

Model Integrator Guide

RATIONAL ROSE® REALTIME

VERSION: 2003.06.00

PART NUMBER: 800-026120-000

WINDOWS/UNIX

Legal Notices

©2002-2003, Rational Software Corporation. All rights reserved.

Part Number: 800-026120-000

Version Number: 2003.06.00

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

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

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

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

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

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

Third Party Notices, Code, Licenses, and Acknowledgements

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

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

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

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

Licensee shall not incorporate any 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.

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

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

Contents

Preface	xi
Audience	xi
Other Resources	xi
Rational Rose RealTime Integrations With Other Rational Products	xii
Contacting Rational Customer Support	xiii
1 Introduction	15
Overview of Model Integrator	15
Memory Requirements and Performance	16
Starting Model Integrator	17
Using the Model Integrator Graphical User Interface	18
A Quick Start to Comparing or Merging Models	21
Using Model Integrator from the Command-Line	22
Using Model Integrator and Rational ClearCase	24
2 User Interface Overview	25
File Menu	25
Edit Menu	26
View Menu	26
Options Menu	28
Merge Menu	29
Help Menu	30
Contributors Dialog Box	30
Subunits Dialog	31
Diff Merge Dialog Box	34
Merge Errors Dialog Box	34
Filter Properties Dialog Box	35

Virtual Path Maps (Overview)	36
How Do Virtual Paths Work?	36
Where Are Virtual Paths Defined?	36
The & and * Symbols	36
Defining Virtual Paths	37
Edit Path Map (File Menu)	39
3 Examining the Composition of Model Files	41
Overview	41
Composition of Model Files	41
Basic Objects	42
Diagram Objects	42
View Objects	42
Mechanism	42
Quids	42
References	42
Unnamed Objects	43
Add-in Properties	43
Rational Rose RealTime Model File Versions	43
Understanding Subunits and Controlled Units	44
4 Considerations for Comparing and Merging	45
Merging Changes from Multiple Streams	45
Example: Adding Dependency Issues	47
Example: Changing Language Semantics	48
Understanding When Merging is Necessary	50
Merging Detailed Code Before Using Model Integrator	50
Merging When Using Unique Ids	52
5 Selecting Contributors	57
Contributors	57
Specifying Files in the Contributors Dialog Box	58
Base Model	59
Node	59
6 Choosing a Mode	61
Compare Mode	61
Merge Mode	61

Subtree Mode	62
View Mode	62
7 Loading and Saving Subunits	65
Subunits and Controlled Units	65
Subunit Status	66
Loading Subunits	66
Understanding Subunit File and Path Names	67
Setting a New Context for Subunits	67
Resolving Subunit Loading Errors	68
Pathmaps	68
How Do Virtual Pathmaps Work?	69
When Do You Need a Pathmap?	69
Saving Subunits	70
8 Comparing and Merging Models	71
Comparing Models	71
Merging Models	72
Automatic Merge	72
Merging Models Without a Base Model	73
Understanding Differences and Conflicts	73
Interpreting Compare and Merge Results	74
Starting a Merge	75
Semantic Checking	76
Using Semantic Checking On-the-Fly	77
Limitations of Semantic Checking	78
Resolving Merge Errors	78
Merging Models with Controlled Subunits	80
Merging Options	81
Accepting Changes from Contributors	81
Deciding Which Contributor to Use	82
Changing Nodes with Differences	82
Reversing Changes to Nodes	83
Performing a Partial Merge	83
Differencing and Merging Model Elements	84

9	Navigating through Models	85
	Filtering	85
	Searching for a Model Element	86
	Finding Nodes that Have Moved	86
	Viewing Model Elements that Have Moved	86
	Finding Referenced Nodes	86
	Viewing Nodes Referenced by a Node	87
	Viewing Conflicts and Differences	87
	Viewing Conflicts and Differences with Auto Advance	88
	Viewing the Parent of a Node	88
10	Using the Rational ClearCase Diff Merge Tool	89
	Overview	89
	Merge Source Code Example	90
	Specifying Contributors	93
	Starting the ClearCase Diff Merge Tool - Merge Source Code	96
	Recommendations	100
	Index	101

Figures

Figure 1	Model Integrator Graphical User Interface for Compare Mode	18
Figure 2	Model Integrator Graphical User Interface for Merge Mode	19
Figure 3	Diff Merge Dialog Box	34
Figure 4	Merging Changes Prior to Check-In	45
Figure 5	Comparison Between Versions	46
Figure 6	Removing Required Dependencies	47
Figure 7	Code Example Showing Changes to Language Semantics	48
Figure 8	Resulting File After Merging Changes	49
Figure 9	Figure 9 Incorrect Merge Scenario	54
Figure 10	Correct Merge Scenario	55
Figure 11	Merging Changes - Parallel Development	90
Figure 12	Model Integrator Window for Merge Mode	95
Figure 13	Diff Merge Notification	97
Figure 14	Model Integrator Merge Mode Window	99

Preface

This manual describes how Rational Rose RealTime Model Integrator allows you to compare and merge Rational Rose RealTime models. You can compare model elements, discover their differences, and merge them into a recipient model.

This manual is organized as follows:

- *Introduction* on page 15
- *User Interface Overview* on page 25
- *Examining the Composition of Model Files* on page 41
- *Considerations for Comparing and Merging* on page 45
- *Selecting Contributors* on page 57
- *Choosing a Mode* on page 61
- *Loading and Saving Subunits* on page 65
- *Comparing and Merging Models* on page 71
- *Navigating through Models* on page 85
- *Using the Rational ClearCase Diff Merge Tool* on page 89

Audience

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

This guide is specifically designed for software development professionals familiar with the target environment they intend to port to.

Other Resources

- Online Help is available for Rational Rose RealTime.

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

All manuals are available online, either in HTML or PDF format. To access the online manuals, click **Rational Rose RealTime Documentation** from the **Start** menu.

- To send feedback about documentation for Rational products, please send e-mail to techpubs@rational.com.

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

Rational Rose RealTime Integrations With Other Rational Products

Integration	Description	Where it is Documented
Rose RealTime–ClearCase	You can archive Rose RT components in ClearCase.	<ul style="list-style-type: none"> ▪ <i>Toolset Guide: Rational Rose RealTime</i> ▪ <i>Guide to Team Development: Rational Rose RealTime</i>
Rose RealTime–UCM	Rose RealTime developers can create baselines of Rose RT projects in UCM and create Rose RealTime projects from baselines.	<ul style="list-style-type: none"> ▪ <i>Toolset Guide: Rational Rose RealTime</i> ▪ <i>Guide to Team Development: Rational Rose RealTime</i>
Rose RealTime–Purify	When linking or running a Rose RealTime model with Purify installed on the system, developers can invoke the Purify executable using the Build > Run with Purify command. While the model executes and when it completes, the integration displays a report in a Purify Tab in RoseRealTime.	<ul style="list-style-type: none"> ▪ Rational Rose RealTime Help ▪ <i>Toolset Guide: Rational Rose RealTime</i> ▪ Installation Guide: Rational Rose RealTime
Rose RealTime–RequisitePro	You can associate RequisitePro requirements and documents with Rose RealTime elements.	<ul style="list-style-type: none"> ▪ <i>Addins, Tools, and Wizards Reference: Rational Rose RealTime</i> ▪ <i>Using RequisitePro</i> ▪ Installation Guide: Rational Rose RealTime
Rose RealTime–SoDa	You can create reports that extract information from a Rose RealTime model.	<ul style="list-style-type: none"> ▪ Installation Guide: Rational Rose RealTime ▪ <i>Rational SoDA User's Guide</i> ▪ SoDA Help

Contacting Rational Customer Support

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

Your Location	Telephone	Facsimile	E-mail
North America	(800) 433-5444 (toll free) (408) 863-4000 Cupertino, CA	(781) 676-2460 Lexington, MA	support@rational.com
Europe, Middle East, Africa	+31 (0) 20-4546-200 Netherlands	+31 (0) 20-4546-201 Netherlands	support@europe.rational.com
Asia Pacific	+61-2-9419-0111 Australia	+61-2-9419-0123 Australia	support@apac.rational.com

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

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

When sending email concerning a previously-reported problem, please include in the subject field: "[SR#XXXXX]", where XXXXX is the Service Request number of the issue. For example, "[SR#0176528] - New data on rational rose realtime install issue ".

Contents

This chapter is organized as follows:

- *Overview of Model Integrator* on page 15
- *Starting Model Integrator* on page 17
- *Using the Model Integrator Graphical User Interface* on page 18
- *A Quick Start to Comparing or Merging Models* on page 21
- *Using Model Integrator from the Command-Line* on page 22
- *Using Model Integrator and Rational ClearCase* on page 24

Overview of Model Integrator

Rational Rose RealTime Model Integrator lets you compare and merge Rational Rose RealTime models. You can compare model elements, discover their differences, and merge them into a recipient model.

Model Integrator is a ClearCase-type manager, and it will automatically attempt to resolve differences between models.

Typically, you will merge different versions of the same source (models, packages, classes); however, we recommend that you divide models into controlled units, then merge. For additional information on controlled units, see *Subunits and Controlled Units* on page 65.

Contributors are the models that form the input to Model Integrator. You can compare a maximum of seven contributor files.

For example, two developers can modify a shared model at the same time. They can either:

- Make a copy of the model, modify it separately, and then use Model Integrator to merge their changes into a single, shared copy of the model, or preferably, controlled units.
- Use Model Integrator to compare their models and identify the differences between them.

Model Integrator also lets you view the contents of a single model file; it provides a different way of looking at the model than is provided by Rational Rose RealTime. Model Integrator provides a low-level textual view of all the model elements and their properties. This means that you can examine a model quickly to view all of the property settings currently in use.

Note: Model Integrator is not a visual model or UML semantic-level merge tool, therefore, it lacks certain features that can make the merging of models more efficient and more accurate. For every Model Integrator use-case that fails to do what you may expect, there are many other use-cases that do add value or do what is expected, and will save you time. When using Model Integrator, you must understand what it can do efficiently and properly. For additional information about merging, see *Considerations for Comparing and Merging* on page 45.

Memory Requirements and Performance

For a typical merge operation, Model Integrator must load models (a minimum of two models when comparing without a base model, and a minimum of three models with a base model), and then compile additional information from the loaded models to compare and merge them. This requires an amount of memory proportionate to the number and the size of the contributors. The exact proportion varies, however, you can estimate the maximum amount of memory required.

To determine the amount of memory Model Integrator requires to complete a merge operation, calculate the sum of the sizes of all model files being merged and multiply that number by five (5x). This number represents the amount of memory that you must have to complete a merge operation, in addition to that used by your operating system and other programs.

Loading large models, for example 30 MB (megabyte) models, can put a strain on your system. A serious memory deficiency may be evident when loading models takes a very long time. Model Integrator may appear to be frozen while the disk drive is busy. This condition is known as thrashing. Thrashing occurs because Model Integrator constantly requires access to the entire data set for all models being merged but because of the physical memory shortage, much of this data is stored in virtual memory on your hard disk (in your computer's *pagefile* or *swap file* depending on which operating system you are using). The computer is busy reading and writing the disk, and very little real work is done. If your virtual memory configuration is also insufficient, you may need to reboot your computer to recover.

