail code, build, and test.*
9. Update your components to C components.
10. Configure any of the build properties that are required.

ObjecTime Developer for C Migration

ObjecTime Developer for C models can be imported into Rose RealTime, compiled, and run with only minor modifications to the model. Functional updates (like a proper recall mechanism and data integration) was not provided via the ObjecTime Developer for C interface and thus will only be available via the new C UML Services Library API.

Importing models

Prior to importing a model, you should read the *ObjecTime Developer Conversion Guide* to understand important issues involved with migrating ObjecTime Developer models to Rose RealTime.

To import an ObjecTime Developer for C model into Rational Rose RealTime:

1. Set the default language to C.
2. Set the default environment to C TargetRTS through **Tools > Options > Language/Environment Tab**. This will ensure that protocol classes import as C Protocols.

3. Export and import your OTD for C model. For details, see the *ObjecTime Developer Conversion Guide*.
4. When the model has been imported, replace all ports of type Timing with type CTiming in your model.
Note: *Your triggers (on timeout) will remain valid.*
5. Update your timing service. If you have a simple timing service, to get you started, replace whatever timing capsule you had with the one available in **LogicalView::RTCClasses::TimerPackage::Timer**. You can override this later with a custom timer after you get your model working.
6. Build your target.
Note: *If you receive a **signal** is undefined build error, replace **signal** with **ROOM_Signal(port, signal)** for the given port.*

Converting global signals to local signals

A common update that may be required to some imported models involves the way the signals are now represented. In order to provide local signals, and thus the ability to build libraries without global system knowledge, more macro operations are necessary.

The only supported way of creating signals with the backwards compatible interface is with these primitives:

- `ROOM_Signal(port, signal)`, where `port` is the name of the port (unqualified with respect to the this pointer) and `signal` is the name of the signal.
- `ROOM_InSignal(port, signal)`, where the parameters are specified identically to the previous case.

In ObjecTime Developer, these macros unfortunately returned `signal`. You may have tried to optimize out the use of these macros, and used the signal name when sending messages through these services. However, this will no longer work because these macros now create a local signal (relative to the protocol class of the port). As a result, you will find compile errors when you go to build your model indicating that the signal is undeclared. Do the following:

Every call of

`ROOM_PortSend(port, signal)`

needs to be replaced with


```
ROOM_PortSend( port, ROOM_Signal( port, signal ) )
```

This change applies to all signals used in ROOM_ macros.

Timing service

The global signal **timeout** no longer exists. You need to use **Timing_rt_timeout** or use the ObjecTime Developer **RSL_Timeout()** macro that has been mapped towards **Timing_rt_timeout**.

Also, remember that these macro operations de-references the pointer for you, so all you have to do is provide the names.

C++ Language Migration

The following section provides details on migration issues specific to the C++ Language Add-in.

For more information on the C Language Add-in, refer to the *C++ Reference*.

If you are upgrading from a previous release of either ObjecTime Developer or Rose RealTime, to build and run your model in “Backwards Compatibility Mode” on page 71. Then, you can convert to the new syntax described in “Changes” on page 76.

See the *ObjecTime Developer to Rational Rose RealTime Conversion Guide*, that is available as part of the online Help system.

Backwards Compatibility Mode

An essential requirement of the C++ Language Add-in is that it allows models from previous releases to be loaded, compiled, and run with only small syntax changes to the model. Because of the scope of the changes required to the Language Add-in, most models will contain constructs that still will not compile even in backwards compatibility mode because of the increased send type checking and removal of global signals.

Note: *Global signals have been replaced by a signal number local to each protocol class defining the signal. Signals with the same name in different protocols do not share the same integer value.*

Migrating in two steps

You can plan your conversion in two steps:

1. Build your model in backwards compatibility.
2. Convert to the new syntax.

Since you retain the benefits of type safety even in backwards compatibility mode, one option would be to keep active projects in backwards compatibility and only use the new syntax on new projects.

Advantages of backwards compatibility versus changing all syntax

- Only small changes to user code is required.
- There are no run-time penalties.
- You can optionally benefit from the new message send type safety.

Disadvantages

- There are stubs generated for each protocol to allow backwards compatibility. More code is therefore generated in backwards compatibility mode.
- Compilation times are longer because there is more code to compile.

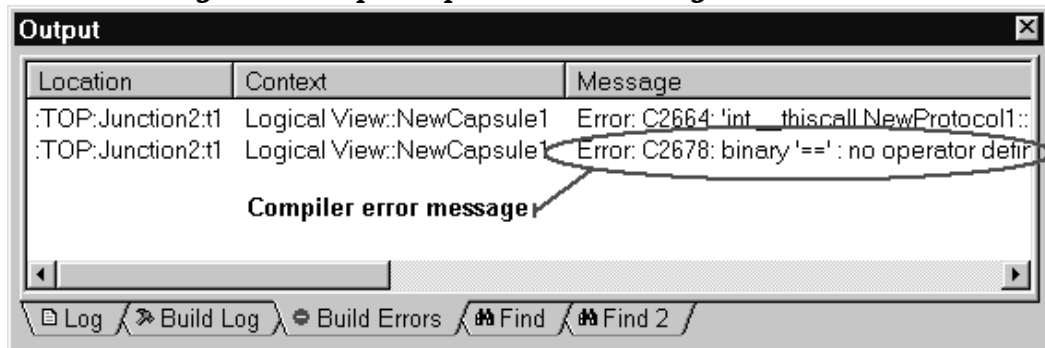
What does backwards compatibility do?

Protocols can be marked as backwards compatible (see the **C++ Target RTS** tab of the Protocol Specification). This will tell the code generator to create stub code in the protocol classes to allow use of the old Communication Services syntax.

Compiler will find all errors

Many errors in existing models will be discovered by the compiler. After a build, the **Build Errors** pane of the output window will have a list of all compile errors. Double-click on the error and the code section containing the error appears.

Figure 6 Sample output window showing build errors



Building a model in backwards compatibility mode

Follow these steps to build and run a model loaded into Rose RealTime to be built and run in backwards compatibility mode.

Step 1: Optional type checking

A flag has been added to the C++ TargetRTS tab for protocols called **TypeSafeSignals**. By default this property is turned on. Turning off the flag causes the code generator to ignore the types for all signals in the protocol class. This is the same as setting them all to blank (i.e. any). This sets the type of the data to be sent to void * and allows SEND_SCALAR to work without change. This is considered a true backwards compatibility mode with the added advantage that it affects the new send syntax as well (i.e. you can turn off backwards compatibility and turn off type safe signals).

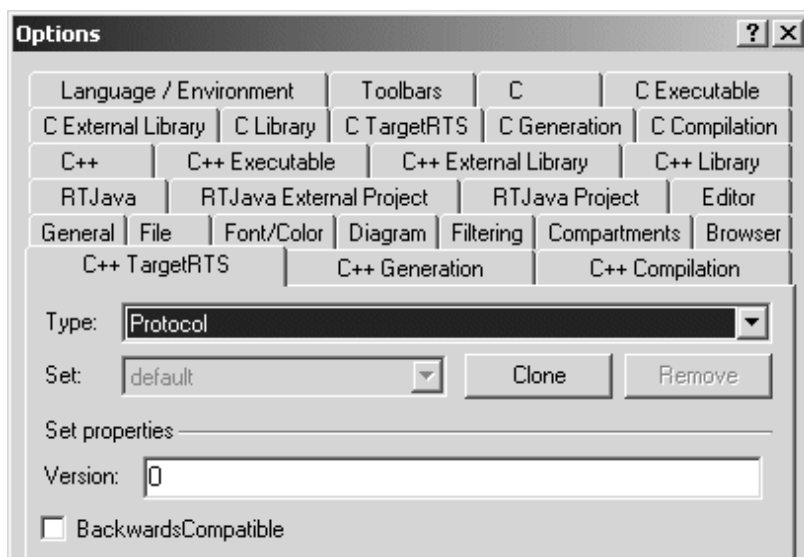
If you want to continue to use the SEND_SCALAR macro you should turn off the **TypeSafeSignals** property on these protocols.

Step 2: Enable BackwardsCompatible protocol property

- Press **F12** or select **Tools > Options** from the main menu, and in the options tab and select the **C++ Target RTS** tab. Then set the **Type** to **Protocol** and ensure that the **BackwardsCompatible** checkbox is checked.

This will ensure that all protocols default to backwards compatibility mode.

Note: On loading of ObjectTime Developer models all protocols will automatically be set to backwards compatibility mode.



Step 3: Clean up unsafe sends

Most models contain unsafe sends and sends that are not used as defined in the associated protocol. You should fix these constructs so that you do not need to debug bugs caused by these kinds of errors.

The compiler will find these errors. However if you know where you have signal-type incompatibilities, you can manually fix them.

Previous versions of the C++ UML Services Libraries allowed sending a signal, defined in the protocol to have a data class, to be sent without data. Because of the new tightened type safety of sends, this is no longer allowed and will result in compilation errors. To compile in backwards compatibility mode you will have to modify all errors of this type.

This is an example of a typical compile error for a signal-data class mismatch:

```
int __thiscall NewProtocol1::base::send(const struct
RTSignal_start &,const class AClass1 &,int)' : cannot
convert parameter 2 from 'int' to 'const class AClass1 &
```

Step 4: Remove unspecified '*' replication values

You can search your model for unspecified replication values by using the find tool and searching Cardinality/Multiplicity fields for the value '*'.

Step 5: Investigate remaining syntax changes

- The first step is to identify if you use message forwarding or if you access signal names in user code. You will have to convert these constructs as described in “Forwarding” on page 83 and “Discriminating in code the signal of a received message” on page 82.

Example compile error message when using old forwarding syntax:

```
int __thiscall NewProtocol1::base::send(const struct
RTSignal_start &,const class AClass1 *,const struct
RTObject_class *,int)' : cannot convert parameter 1 from
'int' to 'const struct RTSignal_start &'
```

Example compile error message when using signal name in user code:

```
binary '==' : no operator defined which takes a left-hand
operand of type 'int' (or there is no acceptable conversion)
```

Note: *If you still have compilation problems, review “Changes” on page 76 to ensure that you are not using classes that have been removed from the Services Library.*

Full migration

When your model is compiling and running in backwards compatibility mode, the next step for full migration is a communication service syntax change. You will have to find and replace occurrences of old syntax with the new syntax and individually turn off the BackwardsCompatibility flag on a per protocol basis. For a complete listing of the change communication service primitives, see “Changes” on page 76 section.

Changes

This section explores all the changes affecting users of the C++ Language Add-in who will be migrating their existing models to this new version.

C++ UML Services Library

Adding support for libraries and type safety required changing the Communication Service API. Review these sections to understand the new C++ Services Library changes.

- “Type safety explained” on page 77
- “New classes for protocols, signals, and ports” on page 77
- “API changes summary” on page 78
- “Macros” on page 87
- “External Layer Service (ELS)” on page 87

No attempt will be made to describe changes made to the private or undocumented features of the C++ Services Library. We recommend that you always use only the documented interfaces.

Note: For minor problems migrating customizations or configurations of the C++ UML Services Library contact Rational Technical Support. For all other problems migrating your custom changes contact your sales representative to arrange for consulting services to assist in the migration.

Code generation and compilation

Components have been expanded to allow building libraries and model external libraries.

New classes for protocols, signals, and ports

In previous versions of the Services Library **RTEndPoint** and **RTEndPointRef** classes were used to represent port instances and port references. These classes have been replaced by **RTProtocol**, **RTOutSignal**, **RTInSignal**, and **RTSymmetricalSignal** classes.

For each protocol in a model a structure is generated. Contained in the structure are a Base and Conjugate class which are subclasses of **RTProtocol**. For each signal defined in the protocol an operation is generated in the Base and Conjugate classes. The introduction of the new classes has changed the syntax of communication service operations.