To improve Model Integrator performance, consider the following:

- Configure your computer with enough RAM to meet or exceed the 5x requirement stated above. If you have 30 MB of models to merge, you should have at least 150 MB of RAM in your computer. Anything less will compromise performance because Model Integrator will have to store its data to disk.
- Use fine-grained controlled units (see *Subunits and Controlled Units* on page 65).
- If there is not enough physical memory to meet the requirements of Model Integrator, ensure that you allocate enough virtual memory. Consult your operating system documentation .
- Close other programs to free-up memory. If you have sufficient RAM and virtual memory in your computer, other programs may use large portions of it. In some extreme cases, applications may load system components that are not unloaded even when the application exits. If you continue to encounter memory problems, try running Model Integrator immediately after rebooting your computer and prior to starting other applications.
- Use your operating system's tools to measure and report on memory usage.

Starting Model Integrator

You can start Model Integrator by launching the application as follows:

- From within Rational Rose RealTime, click **Tools > Model Integrator**.
- From within Rational ClearCase as part of a ClearCase compare or merge operation.

For information on using Model Integrator with ClearCase, see *Using Model Integrator and Rational ClearCase* on page 24.

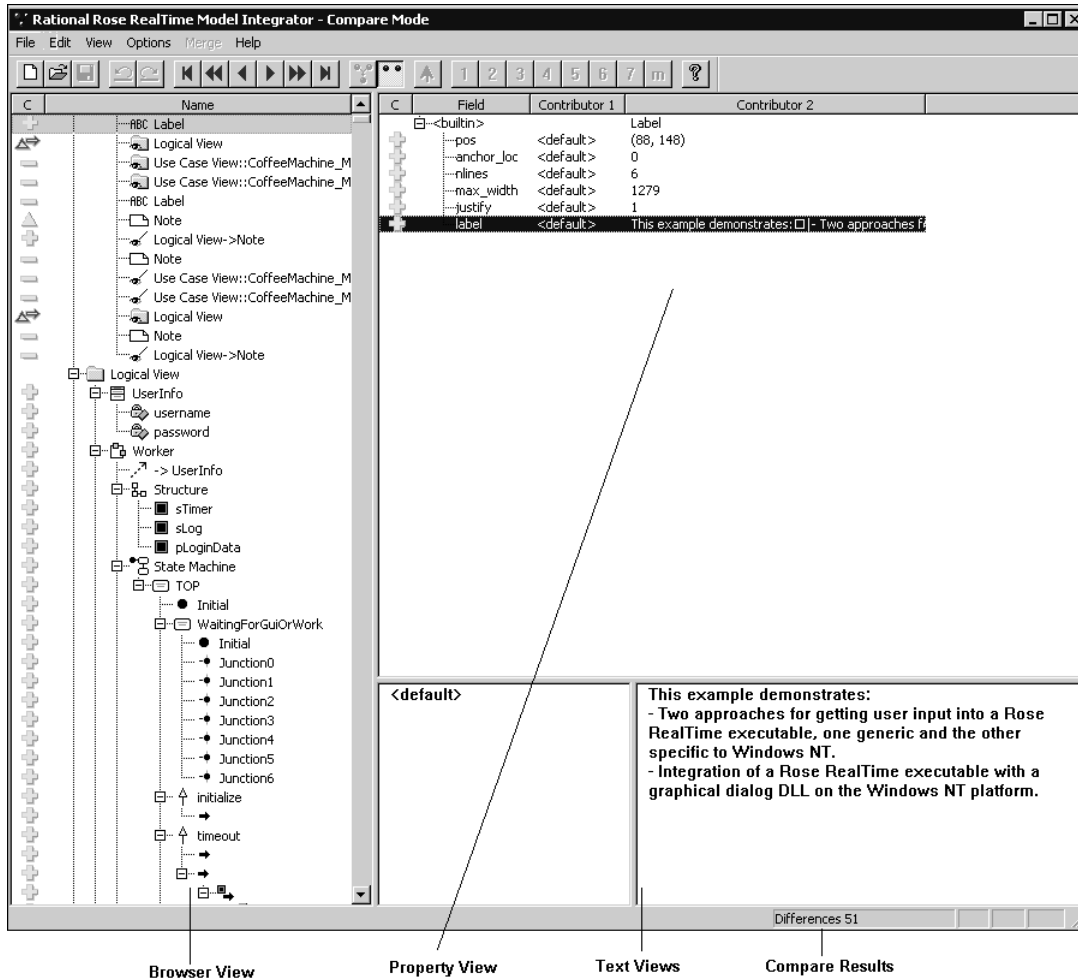
- From a UNIX shell process or a Windows console process, use the command-line.

For information on command-line options for Model Integrator, see *Using Model Integrator from the Command-Line* on page 22.

Using the Model Integrator Graphical User Interface

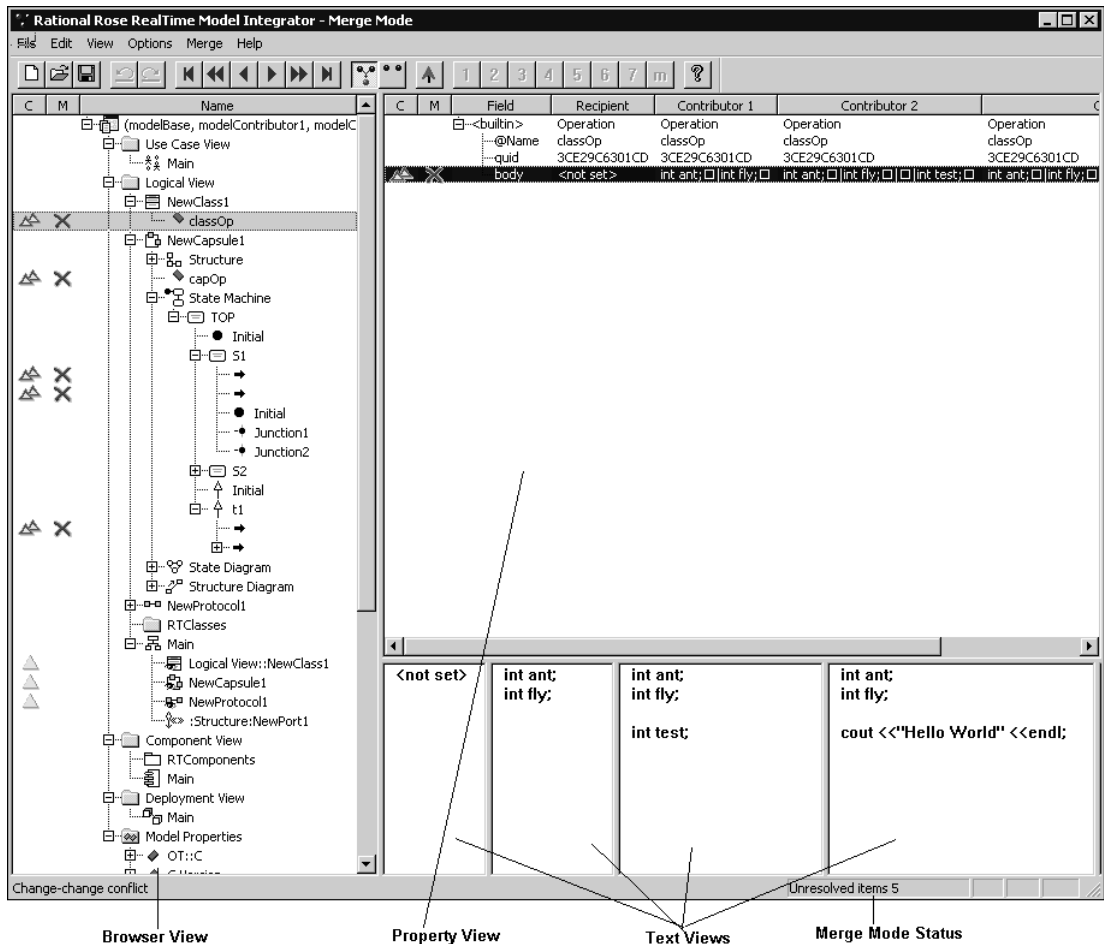
The two Model Integrator windows Compare Mode and Merge Mode. Figure 1 shows the Compare mode window, and Figure 2 shows the Merge mode window.

Figure 1 Model Integrator Graphical User Interface for Compare Mode



Note: In Compare mode, the Merge icons are not displayed, you cannot make any changes to the model, and the Merge menu and toolbar buttons are disabled.

Figure 2 Model Integrator Graphical User Interface for Merge Mode



The following components comprise the GUI for Model Integrator:

- **Browser View**

The left pane contains the primary objects that comprise the model. The objects displayed in the **Browser View** are not identical to the hierarchical tree structure on the **Model View** tab in Rational Rose RealTime. Model Integrator displays

some objects that Rational Rose RealTime hides from your **Model View**. For additional information on the objects that Model Integrator displays, see *Composition of Model Files* on page 41.

Although there are several models loaded into Model Integrator's memory, the **Browser View** displays only a single view of the model hierarchy. The **Browser View** shows all of the objects from all of the contributing models and it attempts to partner objects that are the same across all the models. If all of the Contributors have the same model element in the same location in the hierarchy, the browser will only display a single entry for that node of the model. If different Contributors have the same model element in different locations in the model, there will be a node in the **Browser View** for each location where the model element exists in the merged model. Only one of these locations is written to the final merged output model file which you determine when you resolve the conflict for that node.

The left side of the window has icons that display the results of the comparing and merging of the models. For detailed information on these icons, see *Interpreting Compare and Merge Results* on page 74.

- **Property View.** The upper right pane displays the set of properties that belong to the currently selected object in the **Browser View**. In this view, there is a column for each contributor and recipient model (in merge mode). There is also a column of icons to help you see the comparison state of the properties provided by the different contributors. These icons are the same as the comparison icons described earlier.
- **Text Views.** These windows along the lower right side of the main window display the values from each contributor for the property currently selected in the **Property View**. In Merge mode, the left-most text view displays the value for the recipient model, with the other contributors following it to the right in numerical order. These windows are for viewing purposes only; you cannot change the values.
- **Other GUI Features.** The Toolbar along the top makes some commonly used functions available as buttons. All of these functions are also available from the menus. If you position the cursor over the icons in the **Browser View**, a message explains the compare or merge state. At the bottom of the screen, a status bar displays the merge status of the node currently selected in the **Browser View**.

A Quick Start to Comparing or Merging Models

Follow these steps for a quick start to comparing or merging your models.

To compare or merge models:

- 1 Start Model Integrator. For information on how to start Model Integrator, see *Starting Model Integrator* on page 17.

Note: If you started Model Integrator from Rational ClearCase, continue with Step 3.

- 2 Click **File > Contributors**.

- 3 Select a maximum of seven files that you want to compare or merge. For information on how to specify files, see *Differencing and Merging Model Elements* on page 84.

The **Compare/Merge Against Base Model** option allows you to specify whether to compare or merge:

- All contributor models against each other (requires a minimum of two models). The result is a recipient model where all changes between the contributor files are merged together.
- Two or more contributor files against a base model (the version of the model that existed before any changes were made). The result is a recipient file that merges the contributor files against the base model.

Note: You must click in the empty area in the **Base** box for the **Compare** button to appear on the **Contributors** dialog box.

- 4 Click the **Compare** or **Merge** button to display the main window.

For a description of Compare mode, see *Compare Mode* on page 61, and for Merge mode, see *Merge Mode* on page 61.

- 5 If the **Subunits** dialog box appears, load or unload the appropriate units. For information on how to load and save subunits, see *Loading Subunits* on page 66 and *Saving Subunits* on page 70.