Type safety explained

In a protocol specification, a signal may be defined with an associated data class. Previously, it was optionally up to the software designer whether or not to actually send data along with such signals. In addition you were able to send signals that were not defined on the port on which they were sent.

In summary, there has never been any support for compile-time validation that user code conformed to a protocol specification. Consequently all errors of this type could only be caught at run-time, resulting in developers having to track down “unexpected message warnings” and run-time exceptions.

How has this been changed?

In the new UML Services Library, you must send data if the signal has an associated data type. The data must be of the type, or a subclass of the type, specified for that signal. Alternatively, the data may be of type void or left empty. A data class type left empty (that is, no type specified) implies that you can send anything with the signal. In addition you can only send signals that have been defined on the protocol role associated with the port.

Note: *Backwards compatibility mode allows previous release syntax to be used in models compiled with the current release of the C++ Services Library.*

The **TypeSafeSignals** flag on protocols can be used to force the code generator to ignore the data class value of all signals defined in a protocol. The code generator treats the signal's data class as being empty, thus allowing any type of data class to be sent with the signal.

API changes summary

The changes affecting the communication service interface can be grouped into the following usage scenarios:

- “Asynchronous sends” on page 79 (to one or all port instances)
- “Synchronous sends” on page 80 (to one or all port instances)
- “Message reply” on page 80
- “Defer, recall, and purge” on page 81 (one or all signals to one or all port instances)
- “Port indexes” on page 82
- “Discriminating in code the signal of a received message” on page 82
- “Forwarding” on page 83 (potentially from one protocol to another and to one or all port instances)
- “RTPortRef operations” on page 85

In addition to the changes in the communication service review these issues that may impact your conversion:

- RTTimespec parameters

All examples in this section assume that a replicated port called **aPort** of type **aProtocol** is defined on a capsule. The protocol is symmetric (in and out signals are the same) and is defined as:

Signal	Data Class
start	AClass1
stop	int
reset	RTInteger

Note: The examples show sending *RTInteger* (a type of *RTDataObject* with which *ObjectTime Developer 5.2* users will be familiar), and regular classes created using *Rose RealTime 6.0*, *AClass1*.

Asynchronous sends

5.2/6.0

```
port.send(signal, rtdataobject, priority);  
port.send(signal, data, type, priority);  
port[index]->send(signal, rtdataobject, priority);  
port[index]->send(signal, data, type, priority);
```

New syntax

```
port.signal(rtdataobject).send(priority);  
port.signal(data).send(priority);  
port.signal(rtdataobject).sendAt(index, priority);  
port.signal(data).sendAt(index, priority);
```

New syntax example

```
RTInteger level(15); // RTDataObject  
AClass1 mdata(49, 1.23);  
  
aPort.reset(level).send(); // broadcast  
aPort.start(mdata).send(); // broadcast  
aPort.reset(level).sendAt(1); // single port  
aPort.start(mdata).sendAt(1); // single port
```

Synchronous sends

5.2/6.0

```
port.invoke(repbufs, signal, rtdataobject);
port[index]->invoke(repbuf, signal, rtdataobject);
port.invoke(repbufs, signal, data, type);
port[index]->invoke(repbuf, signal, data, type);
```

New syntax

```
port.signal(rtdataobject).invoke(repbufs);
port.signal(data).invoke(repbufs);
port.signal(rtdataobject).invokeAt(index, repbuf);
port.signal(data).invokeAt(index, repbuf);
```

New syntax example

```
RTInteger level(5); // RTDataObject
AClass1 mdata(49, 1.23);
RTMessage replyBuffers[5];
RTMessage replyBuffer;

aPort.reset(level).invoke(replyBuffers); // broadcast
aPort.start(level).invokeAt(1, &replyBuffer); // single port
aPort.reset(mdata).invoke(replyBuffers); // broadcast
aPort.start(mdata).invokeAt(1, &replyBuffer); // single port
```

Message reply

5.2/6.0

```
msg->sap()->send(signal, rtdataobject);
msg->sap()->send(signal, data, type);
msg->reply(signal, rtdataobject);
msg->reply(signal, data, type);
```

New syntax

```
rtport->signal(rtdataobject).reply();
rtport->signal(data).reply();
```

New syntax example

```
RTInteger level(5); // RTDataObject
AClass1 mdata(49, 1.23);

rtport->reset(level).reply();
rtport->start(mdata).reply();
```

Note: `rtport` is an argument passed to each transition code segment. It is a pointer to the port on which the triggering signal was received. For more information see “Parameters available in transition code” on page 88.

If a transition is triggered by signals arriving from different ports with different protocols, then the `rtport` argument cannot be used to reply. In these cases you will have to either explicitly cast the port or create a separate transition to reply to signals arriving on a specific port.

```
((AProtocol::Base *)msg->sap())->Ack().Send();
```

The difference between `rtport` and `msg->sap()` is that `rtport` is coerced to the correct protocol type by the code generator whereas `msg->sap()` is a pointer to the a generic `RTProtocol` object.

Defer, recall, and purge

5.2/6.0

```
port.purge(signal);
port[ index ]->purge(signal);
port.recall(signal, front);
port[ index ]->recall(signal, front);
port.recallAll(signal, front);
port[ index ]->recallAll(signal, front);
port.recallAll();
port.recallAll(0, front);
```

New syntax

```
port.signal().purge();
port.signal().purgeAt(index);
port.signal().recall(front);
port.signal().recallAt(index, front);
port.signal().recallAll(front);
port.signal().recallAllAt(index, front);
port.recall();
port.recallFront();
port.recallAt(index);
port.recallAll();
port.recallAllFront();
port.recallAllAt(index);
port.purge();
port.purgeAt(index);
```

New syntax example

```
// a signal must have already been deferred
// using a call to msg->defer().

// purge all deferred messages on all port instances
aPort.purge();

// recall all deferred by signals
aPort.by().recall();
```

Port indexes

5.2/6.0

```
msg->sap()->getIndex(); // 0-based
msg->sap()->index(); // 1-based
msg->sap()->at(index) // 1-based
```

New syntax

```
msg->sapIndex0(); // 0-based
msg->sapIndex(); // 1-based
```

Note: The `at()`, and `getIndex()` operations are no longer supported.

New syntax example

```
AClass1 mdata(1, 4.56);
int      index = msg->sapIndex0();

// send back to same port instance on
// which we just received a message.
rtport->start(mdata).sendAt(index);
```

Discriminating in code the signal of a received message

You may have code that used a signal outside the scope of a message send. For example:

```
AClass1 mdata(1,4.56);
int      index = msg->sap()->getIndex();

if( msg->getSignal() == hello )
{
    aPort.start(mdata).sendAt(index);
}
```

Since these signal values are not global you have to use the enumeration values for the signals defined in their respective protocol role. For example, you would have to change the above code fragment to:

```
AClass1 mdata(1,4.56);
int      index = msg->sapIndex0();

if( msg->getSignal() == NewProtocol1::Base::rti_hello )
{
    aPort.start(mdata).sendAt(index);
}
```

Note: The signal value in the protocol will always be called *rti_<signalname>*. You can easily reference it by using the following syntax: *Protocol::<ProtocolRole>::rti_<signalname>*, as shown above. *ProtocolRole* will be either **Base** or **Conjugate**.

Forwarding

In previous versions of the C++ UML Services Library, you were permitted to blindly forward signals out other port instances. Because signal numbers are no longer global (that is, a signal with the same name and data class in two protocols won't have the same signal number) this will no longer work.

5.2/6.0 forwarding syntax:

```
port.send(msg->signal, msg->data);
port.send(msg->signal, msg->data, msg->type);
```

Static forwarding pattern

In most cases, you can implement simple forwarding behavior by discriminating the received signal then explicitly sending a signal out another port. The outgoing signal doesn't necessarily have to be the same name as the incoming signal. Static forwarding requires signal discrimination in a transition (for example, using a switch statement) or adding transitions for each signal being forwarded.

Examples

```
// using one transition to route all
// incoming messages to other ports.

switch( msg->getSignal() )
{
```

```
    case NewProtocol1::Base::rti_start:
        output.start(*rtdata).send();
        break;
    case NewProtocol1::Base::rti_stop:
        output.stop(*rtdata).send();
        break;
    default:
        log.log("Unexpected message");
}

// or you could have one transition per
// signal. In this case each transition
// would forward one signal.

output.start(*rtdata).send();
```

Dynamic forwarding

Some routing capsules are designed so that they won't know the exact protocols for the forwarding ports at design time (that is, they could be overridden at run-time). In these cases, the switch statement described in the static forwarding pattern does not provide a good solution.

Dynamic forwarding provides run-time mapping from one protocol to another. It works by creating a signal map table to map signal numbers from one protocol to another based on the signal name and the data class. This provides constant signal lookup. In addition, signals that don't have compatible data classes are not added to the signal map.

Dynamic forwarding support has not been added to the UML Services Library. Instead a set of classes has been created that can be used in any model that requires this level of forwarding. To use dynamic forwarding please refer to the Dynamic Forwarding model example in the Examples. The example model contains the forwarding classes, or adaptors, and sample usage of these classes. In general capsules requiring dynamic forwarding will have to do the following:

1. For each port pair where forwarding will be used, an adaptor object is created to initialize and encapsulate the signal map. If you have forwarding from port A to B and A to C you will need 2 adaptor objects.
2. Each adaptor is initialized at run-time with the in and out protocols. This will create the signal map.

3. When forwarding is required in a transition, pass the message to be forwarded to the adaptor.

The example model that contains the forwarding classes (adaptors and signal maps) can be found in:

```
$ROSEXT_HOME/Examples/Models/C++/DynamicForwarding
```

RTPortRef operations

The **RTPortRef** class is no longer part of the UML C++ Services Library. Operations that used to be available on this class have been moved to the **RTProtocol** class. This is a summary of the operations that have changed going from the **RTPortRef** to the **RTProtocol** class:

RTEndPoint ** RTPortRef::incarnations()

This was last present in ObjecTime Developer 5.2. You will have to use a port (RTProtocol) paired with an index wherever a pointer to RTEndPoint appeared previously. For example, before you would have:

```
RTEndPoint ** ports = portref.incarnations();
for( int i = 0; i < portref.size(); i++ )
    (*ports) [i] ->send(ack);
```

This has to be converted to:

```
for( int i = 0; i < portref.size(); i++ )
    portref.ack().sendAt(i);
```

The valid indices are from 0 to (port.size()-1), inclusive.

RTEndPoint ** RTPortRef::incarnationsTo()

There is no direct replacement for this. Users will have to base their loop on the port index rather than an index into the returned array of pointers. Within that loop you will want to use

```
int RTProtocol::isIndexTo( int, RTActor * ) const
```

to discover the replication indices which correspond to incarnations that would previously have been included in the array. This new interface is more efficient because it avoids the need to allocate and release a block of memory.

RTEndPort * RTPortRef::incarnationTo():

This operation is replaced by `RTProtocol::indexTo()`. For example, here is a common use of `incarnationTo` and how it can be converted to use `indexTo`:

```
RTActorId aid = frame.incarnate( role1 );
RTEndPort * port = (RTEndPort *)0;
if( aid.isValid() ) {
    port = replicatedportref.incarnationTo( aid );
    if( port != (RTEndPort *)0 )
        port->send( Signal );
}
Is replaced with RTProtocol::indexTo(),
RTActorId aid = frame.incarnate(role1);
int port_index;
if( aid.isValid() ) {
    port_index = replicatedportref.indexTo( aid );
    if( port_index != -1 )
        port.Signal().sendAt(port_index);
}
```

RTTimespec parameters

ObjecTime Developer (OTD) models which used the **RTTimespec** constructor with only one parameter, as in the following code:

```
timer.informIn(RTTimespec(2));
```

will result in a compile error after conversion of the model to Rose RealTime. The compile error will appear something like:

```
..\rtg\Driver.cpp(67) : error C2440: 'type cast' : cannot
convert from 'const int' to 'struct RTTimespec'
No constructor could take the source type, or constructor
overload resolution was ambiguous.
```

The reason is that in OTD, the **RTTimespec** constructor included default arguments, that is, **RTTimespec** (`long=0, long=0`). The default constructor values are not supported on **RTTimespec** in Rose RealTime. Any code that made use of the default arguments needs to be changed to supply both constructor arguments. For example:

```
RTTimespec(2);
```

must be changed to:

```
RTTimespec(2, 0);
```


RTSignalNames

Some users have accessed this private structure to find signal names. Support for accessing this structure was never supported and has been removed from the UML Services Library. If you have referenced this structure look at replacing this functionality with the **RTMessage::getSignalName()** operation which returns the name of the signal received in the current message.