Note: Subunits for each contributor are loaded separately, so you will see a separate **Subunits** dialog box for each contributor .rtmdl file containing subunits. You can specify whether to load or unload an item by changing the type in the **Status** column between **> LOAD <** and **unloaded** by clicking on the text with your left mouse button. By default, Model Integrator loads all non-shared subunits for a model.

After the Model Integrator main window appears, you are ready to begin working.

Note: The bottom right corner of the Model Integrator window indicates the number of differences it encountered during the compare process.

Using Model Integrator from the Command-Line

Model Integrator supports a simplified command-line interface that you can use from DOS and UNIX command lines.

Note: To use Model Integrator from the command-line, the Rational Rose RealTime bin directory must be in your path.

To Launch Model Integrator from the command line:

- For Windows, type `modelintRT`.
- For UNIX, type `RoseRT -modelintRT`.

Table 1 shows the different options available for launching Model Integrator from the command line.

Table 1 Launching Model Integrator from the Command-Line

Command	Description
Windows: ModelintRT <i>file.mdl</i> UNIX: RoseRT -modelintRT <i>file.mdl</i>	Starts Model Integrator with <i>file.mdl</i> in the View mode.
Windows: ModelintRT <i>file1.mdl file2.mdl</i> UNIX: RoseRT -modelintRT <i>file1.mdl file2.mdl</i>	Starts Model Integrator in Compare mode for the two files specified.
Windows: ModelintRT <i>file1.mdl file2.mdl filen.mdl</i> UNIX: RoseRT -modelintRT <i>file1.mdl file2.mdl filen.mdl</i>	Starts Model Integrator in Merge mode with the first file named on the command-line selected as the base contributor. You can have a maximum number of seven contributor files.

Additionally, you can use the command-line options in Table 2. You can include the command line options immediately before or after specifying the files.

Note: Prefix each command with a forward slash (/) for Windows, or a minus sign (-) character for UNIX.

Table 2 Command-Line Options for Model Integrator

Command	Description
/xcompare	Starts Model Integrator in Compare mode for the files specified on the command line. Since this option is the default mode for two files, you do not need to specify this option but it must be specified when comparing three or more files.
/xmerge	Starts Model Integrator in Merge mode for the files specified on the command-line. This is the default mode for three or more files.
/compare	Starts Model Integrator in Compare mode but does not display the results in graphical mode. Graphical mode performs the compare operation and then exits to the operating system with an exit code indicating the result of the compare operation: 0 for identical models or 1 for models with differences.
/merge	Starts Model Integrator in Merge mode but does not display the results in graphical mode. If the merge algorithm detects conflicts, the merge is aborted and the program returns an exit code of 1. If the merge can be completed without conflicts, the merged file is saved using the file named by the /out command. If you do not specify an /out command, the Save dialog box appears. The Subunits dialog box will also appear unless a subunit policy choice is made.
/out filename	Specifies the name of the file to write to the merged output file. You must specify an absolute or relative path name for the file. The following examples are valid commands: <code>/out c:\models\test.rtm dl</code> <code>/out .\test.rtm dl</code> This command is <i>not</i> valid: <code>/out test.rtm dl</code>
/ask /all /none	Subunit policy options. The /ask command is the default for the graphical mode of Model Integrator. By default, when reading and writing models, Model Integrator displays a Subunit dialog box that allows you to specify whether to load or save subunits. The /all command loads or saves all the subunits without prompting you with the Subunit dialog box. The /none option suppresses the loading and saving of subunits.

Using Model Integrator and Rational ClearCase

Model Integrator is designed to work with Rational ClearCase to allow you to compare and merge individual model files from within the ClearCase environment. You can use the standard ClearCase tools such as the Version Tree Browser or the ClearCase context menus in Windows Explorer to compare model file versions and merge branched versions of models.

For example, you can do one of the following:

- Right-click on a model file version displayed in the ClearCase Version Tree Browser to display a context menu. Then, select **Compare > with Previous Version**. ClearCase invokes Model Integrator to show the differences.
- From the Windows Explorer, right-click on a model file in a ClearCase view and select **ClearCase > Compare with Previous Version**.

Note: Model Integrator is a ClearCase type manager. It will automatically resolve differences, where possible. If there are no conflicts to resolve, Model Integrator will not display the GUI. To configure Model Integrator with ClearCase, see the *Rational Rose RealTime Guide to Team Development* and the Rational ClearCase documentation.

Contents

This chapter is organized as follows:

- *File Menu* on page 25
- *Edit Menu* on page 26
- *View Menu* on page 26
- *Options Menu* on page 28
- *Merge Menu* on page 29
- *Help Menu* on page 30
- *Contributors Dialog Box* on page 30
- *Subunits Dialog* on page 31
- *Diff Merge Dialog Box* on page 34
- *Merge Errors Dialog Box* on page 34
- *Filter Properties Dialog Box* on page 35
- *Virtual Path Maps (Overview)* on page 36
- *Defining Virtual Paths* on page 37
- *Edit Path Map (File Menu)* on page 39

File Menu

New Session

Clears previous results in preparation for a new compare/merge session.

Contributors

Specify the input (contributor) files to compare or merge.

Save

Saves the merged model.

Save As

Saves the merged model with the option to specify a new file name for the model or to change the subunit configuration of the model.

Edit PathMap

Open the Virtual PathMap Editor Dialog Box.

Exit

Exits Model Integrator; prompting you to save the merged model if it has not been saved.

Edit Menu

Undo

Restores the previous value for the most recent change made to the merged model. All of your undo choices are remembered for the entire session.

Redo

Reverses the action of the Undo command.

Select All

Selects all items in the current window. Selected items are highlighted.

Search

Highlights all instances of a search word or phrase in the Model Integrator windows.

View Menu

Toolbar

Displays or hides the toolbar.

Status Bar

Displays or hides the status bar.

Expand

Expands the selected node of the browser view to display the model hierarchy.

Elide

Collapses the selected node of the browser view.

Expand All

Expands all nodes of the browser view to display the entire model hierarchy.

First Conflict

Goes to the first conflict, difference, or item in the browser view, depending on the setting of **Options > Auto Advance**.

Previous Conflict

Goes to the previous conflict (if any) in the browser view.

Previous Difference

Goes to the previous difference or item in the browser view, depending on the setting of **Options > Auto Advance**.

Next Difference

Goes to the next difference or item in the browser view, depending on the setting of **Options > Auto Advance**.

Next Conflict

Goes to the next conflict (if any) in the browser view.

Last Conflict

Goes to the last conflict, difference, or item in the browser view, depending on the setting of **Options > Auto Advance**.

Parent Node

Changes the current selection in the browser view to be the parent of the current node.

Other Location

Changes the current selection in the browser view to show you another location where this same node exists in the model. Applies to nodes which have been moved by one or more contributors.

Referenced Node

Changes the current selection in the browser view to show you the node referenced as the "client", "quidu", or "supplier" by the current node.

Previous Location

Changes the current selection in the browser view to return to the node you were at before the last View > Parent, View > Other Location, or View > Referenced Node command.

Options Menu

Merge Mode

Changes Model Integrator to use **Merge** mode. Activates the Merge menu. Before exiting the program, you will be prompted to save the merged model.

Compare Mode

Changes Model Integrator to use **Compare** mode. In this mode, models are compared but no merge results are saved.

Show Differences Only

Filters the nodes displayed in the browser view. In **Merge** mode, it filters the browser view to display only nodes with conflicts. In **Compare** mode, it filters the browser view to display only nodes with differences. In either case, parents of the selected nodes are also displayed.

Hide Deleted Nodes

Hides nodes that are deleted, which can make it easier to see what the model will look like if there are many deletions. The command does not hide nodes deleted by a user merge choice.

Auto Advance

After accepting a change, the Auto Advance function moves the current selection in the browser view to the next conflict or difference (depends on the Auto Advance option selected). Auto Advance > None turns Auto Advance off. The Auto Advance selection also affects the operation of the View > Next/Prev/First/Last commands.

Filtering

Opens the **Filtering Properties** dialog box, where filters can be set that allow small changes in the position of objects in a Rational Rose RealTime diagram to exist without creating a conflict when merging models.

Merge Menu

Resolve All Conflicts Using

Resolves all unresolved conflicts by accepting changes from the selected contributor. Conflicts which have been previously resolved will not be affected by this command.

Resolve Selected Nodes Using

Resolves unresolved conflicts of the selected node by accepting changes from the selected contributor. Conflicts which have been previously resolved will not be affected by this command.

Revert Selection

Changes the selected nodes back to their unmerged state, erasing any previous merge decisions made either by you or by Model Integrator.

AutoMerge Selection

Applies Model Integrator's automatic merge algorithm to the selected nodes. This command will only operate on nodes that have been reverted (see Merge > Revert). Nodes with differences will select the contributor that causes the difference. Nodes with conflicts will remain unresolved.

Subtree Mode

Turns Subtree mode on and off. Subtree mode allows you to apply merge mode commands to both the current node and all of its children.

Semantic Checking

Toggle to enable or disable on-the-fly reference checking. When enabled, merge the choices which would cause a reference to a deleted node are not allowed.

Check Merge

Displays the Merge Errors dialog box.

Merge Source Code

Merges the source code of the contributor models.

Help Menu

Help Topics

Displays the online help for Model Integrator.

About Model Integrator

Displays the copyright information, version number and build identifier for Model Integrator.

Contributors Dialog Box

Files

Displays a list of files to compare or merge. You can enter file names or browse your computer to find files to add to the list. Files can be complete model files (.mdl) or subunit files, including .rtclass, .rtlogpkg, .rtcmppkg, .rtdeploy, .rtcollab, .rtdeploydgm, .rtclassdgm, .rtcmp, .rtprcsr or .rtdev.

Click the **Browse** button to open the **Add Contributor** dialog from which you can browse for the file that contains the model or controlled unit you want to add to the **Files** list.

New (Insert)

Adds a file to the **Files** list.

Note: When selecting a base model, it should be the common ancestor of the models you are comparing or merging. The common ancestor is the model from which the other contributors have evolved.

Delete

Removes a file from the **Files** list.

Move Up

Changes the order of the currently selected file by moving it up in the **Files** list

Move Down

Changes the order of the currently selected file by moving it down in the **Files** list.

View

Uses **Compare** mode to view the differences between the files in the contributors list.

Merge

Compares files and creates a recipient into which changes can be merged. If Model Integrator encounters a conflict, it will mark the conflict and allow you to decide how to resolve it.

Cancel

Closes the dialog without performing a compare or merge of the files.

Help

Displays the online help for the Contributors Dialog Box.

Compare/Merge Against Base Model

When checked, Model Integrator compares the other contributors to the base model (the first contributor) to determine differences and conflicts. This is the normal mode of operation when merging models that represent different branches derived from a common parent.

Subunits Dialog

Context

Displays the path of the subunit you are working with.

Load all nested subunits

Shows all of the subunits that belong to a file specified in the Contributors dialog box.

Unit

Shows all of the subunits that belong to a file specified in the Contributors dialog box.

Click the package name in the Units column and then click **Browse** to find the actual path of the controlled unit.

Status

Shows the status of each subunit that belongs to a file specified in the **Contributors** dialog.