Macros

The following pre-defined macros will continue to be backwards compatible.

```
SEND_PTR( ptr )
RECEIVE_PTR( type )
SEND_SCALAR( value )
RECEIVE_SCALAR( type )
SEND_EXT( value )
RECEIVE_EXT( type )
```

External Layer Service (ELS)

In version 6.0 of the C++ Services Library the ELS was included in the pre-compiled C++ UML Services Libraries. However source code was not shipped. In the current release of Rose RealTime the ELS is not provided for use, nor supported with the release. Please refer to the IPC Application Note and Example for information on how the ELS can be replaced. The External Layer has been replaced by Rational Connexis. Further information on Add-ins, including Connexis, can be found on the Rose RealTime product web site:

<http://www.rational.com/products>

Code Generation

To support scalable build environments the C++ Language Add-in now supports the ability to break systems into a number of independently buildable components. You can now use components to build libraries, executables, and model external libraries. See “Components” on page 88. To support different component types and provide an extensible interface for components several Model Properties have been added to components.

Components

Components are collections of references to model elements that are used to build something. In Rose RealTime, there are three kinds of components:

- **C++ Executable:** produces an executable.
- **C++ Library:** produces a library file containing the object files for the classes referenced by the component.
- **C++ External Library:** does not actually produce a build output, but represents a pre-built and packaged component within a model.

The build options for each component type are stored in a set of model properties. In Rose RealTime 6.0 a component's build options were hard-coded attributes of the component. See the *Rational Rose RealTime C++ Reference* for more information about how to use the new component types.

Directory structure

The code generation directory structure has changed, it is now:

```
<component name>
  <build>
    capsule1.exe
    capsule1.obj
    ...
  <src>
    Makefile
    capsule1.dep
    capsule2.dep
    capsule1.cpp
    capsule1.h
    ...
```

Parameters available in transition code

Within each transition code segment there are two new parameters that are available.

Note: The *msg* variable is still available in transition code and capsule operations.

rtdata: This is the equivalent of the **RTDATA** macro. It is the data sent with the message cast to the data type specified in the protocol for the incoming signal. The `rtdata` parameter is cast to the lowest common superclass of the possible data classes for the given code segment.

```
int level = *rtdata;
```

Note: *Models which used RTDATA do not have to change. RTDATA and rtdata are equivalent.*

If a transition is triggered by multiple signals with different data classes, you will have to cast **msg->data** yourself.

```
int level = *(const int *)msg->data;
```

rtport: This is a pointer to the port cast to the appropriate protocol type, on which the message that triggered the transition was received. You can use this parameter to reply to messages. See “Message reply” on page 80.

Port cardinality cannot be unspecified

Because there is no way to resolve unspecified cardinalities between libraries, capsule role replication cardinalities cannot be left unspecified as ‘*’. You should use constants to specify replication values..

Makefile overrides changes

Previously the makefile override property was set to a file name which contained a makefile fragment which was to be included into the main makefiles with an **include** statement. Now the makefile overrides property is added, as is, to the makefile. That means that you don’t have to create a separate file outside of the toolset to contain any additional makefile commands.

Previous models which contain makefile overrides are converted by adding the include statement to the property.

Model Properties

Component build settings are now stored in model properties. This allows easy extensibility and sharing of build options. Although the actual build properties have not changed much, they have been re-arranged. Build options now exist for each component type and for generic generation and compilation settings.

Component type properties: C++ Executable, C++ Library, C++ External Library.

Generic build settings: C++ Generation, C++ Compilation

See the *Rational Rose RealTime C++ Reference* for descriptions of the component model properties.

Advanced property editors

A number of properties introduced in this release require more than simply a true or false value. Instead some properties represent a set of parameters. To assist configuring properties that have several parameters that can be set, graphical editors have been added to property sheets to allow editing of these complex properties. If a property has an advanced property editor you will notice an **Edit...** or **Select...** button beside the property. Press the button to access the extended property editor window.



Chapter 8

Integration Notes

Rational Rose RealTime can coexist on the same workstation with any Rational or ObjecTime product. In addition Rational Rose RealTime is shipped with “out-of-the-box” integrations with several popular development tools. It will simplify tool-chain complexity by providing teams with seamless integration to leading real-time operating systems, compilers, symbolic debuggers, and other market-leading Rational Software products. For a list of supported platform ‘line-ups’, see “Supported Host Platforms” on page 13.

Configuration Management (CM) Tools Integration

The following CM tools are supported with integration for Rose RealTime. For more information on integrating these tools, see the *Guide to Team Development*.

Tools	Version
Rational ClearCase (Base and UCM)	3.2.1 (requires patch 10), 4.0, 4.1
Microsoft Visual SourceSafe (NT only)	5.0 and 6.0
RCS (UNIX only)	5.7
SCCS (UNIX only)	5.6 on Solaris 76.1.1.1 on HPUX
PVCS Integration	6.5

ClearCase on a UNIX Server and Clients on both NT and UNIX

You can access a ClearCase server on UNIX with Rose RealTime clients running on both NT and UNIX workstations. For more information on integrating these tools, see the *Guide to Team Development*.

Migrating from Rational Rose and ObjecTime Developer

In order to migrate models into Rose RealTime from either Rational Rose or ObjecTime Developer where models were previously stored in a configuration management system, the model must be brought into the Rational Rose or the ObjecTime Developer tool and written to a single file. Please refer to “Migration” on page 53.

When importing a model from Rose into Rose RealTime, you are encouraged to resolve any model errors in Rose (**Tools > Check Model**) before trying to import the model. It is important to fix unresolved references. In general, Rose is not concerned with unresolved references; however, they are very important in Rose RealTime as they can result in incomplete code generation and compilation errors.

In order to export the ObjecTime model in a format that is readable by Rose RealTime, a patch must be applied to the 5.2 or 5.2.1 toolset to format the file in a single linear form file with all the required information. The patch is available from Rational Customer Support for both the 5.2 and 5.2.1 product release only. Please contact the Rational Customer Support group for further information.

Once the model has been imported into Rose RealTime, it can then be stored in the configuration management system.

Requirements Management Tools Integration

The following tools are supported for integration with Rose RealTime.

Tools	Version
Rational SoDA for Word (NT only)	2000.02.10 and later
Rational RequisitePro	2000.02.10 and later

Rational SoDA for Word

SoDA and Rose RealTime will work together out of the box if installed from the Suite. Rose RealTime offers the same level of SoDA integration as Rose. For information on how SoDA and Rose RealTime integrate, see the Rose integration section in the SoDA documentation.

Please refer to the product support page at <http://www.rational.com/support> for the latest updates on SoDA integration.

Note: *In order to generate a report using SoDA, the Rose RealTime model must have been saved at least once. If the Rose RealTime model has never been saved, it will be untitled. An untitled model will cause SoDA to generate errors.*

Rational RequisitePro

RequisitePro and Rose RealTime will work together out of the box if installed from the Suite. Rose RealTime offers the same level of RequisitePro integration as Rose. For information on how RequisitePro and Rose RealTime integrate, see the Rose integration section in the RequisitePro documentation.

Note: *The Rose RealTime Requisite Pro integration does not support the association of a Rose RealTime package with a RequisitePro project. Use Case and Model association is supported.*

Unit Testing Tools Integration

The following tools are supported for integration with Rose RealTime.

Tools	Version
Purify for UNIX and Windows NT	2001.03.00 and later

Rational Purify

Once a component is built and a component instance has been created, the instance can then be run and observed. Purify detects errors in your own code as well as the components your software uses. For information, see the Running and Debugging section in the *Rational Rose RealTime Toolset Guide*.

Adding options to Purify on UNIX

The toolset looks for an installation of Purify by checking for an environment variable named **PURE_HOME**. This environment variable is not set up by installing Rational Purify.

You must set this environment variable manually. The variable need not point to a directory containing Purify, nor is it required to point to a directory. The variable may contain anything, but must be set.

Occasionally, you may need to add options during a Purify'd build on UNIX. For example, Purify on HP needs to know the name of the linker or collector used by **Gnu g++**.

Options can be added by changing **PURIFY_OPTIONS** in the **CompilationMakeInsert** field of the executable component.

The default value of **PURIFY_OPTIONS** (generated in the Makefile by the code generator) is:

```
PURIFY_OPTIONS = -log-file=$(BUILD_TARGET).txt -windows=no
```

To accommodate using **g++** on HP, you can add the following:

```
PURIFY_OPTIONS = -log-file=$(BUILD_TARGET).txt -  
windows=no -collector=/usr/lib/gcc-ld -g++=yes
```

Where the path of to the collector, **gcc-ld** in most cases, should be the path that is specific to your environment.

For proper integration of Purify when running the Purify'd executable from the toolset, you should preserve the default options.

For an explanation of Purify options, see *Running a component instance with Purify* in the *Toolset Guide*.

Microsoft Development Environment

We recommend that you install the latest service packs available from Microsoft for Visual Studio or Visual C++.

Integration with Rational Robot

Installing the 2001A.04.00 release of Rational Rose RealTime will interfere with the operation of the 6.1 release of Rational Robot.

We recommend that you upgrade to the 6.3 release of Rational Robot.

Naming Directories

Avoid using spaces in directory names if you plan to integrate with Tornado, OSE or VRTX embedded operating systems. For additional information, see the Technical Notes in the Technical Support section on our web site at:

<http://www.rational.com/support>



Chapter 9

Starting Rational Rose RealTime

This section describes:

- “Starting Rose RealTime under Windows” on page 97
- “Starting Rose RealTime on UNIX” on page 97
- “Rose RealTime for UNIX and the X Window System” on page 99
- “Automating Rose RealTime” on page 101
- “Command Line Options” on page 101

Starting Rose RealTime under Windows

To start Rose RealTime on Windows, on the **Start** menu, choose **Programs > Rational Rose RealTime**.

Note: *You must first install license keys before running Rose RealTime.*

Temporary license keys can be found in the product package.

Instructions on how to request permanent license keys, see “Installing License Keys” on page 41.

Starting Rose RealTime on UNIX

Before starting Rose RealTime on UNIX, ensure your environment is set up correctly according to the instructions provided in “Installing Rational Rose RealTime on UNIX” on page 27. If not, your path may not be defined properly and the various programs required to run Rational Rose RealTime will not be found.

Rose RealTime is started from a UNIX command shell prompt by typing:

```
RoseRT
```

Start-up options for UNIX

-regedit

Edits the internal registry that maintains mappings of directory names and other information required by the Rose RealTime tool. This registry mimicks the function of the WindowsNT registry, except that on UNIX the registry is maintained directly by Rose RealTime.

-startuplicense

Creates a startup license.

-recreate_registry

Creates a default registry, throwing away any changes made through the -regedit option.

-q | -quiet

Limits the output of the tool on startup.

-v | -verbose

Provides verbose output on startup.

-cleanup

Kills all running applications using MainWin and then cleans up the x-server resources.

You should be very careful with this command as it will kill all MainWin applications running under your Id.

Rose RealTime for UNIX and the X Window System

When running on UNIX platforms, Rose RealTime relies on the X Window System to provide basic user interface services. Rose RealTime supports the most common versions of the X Window System: Version 11 Release 5 and Version 11 Release 6.

The following topics provide background information on how Rose RealTime interacts with the X Window System and highlights any specific requirements.