The Status values are:

- **Loaded** - Indicates that the package is a controlled unit and that the unit is currently loaded.
- **Unloaded** - Indicates that the package is a controlled unit that is not currently loaded. To be considered during the compare or merge, the unit must be loaded.
- **Not a unit** - Indicates that the package is not a controlled unit and therefore does not need to be loaded or unloaded. It is always considered during the compare or merge.
- **LOAD** - Is the default value for controlled units that are not currently loaded. If you click on **LOAD**, its value toggles to **Unloaded**. Any unit whose status is **LOAD** will be loaded when you click **Apply** or **OK**.

Actual Path

Contains the actual path of each controlled unit for which a valid virtual path has been defined. If the actual path does not appear or is incorrect, it may be because no virtual path map entry exists for the unit.

Virtual Path

Contains the PathMap symbol, if any, defined for each controlled unit. If you need to define a new PathMap symbol, click **PathMap** to start the Rose RealTime PathMap Editor. When you return to the Subunits dialog, it will include a correct virtual and actual path for the unit.

OK

Loads any subunits whose status is **LOAD** and displays the next Subunits Dialog Box or completes the **Compare** or **Merge** operation.

Cancel

Continues without loading any subunits.

If another contributor file also has subunits, the **Subunits** dialog displays again, this time for the next contributor file.

When there are no more contributor files with subunits, Model Integrator completes the **Compare** or **Merge**, without loading subunits.

Apply

Loads any units whose status is set to **LOAD** without closing the dialog

PathMap

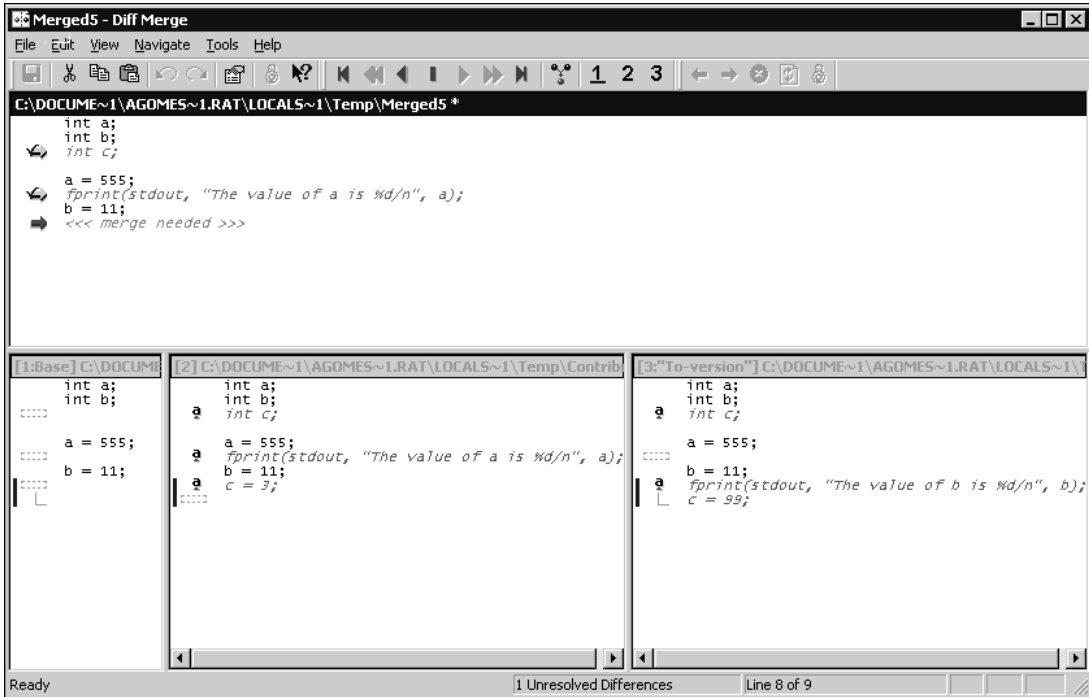
Starts the Rational Rose RealTime PathMap editor to add an appropriate PathMap symbol for the controlled unit. You can use the PathMap editor to add or change PathMap symbols as required. For detailed instructions, click **Help** in the **Virtual PathMap** dialog. When you return to the **Subunits** dialog, it will include a correct virtual and actual path for the unit.

Browse

Select a subunit to load; then click **Browse** to find its actual path.

Diff Merge Dialog Box

Figure 3 Diff Merge Dialog Box



Merge Errors Dialog Box

Errors List

Displays the list of errors which resulted from the last merge.

View Error

For the item currently selected in the error list, shows you the node of the model where a dangling reference is being defined. Depending on the error condition, either this node or one of its parents has been deleted or moved by the contributor currently selected for the recipient. To repair the problem, you must accept changes from a different contributor at either the error definition node (to define the missing node) or at the referencing node (to eliminate the reference).

View Parent

For the node currently selected in the browser view, this button shows you the parent of that node. It is the same as the View > Parent menu command. It is useful when the error condition states that the parent of a referenced node has been deleted. Use this button to view the parent nodes.

View Definition

View Definition takes you to the referenced element in the browser tree. This is useful if you don't know where the referenced item/element is located in the browser tree.

View Other Locations

For the item currently selected in the error list, shows you another place in the model where this same node is defined (because different contributors have defined the same object in different places). This command is useful when the error condition is a forward reference error. In this situation the contributor that is currently accepted into the recipient has moved the object to a place in the model where it is being used before it is defined. Use this command to view all the locations.

Refresh List

Updates the Merge Errors list.

Filter Properties Dialog Box

All Coordinates - When enabled, differences in position between diagram elements in the model files that are less than the filter value will not result in a conflict. Vertical and horizontal differences are calculated separately.

Max_width - When enabled, differences in the maximum width of Label tags in the model files that are less than the filter value will not be reported.

Filter Value - Sets the level in pixels, at which differences in the Property value will result in a conflict. Changes to the Property of pixel values less than the Filter Value will not create a conflict.

Virtual Path Maps (Overview)

The references to controlled units are stored as file paths in the model file or parent units. To allow models to be moved between different folder structures and to be updated from different workspaces, Rational Rose provides a mechanism called virtual path maps. Virtual path maps are used to reference controlled units using a virtual path instead of an absolute path.

How Do Virtual Paths Work?

When Rational Rose saves a model, it tries to substitute every absolute path with a virtual path. When Rational Rose opens a controlled unit, or uses a path specified in a model property, each virtual path is transformed back into an absolute path. (For more information about the substitution algorithm, see Path Map Algorithm.)

For example, if a user has defined a virtual path,

```
$MYPATH=Z:\ordersystem,
```

and saves a package as

```
Z:\ordersystem\user_services.cat,
```

the model file will refer to the package as

```
$MYPATH\user_services.cat.
```

When another user, who has defined \$MYPATH as

```
$MYPATH=X:\ordersystem,
```

opens the same model from his/her "X" drive, Rational Rose resolves the internal reference to the controlled unit and loads the following file:

```
X:\ordersystem\user_services.cat.
```

Where Are Virtual Paths Defined?

Select **File >Edit Path Map** to display the **Virtual Path Map** dialog box you use to create virtual paths. The dialog box contains a list of entries; each entry represents a mapping between a virtual path symbol and an absolute path.

The & and * Symbols

A leading "&" in a path map definition represents the folder where the enclosing controlled unit or model file is located, and can be used to reference units relative to model files and other units. For example, the virtual path symbol, \$CURDIR, in the illustration above is defined as the actual path &. If you save a package as

```
Z:\ordersystem\units\user_services.cat
```

and the model file as

```
Z:\ordersystem\ordersys.mdl,
```

the model file will refer to the unit as

```
$CURDIR\units\user_services.cat.
```

When another user, which has defined \$CURDIR in the same way, opens the same model, Rational Rose interprets the reference to the unit's file in the context of that user's workspace, for example

```
X:\ordersystem\units\user_services.cat.
```

Also, a wildcard, "*", in the path map can be used to parameterize a virtual path.

Defining Virtual Paths

The references to controlled units are stored as absolute file paths in the model. If you control a logical package in a ClearCase view, the model's reference to that controlled unit will include the view name (for example, X:\ordersystem\units\user_serv.cat). When another member in the team opens the model in another view (for example, Y:), Rational Rose cannot find the referenced unit. In order to allow different users to access the model from different workspaces, virtual path maps should be used.

The dialog box contains a list of entries; each entry represents a mapping between a virtual path symbol and an actual pathname.

To define a virtual path:

- 1 Open the model in Rational Rose. Make sure that the model file and all of its controlled units are write-enabled. That is, if they are under version control, you must check them out.
- 2 Click **File > Edit Path Map** to open the **Virtual Path Map** dialog.
- 3 Type the name of the virtual path in the **Symbol** box (for example, "MYPATH"), but omit the leading "\$" character.
- 4 In the **Actual Path** box, enter the folder where the model file is located. In the picture below, the path map symbol, \$MYPATH, points to the folder X:\ordersystem.
- 5 Click **Add**. You have now defined a virtual path map symbol, \$MYPATH.

- 6 To substitute the current physical paths to any existing controlled units in the model file, save the model and all of its controlled units.

Note: Note: Each user that is going to work on this model will have to define the same path map symbol before opening the model. For example, another user with the private workspace Y:\ordersystem, must define \$MYPATH=Y:\ordersystem.

When anyone in the team opens or saves a model hereafter, Rational Rose will try to match the longest possible file path to the symbols in the path map. For example, if a model references the controlled unit X:\ordersystem\units\data_serv.cat, the actual reference in the model file will be \$MYPATH\units\data_serv.cat. Thus, when another user opens the model in his/her private workspace, \$MYPATH will be substituted with the path defined by that user's path map.

To define a path map relative to the location of the model file:

A leading "&" on a path name indicates that the path is relative to the model file or the enclosing controlled unit (if any). For example, suppose you have created a model:

```
X:\ordersystem\ordersys.mdl
```

and a controlled unit:

```
X:\ordersystem\units\data_serv.cat.
```

To allow different users to open the model and load the unit in different workspaces, each user can create a path map:

```
$CURDIR=&.
```

When the model is saved, the reference from the model file to the package is stored as:

```
$CURDIR\units\data_serv.cat
```

When the model is opened in another workspace, \$CURDIR is expanded to the physical path to the model in that specific workspace, for example:

```
Z:\ordersystem.
```

Note: The "&" requires that the controlled units are located in the same folder as the model file or in a subfolder to the model file.

To define a new path map using another path map symbol:

The actual path in a path map definition can contain previously defined path map symbols. For example, if there is a path map, \$ROOT=X:\model_vob, you can define a path map for the path X:\model_vob\ordersys by simply adding the path map \$MYPATH=\$ROOT\ordersys.

To define a parameterized path map:

A wildcard character, "*", in the path map can be used to parameterize a virtual path. For example, if the following virtual path is defined:

```
$SUBSYSTEM=\\server\models\project*\fred
```

and each user working on "project" has his/her own set of model files within each subsystem, then a controlled unit belonging to the display subsystem may have the following path:

```
\\server\models\project\display\fred\diagrams.cat
```

the model file will refer to the unit as:

```
$SUBSYSTEM(display)/diagrams.cat
```

When the model is opened by user "susanne," who has the following virtual path definition:

```
$SUBSYSTEM=\\server\models\project*\susanne
```

the virtual path reference to the unit is transformed back to the actual path:

```
\\server\models\project\display\susanne\diagrams.cat
```

This allows different users to work on the same files, with the same contents, but in different folders, without having to define a virtual path symbol for each such folder.