X clients

The X Window System employs a network-enabled client-server architecture. Rose RealTime is a client application within this architecture. X clients interact with the user via an X server which may or may not be running on the same system as the client application. If the server and client are not running on the same system, the X client is said to be using a remote display.

X servers

The X server is a program that controls interaction between the user and an X client application via the keyboard, mouse and graphical display screen. The X server runs locally on the system where the display is attached.

On UNIX workstations the X server is normally provided by the system vendor. If you want to run Rose RealTime on a UNIX workstation and remotely display it on a Windows workstation, a third-party X server (such as, Hummingbird Exceed) is required. Rose RealTime has been qualified to be used with Hummingbird Exceed 6.1.

X window managers

The X window manager is a special X application that facilitates running multiple X clients within separate windows on a single X server. The window manager provides mechanisms for resizing and moving windows and designating which X client has input focus at a given time.

Most X environments include a window manager. Rose RealTime supports most commonly used window managers including:

- Common Desktop Environment (CDE)
- Motif (MWM)
- Exceed native window manager

When available, the CDE window manager is recommended.

Input focus (active window) policy

The X window manager often allows the user to specify a policy for delegating input focus. This window is also referred to as the active window. There are two common settings:

- Click to focus. In this mode, the user must click on a window with the mouse to give it input focus. This is most consistent with the Windows focus policy and is the recommended configuration.
- Point to focus. In this mode, the user points to a window with the mouse to give it input focus.

Window order policy

The user can also often specify with the window manager whether the active window must be the top-most window displayed. Under CDE this option is called “Raise Window When Made Active”. This option should be enabled for consistency with the Windows user interface.

Notes

- CDE's window manager option “Allow Primary Windows On Top” should also be enabled.
- Exceed's Native Window Manager does not display a button for Rose RealTime in the Windows Taskbar. For this reason it is recommended that a remote CDE window manager be used instead of Exceed's native window manager. If you prefer to use Exceed's native window manager, you can use the ALT+TAB shortcut key to switch from another application to Rose RealTime.

Automating Rose RealTime

Rose RealTime can be programmed to automatically perform a wide variety of tasks through the Rose RealTime Extensibility Interface (RRTEI). The RRTEI is accessible through basic scripts and from COM automation clients. This interface can be used to create add-ins and scripts. Rose RealTime also supports the Rose Extensibility Interface (REI) for compatibility with Rose. The complete documentation for the RRTEI is included in the Rose RealTime Online Help System.

Running Rose RealTime as an automation server consumes a license when the application is made visible.

Command Line Options

The following are command line options for Rose RealTime on UNIX:

<filename>

A user option to load a model on startup.

-nologo

A user option to suppress the logo screen on startup.

-emulateREI

A user option to enable the Rose Extensibility Interface (REI). Overrides the settings in tools/options.

Note: *The Rose RealTime Extensibility Interface (RRTEI) is still available.*

-noEmulateREI

A user option to disable the Rose Extensibility Interface (REI). Overrides the settings in tools/options.

Note: *The Rose RealTime Extensibility Interface (RRTEI) is still available.*

-register or -regserver

Enters the applications registry settings into the registry.

-unregister or -unregserver

Removes the applications registry settings from the registry.

-runScriptAndQuit

Use in conjunction with a compiled script passed as parameter. When the toolset is launched with this command line option, the toolset starts hidden, runs the script and quits. All of this is done without consuming a license. This is particularly useful to allow batch mode builds.



Chapter 10

Add-ins

This section provides an overview of the following add-ins:

- Web Publisher
- Model Integrator
- Rose C++ Analyzer

Web Publisher

Description

Web Publisher enables you to create a web-based (HTML) representation of a Rose RealTime model, that others can view using a standard browser such as Netscape Navigator or Microsoft's Internet Explorer.

Unlike sequential formats (such as paper or text files), Web Publisher lets you non-sequentially browse, search, and navigate your design. You can publish successive iterations of an evolving model for review or for sharing information. Another potential use is to publish documentation for a frozen API or framework.

Web Publisher recreates model elements, including diagrams, classes, packages, relationships, attributes, and operations. After published, hypertext links enable you to traverse the model much as you would in Rose RealTime.

You can control what Web Publisher includes by setting a variety of options. For example, you can select which packages of a model are published, the amount of detail to include, the notation to use, and the graphics format for diagrams. The View feature lets you launch your default browser and view the published model directly from Web Publisher.

Suggested workflow

To generate the files needed to create a web-based version of a Rose RealTime model:

1. Open the model you want to publish.
2. Choose **Tools > Web Publisher**.
3. From the Web Publisher dialog, select the publishing options you need.

Note that the dialog displays the options that were selected the last time a model was published.

4. Click **Publish** when you are ready to publish the model.
5. Click **View** to open your default web browser and view the published model. Remember that in the future you can open the published model in the browser by opening the *root file name* you specified on the Web Publisher dialog.
6. Click **Close** to close the dialog.

Limitations

The following browsers are supported:

- Microsoft's Internet Explorer 4.0 or later. (www.microsoft.com)
- Netscape's Communicator 4.06 or later. (<http://www.netscape.com/download>) If you want to publish the images in PNG format you need to add PNG support to Netscape Communicator. PNG Live (<http://codelab.siegelgale.com/solutions/pnglive2.html>) is a plug-in that provides PNG support for Netscape Communicator. Netscape Communicator 4.5 or better has built-in support for PNG and therefore does not require any special plug-in to view web pages created by Web Publisher. (www.netscape.com/download)

- Only eight colors are directly supported in published diagrams. Other colors are obtained by dithering. If you want to avoid dithering, set up Rose RealTime to use line and fill colors that are among the eight available.

The following table includes the eight available colors and their RGB values.

Colour	Red	Green	Blue
Red	255	0	0
Green	0	255	0
Blue	0	0	255
White	255	255	255
Black	0	0	0
Yellow	255	255	0
Magenta	255	0	255
Light Blue	0	255	255

- In published diagrams, you can normally click on a model element to go to that model element's specification information. This does not work for some model elements. These include aggregation relationships on the class diagram, transitions on the state diagram, association roles on the collaboration diagram, and connections on the deployment diagram.