To use virtual paths in the value of a model property:

Rational Rose does not convert actual paths in model properties to virtual paths. In order to use a virtual path in the value of a model property, you must manually enter the virtual path map symbol, including the "\$" sign - for example, \$CURDIR - into the value of the model property.

Edit Path Map (File Menu)

The **Edit Path Map** dialog lets you create an entry to represent a mapping between a virtual path symbol and an actual pathname.

This feature allows you to work with models moved or copied among workspaces and archives by redefining the actual directory associated with a user-defined symbol.

The dialog box contains a list of entries; each entry represents a mapping between a virtual path symbol and an actual pathname.

To display the **Edit Path Map** dialog box, click **File > Edit Path Map**.

Examining the Composition of Model Files

3

Contents

This chapter is organized as follows:

- *Overview* on page 41
- *Composition of Model Files* on page 41
- *Understanding Subunits and Controlled Units* on page 44

Overview

To understand how Model Integrator works, you must first understand the composition of the model files generated by Rational Rose RealTime. A Rational Rose RealTime model consists of a set of objects (also called model elements, items, or nodes). Each object has its own set of properties that define attributes of the object. Model Integrator exposes all of the objects and properties defined in the models that you are comparing or merging.

Composition of Model Files

A Rational Rose RealTime model can include the following object types:

- *Basic Objects* on page 42
- *Diagram Objects* on page 42
- *View Objects* on page 42
- *Mechanism* on page 42
- *Quids* on page 42
- *References* on page 42
- *Unnamed Objects* on page 43
- *Add-in Properties* on page 43
- *Rational Rose RealTime Model File Versions* on page 43

Basic Objects

Basic objects are the main objects in your model. They are the objects that represent things in your model, such as actors and classes.

Diagram Objects

All of the diagrams you create in a model are objects. Diagrams are displayed differently in Model Integrator than in Rational Rose RealTime. The diagram titles will be the same in the **Browser View** area, but the diagrams are not shown as graphics. They are displayed as text lists of their component objects. Some of these components you are already familiar with, such as labels. Others are new because Rational Rose RealTime does not display them for you; these objects include the View Objects.

View Objects

Every basic object is represented by a view object when it appears in a diagram. For example, a class appears on a diagram in your model. The diagram object will have a child of the **ClassView** object for that class, and so on for every type of basic object. Other view objects exist for items that are part of a Mechanism.

Mechanism

A mechanism is a hidden part of a model which contains a set of objects used internally to implement the parts of the model you created. A mechanism will contain more child objects.

Quids

A QUID is a unique identifying number that distinguishes the object it is attached to, regardless of the object's name. Rational Rose RealTime generates a QUID property for each object when as it is first created in the model. QUIDs are unique, because they identify an object when the name of the object changes, or when the object is moved in the model. Model Integrator uses QUIDs extensively to determine whether objects are the same; if the QUID is the same, then the objects have a common ancestry.

For information on setting the unique id option, see *To set the Generate unique identifiers for all elements option*: on page 53.

References

Much of the power of the Rational Rose RealTime model comes from the relationships that exist between objects. These relationships are identified by reference properties (or just references) based on QUIDs, that enable one object to point to another. A given object in a model may have no references at all, or it may have many. Reference

properties have names such as *client* and *supplier*. Model Integrator provides a **View > Referenced Nodes** command that lets you to view the referenced model elements.

Note: It is essential to maintain valid references between the objects in the model after a merge is completed. When objects are deleted or moved, Model Integrator must check to ensure that references from other objects continue to be valid. This semantic checking function is performed before the model is saved.

Unnamed Objects

Virtually every object in a Rational Rose RealTime model has its own unique name. However, you are not required to name every object that you create. For objects that you do not name, Rational Rose RealTime creates a name of the form:

\$UNNAMED\$*nn*

where *nn* is a number for that object.

Often, a model will contain many unnamed objects that you are not aware of because Rational Rose RealTime never shows you the string \$UNNAMED\$. Model Integrator displays the name of every object including unnamed objects where *nn* represents an object's name. Usually you can tell what an object is by looking at its icon in the **Browser View**, its properties (the object type will be at the top of the **Property View**), and in some cases, by looking at the children of the object.

Add-in Properties

Models can contain properties specific to, for example, language add-ins. During a compare or merge, these properties are also considered and compared.

Rational Rose RealTime Model File Versions

Each model is assigned a number that corresponds to the version of Rational Rose RealTime in which it was created. You can see the model file version information listed in the property view of the first node of the model in the **Browser View**, under the **@Petal** property.

Models must have the same version number to be successfully merged.

Understanding Subunits and Controlled Units

Rational Rose RealTime stores all or part of a model in one or more files. If a model is divided into separate files, the files other than the main .rtmdl file are called subunits. In a team environment, these files would be under version control. When under version control, these subunit files are called controlled units. Subunits and controlled units are created to define portions of the model upon which individual developers can work. Breaking up a model into controlled units allows a project team to develop a model in parallel.

Note: For optimum merge results, use controlled units (fine-grained controlled units are best).

If the files that you are attempting to compare or merge contain subunits or controlled units, Model Integrator prompts you to determine whether to load or unload these units.

Rational Rose RealTime supports the following controlled units as separate units (file extensions for corresponding files are shown in parentheses):

- Model (.rtmdl)
- Package (.rtlogpkg), (includes Use Case Packages and Logical Packages)
- Class Diagram (.rtclassdgm) (includes Use Case Diagram)
- Class (.rtclass)
- Capsule (.rtclass)
- Protocol (.rtclass)
- Use Case (.rtclass)
- Actor (.rtclass)
- Collaboration (.rtcollab)
- Component Package (.rtcmppkg)
- Component Diagram (.rtcmpdgm)
- Component (.rtcmp)
- Deployment Package (.rtdeploy)
- Deployment Diagram (.rtdeploydgm)
- Processor (.rtprcsr)
- Device (.rtdev)

For additional information on subunits, see *Loading and Saving Subunits* on page 65.

Considerations for Comparing and Merging

4

Contents

This chapter is organized as follows:

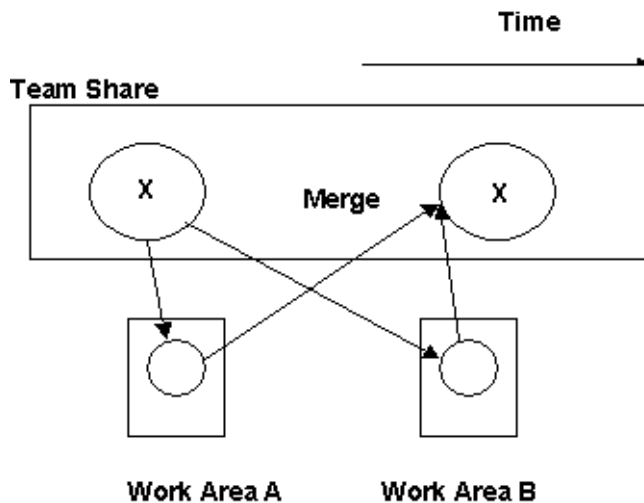
- *Merging Changes from Multiple Streams* on page 45
- *Understanding When Merging is Necessary* on page 50
- *Merging Detailed Code Before Using Model Integrator* on page 50
- *Merging When Using Unique Ids* on page 52

Merging Changes from Multiple Streams

You can use Model Integrator to apply a combined set of changes in situations where changes have been made to multiple versions of a single file in different streams.

Figure 4 shows how you can merge two changes made to the same file. A file **X** is modified in stream **A**. A version of the same file is also modified in stream **B**. Later, the changes are merged together to create **X'** with both results merged.

Figure 4 Merging Changes Prior to Check-In

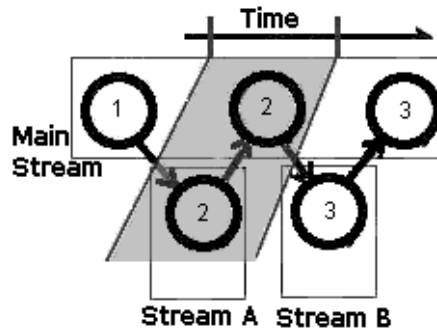


Considerations

- It may be difficult to remove a set of changes that occurred in a previous version of a file.
- When merging, a conflict may result with another change if it removes a dependency that one or more other files rely on. For additional information on this type of merging conflict, see *Example: Adding Dependency Issues* on page 47.
- The changes made may affect the language semantics of the file. For additional information, see *Example: Changing Language Semantics* on page 48.

The situation in Figure 5 shows us three versions of a file. If you want to remove all changes applied to the second version (the changes occurring between the two diagonal lines), you may encounter difficulties.

Figure 5 Comparison Between Versions



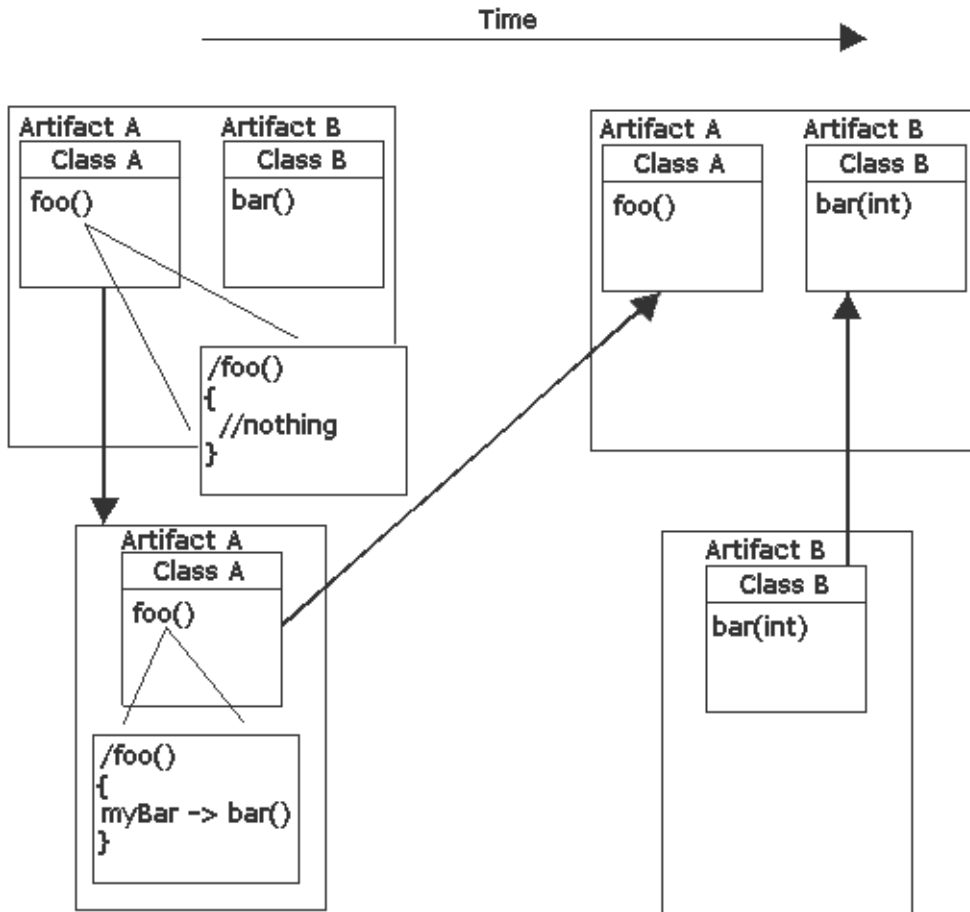
Considerations

- For example, the changes between version 1 and version 2 must be compared to the changes between version 1 and version 3.
- Obtaining adequate permission to modify files helps to ensure that unintentional changes do not occur. Configuration management can choose to implement and enforce this type of policy.
- Practice ownership of files. This means that changes can only be made by the owner of the file. This makes it easier to resolve merge conflicts since the owner is aware of the files and what changes should be made.

Example: Adding Dependency Issues

Modifying an file may cause a conflict with another change if it removes a dependency that one or more other files rely on. Figure 6 shows how this type of problem can occur.

Figure 6 Removing Required Dependencies



Files **A** and **B** are checked out to different streams. In stream **A**, a developer creates a new dependency in `foo()` by adding `myBar -> bar()`.

In stream **B**, a developer makes changes to `bar()` in class **A** by changing the parameter signature to integer.

Changes to bar - from bar() to bar(int) - cause any references to this function to fail. The changes made by the developer in stream B (to file B) that are referenced by foo in file A in stream A are not valid.

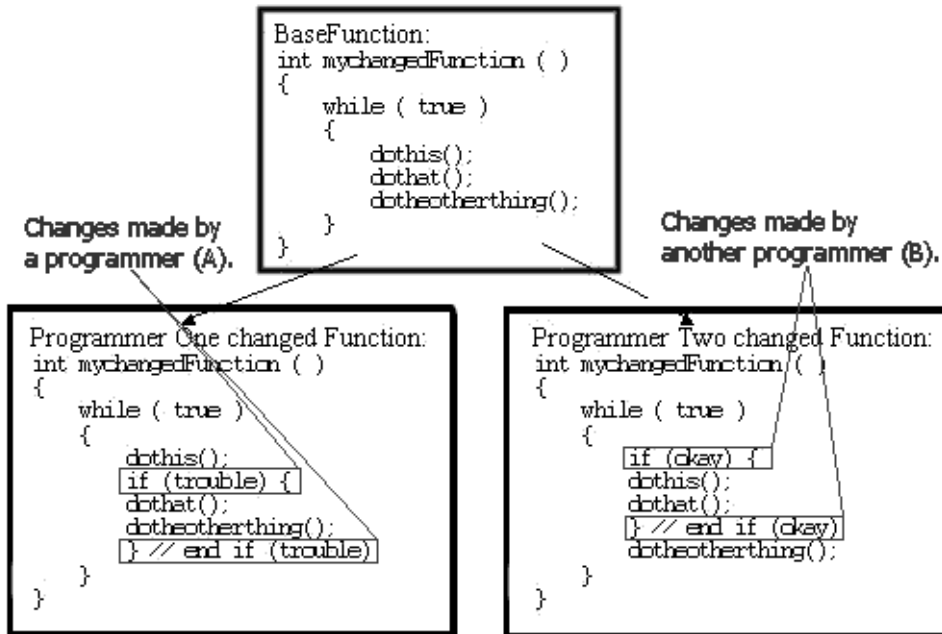
Note: Most merge tools are unable to identify a conflict here because they compare items of work individually and not against all referenced work.

Example: Changing Language Semantics

When product maintenance is underway and feature development is concurrently managed, the developer may be unsure of all dependencies involved in a proposed change. Rather than search all the dependencies associated with the file, they do not modify the original stream. Instead, they create a new item with the proposed changes.

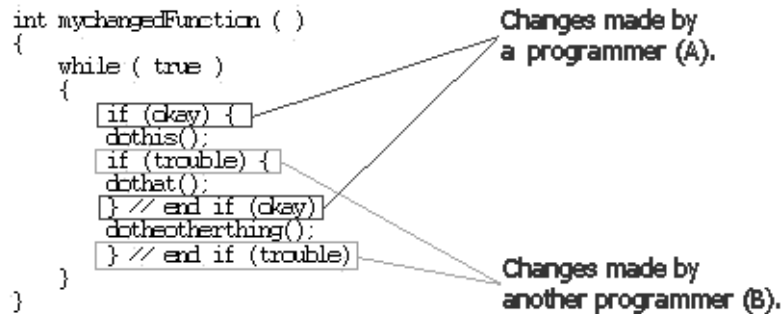
When modifying a file, team members must be aware that subsequent changes can affect the language semantics of the file. This type of change is common and may have serious implications. Figure 7 shows an example of how changes can produce unexpected results.

Figure 7 Code Example Showing Changes to Language Semantics



Unless your merge tool knows something of the language semantics used, it may produce a file like that in Figure 8.

Figure 8 Resulting File After Merging Changes



The function `dothat()` is called only when there is `trouble`, and the function `dotheotherthing()` is called only when `okay` is true. Since most developers do not comment ending braces, it is difficult to identify the problem created by merging. Figure 8 only illustrates a small example. A much more complex code block may present a situation where it is difficult to see the unintended change.

Considerations

Rational Rose RealTime Model Integrator is aware of the language semantics/model syntax of model files, but it is not aware of the language semantics for any language add-ins, or the UML.

Note: Although Rational Rose RealTime files are text files, standard text file merge tools are not aware of the Rational Rose RealTime language semantics or model syntax, and should not be used to merge model files because it will likely corrupt model files.

If you require merging, perform it outside of these integration streams, and sanity test it before integrating it as a new version.

Note: You can use merge tools, such as Model Integrator, for merging simple, non-conflicting changes. We recommend that:

- changes be made in a controlled manner
- you be aware of the expected results of a merge
- merges be reviewed for correctness after all merge operations, including non-conflicting merges.

Understanding When Merging is Necessary

Merging is necessary when changes are made to two or more versions of a file. Perform the merge as often as possible. Every developer involved in a concurrent change must regularly work with a merged version of the ongoing work to identify adverse or unintended changes.

The intention is to reduce the amount of effort required to resolve a merge conflict.

With the method used by Rational ClearCase to facilitate integration branches, it is wise to choose a special integration stream for verifying merges for a configuration item. This isolates the remaining files in your system from these changes until the configuration has been verified.

Note: After every merge, we recommend that you assess changes to semantic relationships and other dependencies.

Merging Detailed Code Before Using Model Integrator

In Model Integrator, some model elements include detailed code, textual documentation, and other elements, and you must select one contributor over another. When you use Model Integrator with Rational ClearCase, you can resolve these textual merges, see *Using the Rational ClearCase Diff Merge Tool* on page 89.

When comparing models, Model Integrator looks at the model elements which it then compares with other elements based on properties. For example, for a base model, called ModelX, there are two contributors, Contrib1 and Contrib2. If property A of element A from Contrib1 is equal to element A of property A for Contrib2, then all code associated with the transition is in a single property; the base model has element A.

When property A of element A from Contrib1 is **not** equal to element A of property A for Contrib2, Model Integrator detects the difference and allows you to select a contributor. Selecting a contributor causes the base model to change. The result is a merged model with the changes from a single contributor.

To merge before using Model Integrator:

- 1 Abandon the merge.
- 2 Export the code from Contrib1 to a file.
- 3 Export the code from Contrib2 to another file.

- 4 Use another merge tool, such as Rational ClearCase, to merge the source code from the two files.
- 5 Import the merged source code into Contrib1 in Rational Rose RealTime Model Integrator.
- 6 Use Model Integrator to merge Contrib1.

Primary edits that involve more than one controlled unit are the most troublesome, and is more common in projects where the recommendation of not practicing file ownership is not followed. When this situation arises, there are typically two approaches to resolving change:

- The user making the primary edits performs a private check-out of all affected controlled units. The affected controlled units are then later merged into another stream, possibly at integration time. Unfortunately, the type of merging that must be performed is less predictable and planned. It is difficult for any tool to properly and completely address the complexities of these merges in a reliable and robust manner.
- The user making the primary edits coordinates with the owners of the other affected controlled units to implement a change; ultimately, to avoid the necessity for a merge later. This approach is difficult to do and does not take advantage of the change management features of the tools in the tool chain. Important considerations for this approach to implementing a change are who will do the changes and when.

Considerations

When more than one user needs to make changes to the same file, they can make the changes at the same time. The changes are merged back into one file at a later date. The benefit of this approach is that work goes on in parallel, and it saves time. The problem is that arbitrary and uncoordinated changes on the copies of the same file can be difficult to resolve during the merge process. In fact, they may never be resolved, and the changes from only one contributor are accepted and from the other are discarded.

We recommend practicing ownership of files so that the impact of the changes to the multiple streams is understood and can be more easily resolved.

Note: We recommend that these types of changes be **coordinated and merged often**.

Invest time in understanding what Rational Rose RealTime Model Integrator will, and will not do during a merge.

Resolve all issues relating to merging parallel changes prior to integration.

Merging When Using Unique Ids

Unique ids are unique internal names associated with model elements. They are used internally by Rational Rose RealTime. Not all model elements require unique ids. Rational Rose RealTime includes a feature that helps Model Integrator by generating unique ids for internal use for those model elements that would otherwise not require them. For Model Integrator, an element with a unique id is easier to merge.

Rational Rose RealTime Extensibility Interface (RRTEI) users will find traceability easier when they set this option. Unique ids improve the traceability of model elements of other tool integrations that use RRTEI.

It is necessary to plan and choose when to incorporate the new unique ids into the project model since virtually all controlled units will be modified implicitly. Additionally, the generated new ids are dependent on time and location. For example, generating unique ids for a given model at different times or on different machines produces different ids.

The following model elements do not have unique ids, unless you set the **Generate unique identifiers for all elements** option:

- Protocol In Signals ()
- Protocol Out Signals ()
- States (CompositeState)
- Capsule Roles (CapsuleRole)
- Ports (Port)
- Port Roles (PortRole)
- Capsule Structure diagram (CapsuleStructure)
- Classifier Role (ClassifierRole)
- Transitions (Transition)
- Junction Point (JunctionPoint)
- Choice Point (ChoicePoint)
- Connectors (Connector)
- (Guards)
- (Events)
- (EventGuards)
- Parameters ()
- Element hyperlinks (ExternalDocument)

Note: We strongly recommend that any team using multi-stream development use the **Generate unique identifiers for all elements** option.

To set the Generate unique identifiers for all elements option:

- 1** In the **Model View** tab in the browser, right-click on **Model**, and then click **Open Specification**.
- 2** Set the **Generate unique identifiers for all elements** option.
- 3** Click **OK**.

Setting this option creates unique ids for model elements that currently do not have them. This typically affects most of the model so that you are prompted to check out those model components when setting this option.

When saving the model, the size of the affected file increases by approximately 20%, and the time to load the model also increases.

Note: Set this option in the base model before branching development streams.

For existing projects, collapse all streams to a single unique id stream, set the **Generate unique identifiers for all elements** option, and then branch the developer streams.