For more information, see the Web Publisher online Help.

Model Integrator

Description

The Rose RealTime Model Integrator add-in allows you to compare up to seven units/models—called contributors—to a common root model/units—called the base contributor.

The add-in exists as a separate executable that can be launched stand-alone or from the toolset using **Tools > Model Integrator**. It is launched by the toolset when using the **Source Control > Show Differences**.

Model Integrator is capable of acting as a ClearCase Type Manager, meaning that ClearCase uses Model Integrator for showing differences and merging Rose RealTime units/models.

Suggested workflow

Merging two branches of a model

Assume a base model B and two models C1 and C2 and B is their common historical ancestor.

From Rose RealTime, select **Tools > Model Integrator** to launch Model Integrator.

To merge two branches from Model Integrator:

1. Select **File > Contributors** to open the Contributors dialog box.
2. Enter the base contributor B, then the two other contributors C1 and C2.
3. Click **Merge**.

For each contributor, Model Integrator loads the first level of subunits and brings up the **Subunits** dialog box.

4. Click **OK** to load all subunits.
Model Integrator shows the merged model potential conflicts.
5. Resolve each conflict by selecting the contributor to use for that conflict.

Note: To see model differences, select **Options > Compare Mode**.

6. When all conflicts are resolved, select **File > Save As** and choose a file name.
7. The subunits dialog box appears. Click **OK**.

Comparing local unit with the one in source control database

From Rose RealTime, select the unit to compare in the browser. Open the context menu and select **Source Control > Show Differences**.

For more information, see the Model Integrator online Help.

Rose C++ Analyzer

Description

The Rose C++ Analyzer is an executable bundled with Rational Rose 2000's Rose C++ add-in. Used in conjunction with the **Tools > Import** menu command, it provides a way to import legacy C++ systems into Rose RealTime.

Rose RealTime only supports the initial reverse engineering since the code is embedded within its model. Full target observability from the toolset is supported, thus eliminating the need to update code outside the toolset environment.

Note: *The online Help for the Rose C++ Analyzer contains Rose specific information that may not be applicable to Rose RealTime. We recommend that you limit your use of the add-in to the Suggested Workflow described below.*

Suggested workflow

From Rose RealTime, select **Tools > C++ Analyzer** to launch Analyzer.

From Rose C++ Analyzer

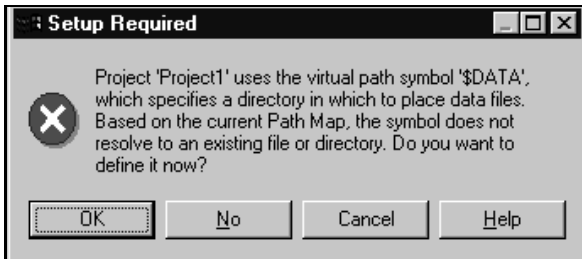
1. Create Project.
2. Set compiler settings.
3. Add Files.
4. Analyze.
5. Code Cycle.
6. Export to Rose.

From Rose RealTime

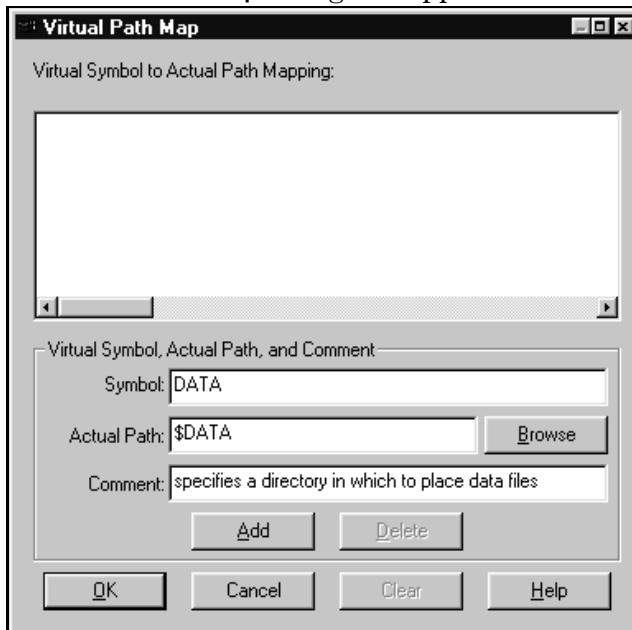
1. Select **File > Open** to load the Rose Model.
2. Select **Tools > Import Code** to import code from source files.

Notes

- When you create a Rose C++ Analyzer project for the first time, the following message prompts you to define the \$DATA/Rose pathmap symbol:



1. Click **OK**. The **Virtual Path Map** dialog box appears.



2. In the Actual Path field, enter an existing path where the Rose C++ Analyzer will store information about analyzed source files.
3. Click **Add** and then **OK**.
 - Windows NT users: You may not get this dialog if Rational Rose is already installed on your machine. In this case, the **Import Code** window appears.

- UNIX users: The default pathmap symbol \$DATA/ must be replaced with \$DATA.

Limitations

- C++ capabilities are limited by Rose RealTime's code generator's own limitation, for example, C++ templates, namespaces
- Round-trip engineering is not supported (and not required).
- Pathmap functionality is not supported (and not required).

For more information, consult the Rose C++ Analyzer online Help.



Chapter 11

Uninstalling Rational Rose RealTime

Windows

To uninstall Rose RealTime from a Windows machine:

1. Click **Start > Settings > Control Panel**.
2. Double-click **Add/Remove Programs**.
3. Select Rational Rose RealTime and click **Change/Remove**.

Follow the instructions on your screen to remove Rose RealTime.

Note: *We recommend that you also remove the Rose RealTime directories and registry settings from your system after uninstalling Rational Rose RealTime.*

UNIX

To uninstall Rose RealTime from a UNIX machine:

1. Save any user data files in another location before removing the installation directory.
2. Remove the installation directory and all of its contents.