Figure 9 Figure 9 Incorrect Merge Scenario

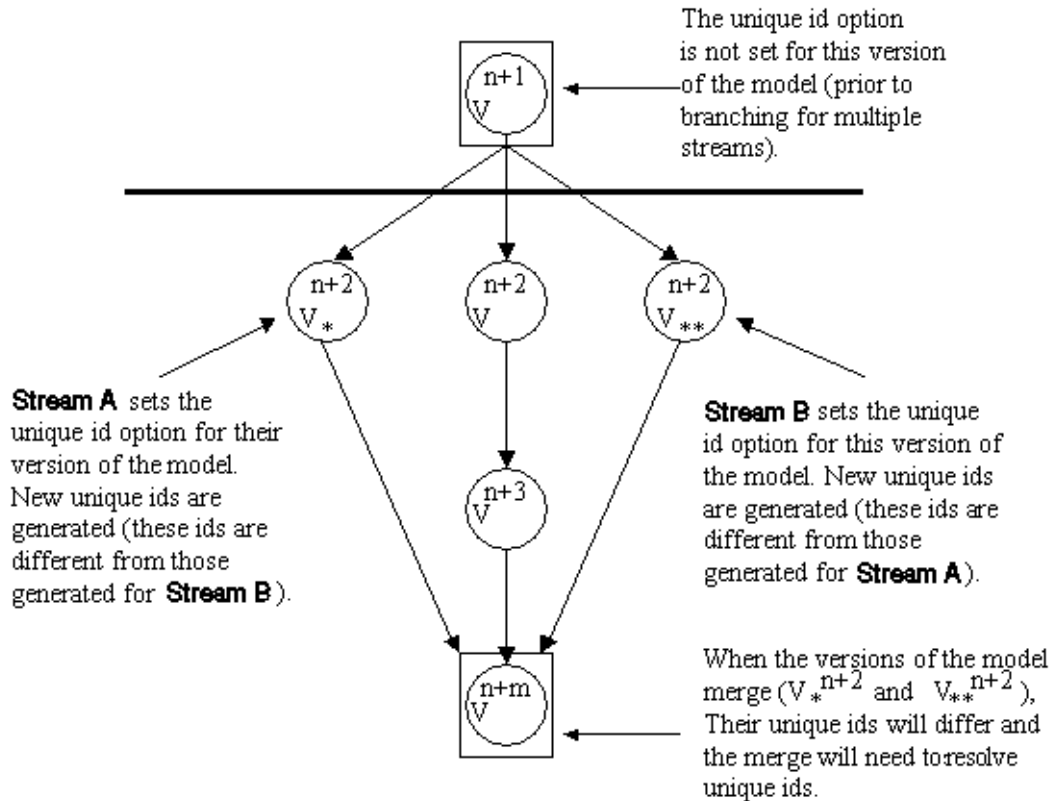
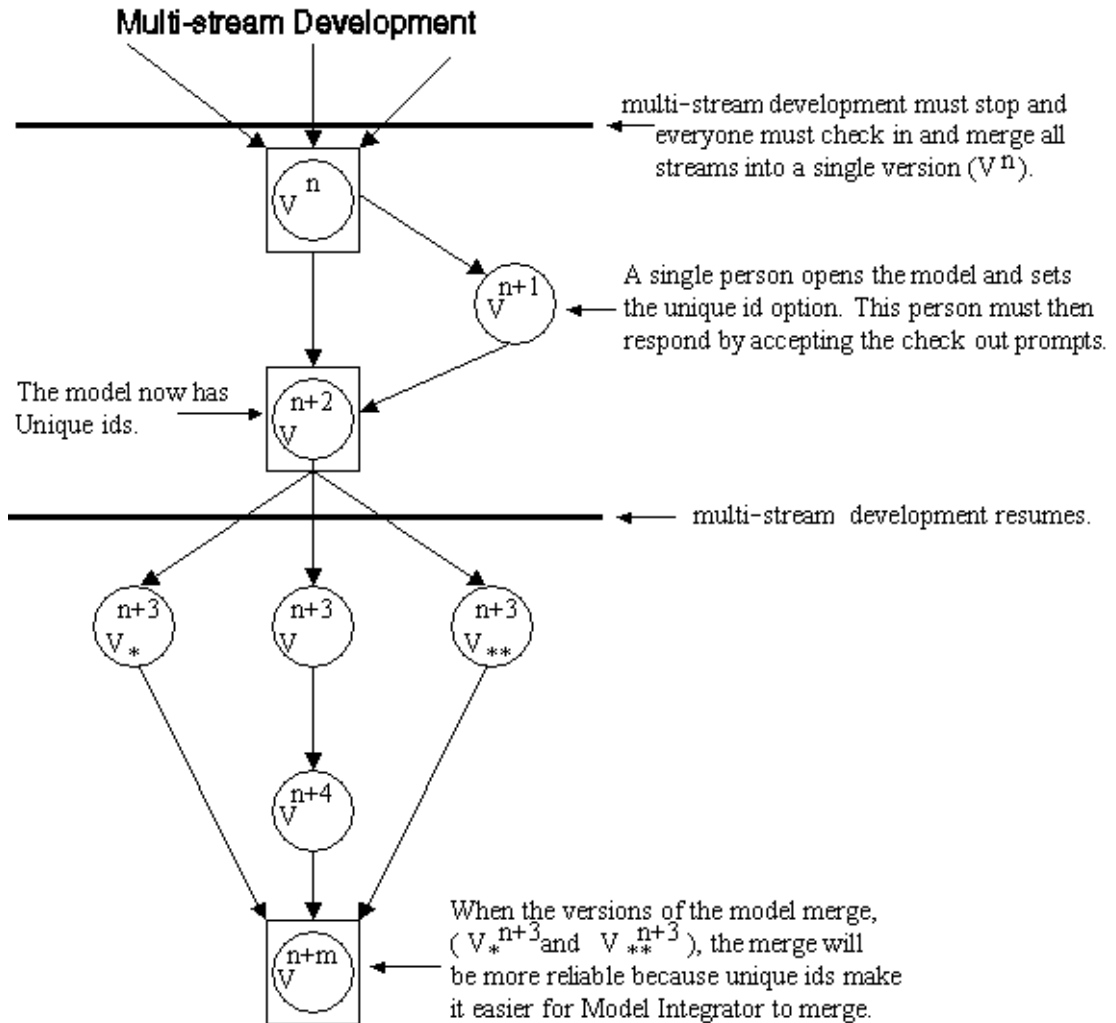


Figure 10 shows an example of when it is appropriate to set the **Generate unique identifiers for all elements** option.

Figure 10 Correct Merge Scenario



Note: This option must be set prior to branching.

For information on how to enable unique ids, see *Model Specification* in the online Help.

To clear the unique id option, follow the same procedure specified in Figure 10.

Note: If you clear this option, Model Integrator will merge based on the string names of the model elements. Merging will not succeed for situations where the names change or the names are not unique.

Considerations

- We recommend that you plan for conflicting merges and attempt to minimize them throughout the development life-cycle.
- Only merge controlled units with primary edits back into the integration stream.
- If you plan for a graphical change (a layout change) to a diagram within a model, only one person should make this change. This ensures that during the merge process, all of the graphical changes are accepted by one contributor and merging at a lower level of detail is not allowed.

Contents

This chapter is organized as follows:

- *Contributors* on page 57
- *Specifying Files in the Contributors Dialog Box* on page 58

Contributors

Contributors are the models that form the input to Model Integrator. Model Integrator accepts up to a maximum of seven contributor models for merging. The first contributor (Contributor 1) has special significance to Model Integrator; it is the base model used for comparing the differences between the other contributor models.

Note: All contributors must be of the same type. This means that, for example, you cannot compare a .rtmdl file with a .rtclass file. All contributors should be different versions of the same file.

A contributor can be any of the following:

- A model file, with or without its associated controlled units or subunits.
Note: If you specify a model file (.rtmdl) as the contributor, and this model contains subunits, you are prompted to load its subunits.
- A controlled unit of a model.

For a detailed list of controlled units, see *Understanding Subunits and Controlled Units* on page 44.

Note: You can specify a single controlled unit as a contributor.

Specifying Files in the Contributors Dialog Box

The easiest way to specify contributor files is to drag-and-drop the files from the Windows Explorer onto the Model Integrator window (Windows platform only). If the **Contributors** dialog box is not open, Model Integrator will open it for you. If it is currently open, drag-and-drop the files into the **Contributors** dialog box, not the main Model Integrator window.

To specify files to view, compare, or merge:

- 1 In Model Integrator, click **File > Contributors**.


By default, the **Compare/Merge Against Base Model** option is selected. When selected, the first file that you specify in the list must be the base model. For additional information, see *Base Model* on page 59.


Note: Model Integrator can provide a base model for you if you do not have one to use. For additional information, see *Merging Models Without a Base Model* on page 73.

- 2 Determine the files that you want to view, compare, or merge.

Note: You can select a maximum of seven files.

- 3 Do one of the following to specify the first .rtmdl, .rtclass, .rtlogpkg, .rtcmppkg, .rtdeploy, .rtcollab, .rtdeploydgm, .rtclassdgm, .rtcmpdgm, .rtcmp, .rtprcsr or .rtdev file in the **Files** list.

- Specify the fully qualified file name directly in the blank area of the **Files** box.
- Click **Browse** () and use the file browser to find a file to add to the list.



- 4 Click below the previously selected file (in the white space), and then click () to create a new file input field.

- 5 Repeat Step 1 through Step 5 until all files are specified.

- 6 Click the type of mode to use:

- View Mode. See *Merging Models* on page 72.
- Compare Mode. See *Comparing Models* on page 71.
- Merge Mode. See *Merging Models* on page 72.

Note: If you select a single file and the **Compare/Merge Against Base Model** option is not selected, the **View** button is available for View mode. If you select more than one contributor file, the **View** button changes to **Compare**. You may have to click in the empty area in the **Base** box for the **Compare** button to appear on the **Contributors** dialog box. If you select a single file, the **View** button is available for View Mode.

If the first file listed is not the base model, you can use the move buttons ( ) to change the order of filenames in the **Files** list so that the base model is first. Select a file by clicking on the name, and then click a move button to change the order of the files.

Base Model

The base model is the ancestor to the other contributor models being merged. That is, the base model is the version of the model that existed before any changes were made. When used, the base model must always be specified as the first contributor, Contributor 1.

Node


A node is another name for an object in the model hierarchy. Examples of nodes are classes, use cases, objects, operations, components, and diagrams. Nodes are displayed in the **Browser View**. Each node has properties that display in the **Property View** when the node is selected from the **Browser View**.

Contents

This chapter is organized as follows:

- *Compare Mode* on page 61
- *Merge Mode* on page 61
- *Subtree Mode* on page 62

Compare Mode

Use Compare mode to scroll through the model and observe the differences between the contributor models. If you decide to merge the models, you can change to Merge mode () and continue, or you can exit Model Integrator without merging.


Merge Mode

In Merge mode, Model Integrator attempts to automatically merge models for you. Your next step depends on the results of the automatic merge.

In the bottom right corner of the main window, you will see a message in the status bar indicating the following:

```
Unresolved items nn
```

where *nn* is the number of unresolved items Model Integrator encountered when it attempted to merge the specified models.

If the number of unresolved items is greater than zero, you must resolve these items before the merge process completes. Use the **Next Conflict** button () to find the first merge conflict. Examine the contributors for this model element, determine what action to take, and accept your choice to resolve the conflict. For detailed information on merging, see *Comparing and Merging Models* on page 71.

If the number of unresolved items is zero, you can save the model. Before saving your model, Model Integrator checks the model for errors. If it encounters errors, you must correct them before the model can be saved.

Note: The **Merge Errors** dialog box has the tools and topics to help you correct these problems. After you are finished, close the **Merge Errors** dialog box and click **Save**. Model Integrator prompts you to specify the location to save the main model file (the merged results file).

If your model has subunits and you loaded them, the **Subunits** dialog box appears to let you save the subunits. Click **OK** to continue the save operation. After the **Subunits** dialog box closes, the merge is complete and saved.

Subtree Mode

Subtree mode automatically applies Merge mode commands to both the current node and all of its children. Subtree mode is useful when you want to accept a group of related objects from a particular contributor. For example, you can accept an entire diagram from a contributor by selecting the top level node for that diagram, enabling Subtree mode, and then clicking **Merge > Resolve Selected Nodes Using**.

To activate Subtree mode, click  in the Toolbar, or click **Merge > Subtree Mode**.

When Subtree mode is not selected, you can visit each subtree node and make independent choices for the contributor for each node. When Subtree mode is selected, Model Integrator automatically applies the selected command to all of the children of the current node.

Note: Subtree mode is very powerful; use it with caution. When the merge is complete, ensure that you select this option so that it is not set (you can click **Edit > Undo** to undo any unwanted changes).

View Mode

Model Integrator supports a View mode for viewing the contents of a single model file. The purpose of View mode is that Model Integrator shows you more information, such as QUIDS and mechanisms, that is not available for display in Rational Rose RealTime.

To view a single model file:

- 1 In Model Integrator, click **File > Contributors**.
- 2 Specify or select a single file in the **Contributors** dialog box.
- 3 Click inside the empty area in the Base box.

Note: You must click in the empty area in the **Base** box for the **View** button to appear on the **Contributors** dialog box.

- 4 Click **View**.

Note: If you have begun to enter a second filename, the button text will change from **View** to **Compare**. If the button text is **Compare**, but you entered a single file name, clicking the **Compare** button will continue to enter View mode.

Contents

This chapter is organized as follows:

- *Subunit Status* on page 66
- *Loading Subunits* on page 66
- *Setting a New Context for Subunits* on page 67
- *Understanding Subunit File and Path Names* on page 67
- *Pathmaps* on page 68
- *Saving Subunits* on page 70

Subunits and Controlled Units

Rational Rose RealTime stores all or part of a model in one or more files. If a model is divided into separate files, the files other than the main model file (.rtmdl) are called subunits. In a team environment, these files would be under version control. When the files are under version control, they are called controlled units. Subunits and controlled units define portions of the model upon which individual developers can work. Breaking up a model into controlled units allows a project team to develop a model in parallel.

Model Integrator refers to both controlled and uncontrolled units as subunits.

If one or more of the contributor files you specify have controlled units, Model Integrator displays the **Subunits** dialog box. From this dialog box, you can specify whether to load or not load (unload) those units before comparing or merging your files, and to save them again when you save the merged model.

Subunit Status

The **Status** column in the **Subunits** dialog box displays the subunit status for each potential subunit in the files you are loading or saving. The **Status** column can display the following status types when loading or saving subunits:

Table 3 Status Types for Loading and Saving Subunits

Subunit status	Loading	Saving	Description
loaded	X		This subunit was successfully loaded.
not a unit	X		This item is not currently a separate subunit. This model section is part of the parent unit.
> LOAD <	X		Model Integrator loads this entry when you click OK or Apply .
> SAVE UNIT <		X	Model Integrator saves this entry to a separate file when you click OK .
unloaded	X		This subunit will not be loaded.
> SAVE SHARED UNIT <		X	Model Integrator saves this entry to a separate file, shared within the merged model when you click OK . The parent unit must be controlled for this option to be available.
> DO NOT SAVE <		X	Model Integrator considers this entry as a controlled unit when saving the parent unit of this item but does not save the actual item.

Loading Subunits

Subunits for each contributor load separately. This means that you are presented with a separate **Subunits** dialog box for each contributor .rtmdl file that has subunits. For each item, you can change the **Status** between > **LOAD** < and **unloaded** by clicking on the item with your left mouse button.

By default, Model Integrator attempts to load all non-shared subunits for a model. Shared units are not loaded by default. If there are units that you do not want to load, click on the **Status** value to change the status to **unload**, and the subunit will be skipped. If you do not want to load any subunits, click **Cancel** in the **Subunits** dialog box.

After you address the items in the **Status** column in the **Subunits** dialog box, and click **OK**, Model Integrator attempts to load the subunits specified with the > **LOAD** < status. If there is an error and some of the subunits cannot be loaded, the **Subunits** dialog box appears.

For help with errors encountered when loading subunits, see *Resolving Subunit Loading Errors* on page 68.

Note: Model Integrator cannot perform reference checking for subunits that are not loaded.

For every contributor with subunits, a **Subunits** dialog box will appear. When you complete the final **Subunits** dialog box, Model Integrator immediately begins the Compare or Merge session.

Understanding Subunit File and Path Names

The **Subunits** dialog box displays two columns of path-related information about the subunits in this model. The **Virtual Path** column shows the value of the path stored in the parent model. This value may be an absolute path or it may contain a pathmap variable. The **Actual Path** column displays the path that Model Integrator uses to load the subunit.

If pathmap variables appear in the **Actual Path** column, you must use the **Pathmap** function to set a value for the pathmap variable. For additional information, see *Pathmaps* on page 68.

You can left-click on an item in the **Actual Path** column and directly edit the path name that Model Integrator uses to find the subunit.

When saving a subunit, we recommend that you define a pathmap variable (in the **Pathmap** dialog box) and set it to the value "&". Creating this type of variable prevents absolute path names from being stored in the .rtmdl file for the subunits, and this makes it easier to move the files to new storage locations in the future.

Setting a New Context for Subunits

The **Context** box at the top of the **Subunits** dialog box shows the default path that Model Integrator uses to substitute for the "&" pathmap symbol. For a discussion of how to use pathmap symbols, see the topic *Virtual Path Maps* in the Rational Rose RealTime online Help.

If you created models using a pathmap symbol, you can define the value of the symbol to be "&" in the **PathMap** dialog box. Model Integrator will replace the "&" symbol in the definition of a pathmap with the actual path specified in the **Context** box.

By default, the value of the **Context** box is the path for the base model file (.rtmdl). If you move the files to a new location, you can change the contents of the **Context** box; Model Integrator will attempt to load the files from the new context.

You can select a new **Context** path by either specifying a new value directly in the **Context** box, or by clicking **Browse** to locate the desired drive and folder.

Resolving Subunit Loading Errors

If you specified a load status for an item in the **Subunits** dialog box and this load fails, Model Integrator displays the **Subunits** dialog box again for you to correct the problem. The **Status** column shows the current status of each subunit.

Note: You may need to scroll down the **Subunits** dialog box to find a subunit that was not loaded.

Items that continue to display > **LOAD** < in the **Status** column were not loaded. To resolve the problem:

- You can directly edit the **Actual Path** column to change the path for that particular subunit.
- If the subunit is utilizing a pathmap variable, you can change the value of the pathmap variable by clicking **PathMap** on the **Subunits** dialog box and modifying the variable in the **Virtual Pathmap** dialog box. For additional information on pathmaps, see *Pathmaps* on page 68.
- Select the subunit from the list and click **Browse** to search for the file. Select a file and click **OK**. The filename will appear in the **Subunits** dialog box.
- You can change the current directory for pathmap variables that take the value "&" by changing the **Context** box located at the top of the **Subunits** dialog box.

Note: A leading "&" in a path map definition represents the folder where the enclosing controlled unit or model file is located, and can be used to reference units relative to model files and other units.
- You can decide not to load the subunit. Click on the **Status** field for the subunit to change the status from > **LOAD** < to **unloaded**. This subunit will not be included in the merge.

Pathmaps

When controlled units are created in Rational Rose RealTime, the names of the subunits are stored in the main model file (.rtmdl). To avoid storing absolute path names in the model file (and making it difficult to move them to a new location later), Rational Rose RealTime provides the pathmap facility. Pathmaps let you specify a variable name used as the path that prefixes the location of a file. The pathmap

variable is stored in the model instead of the absolute path. You can then move the main file and its subunits to another storage location without having to edit the path names stored in the model.

How Do Virtual Pathmaps Work?

When Rational Rose RealTime saves a model, it attempts to substitute every absolute path with a virtual path. When Rational Rose RealTime opens a controlled unit, each virtual path is transformed into an absolute path.

For example, if you define the following virtual path:

```
$MYPATH=Z:\ordersystem,
```

and save a package as:

```
Z:\ordersystem\user_services.cat
```

The model file will refer to the package as:

```
$MYPATH\user_services.cat
```

When another user defines \$MYPATH as:

```
$MYPATH=X:\ordersystem
```

If they open the same model from their "X" drive, Rational Rose RealTime resolves the internal reference to the controlled unit and loads the following file:

```
X:\ordersystem\user_services.cat
```

When Do You Need a Pathmap?

Model Integrator shares pathmap variables with Rational Rose RealTime, and uses the same values transparently. However, Model Integrator may require you to enter a value for a pathmap variable if that variable was not previously defined on your computer. This situation is evident when you see a literal pathmap variable listed in the **Actual Path** column of the **Subunits** dialog box, such as C:\Models.

In the **Actual Path** column, when the pathmap variable \$ROSERT_HOME prefixes the path for each subunit, click **PathMap** and define a value for the pathmap variable \$ROSERT_HOME in the **Virtual PathMap** dialog box.

A typical (and useful) value to use is the "&" (ampersand) character. The "&" instructs Model Integrator to use the same path as the main model file uses (also known as the context path). After defining \$ROSERT_HOME, Model Integrator displays the actual path you defined, instead of the pathmap symbol.

Saving Subunits

When you save a model using **File > Save**, Model Integrator saves any subunits to the same location relative to the main .rtmdl file. When you click the **Save subunits in Root unit's folder** option in the **Subunits** dialog box, these subunits are saved in subdirectories whose name is that of the parent unit or model. Otherwise, Model Integrator uses the **childDirName** property and the **Subunit** dialog box is not displayed when saving. If you want to change the subunit configuration of your model, use **File > Save As** to display the **Subunits** dialog box. Using **Save As** allows you to:

- Save your existing subunits configuration by clicking **OK** in the **Subunits** dialog box.
- Create new subunits by clicking on the **Status** column for the subunit that you want to create. Model elements eligible to become subunits appear in the **Subunits** dialog box with **not a unit**. Click on this value to change it to **> SAVE <**. After you click **OK** or **Apply**, a new subunit is created.
- Eliminate subunits by clicking on an item in the **Status** column and changing the **> SAVE <** to **not a unit**. When you click **OK**, this part of the model is saved in the main .rtmdl file, instead of a separate subunit file.
- Save the shared unit by changing **> SAVE <** to **save shared unit**.
- Not save the unit while keeping it controlled, (different from **not a unit** which releases control of the unit), by changing **> SAVE <** to **do not save**.

Regardless of whether you use the **Subunits** dialog box, if you save subunits to a directory that already contains copies of the same subunits, Model Integrator warns you that you are overwriting the subunits, prompts you to continue, and then prompts you to overwrite the main model file (.rtmdl).

Contents



This chapter is organized as follows:

- *Comparing Models* on page 71
- *Merging Models* on page 72
- *Interpreting Compare and Merge Results* on page 74
- *Starting a Merge* on page 75
- *Semantic Checking* on page 76
- *Resolving Merge Errors* on page 78
- *Merging Options* on page 81
- *Differencing and Merging Model Elements* on page 84

Comparing Models

In Model Integrator, the purpose of Compare mode is to identify the differences between two or more models (maximum of seven). Conflicts are displayed as well, but in Compare mode, the Merge icons do not appear.

For a description of the icons that appear on the Model Integrator window after comparing or merging models, see *Interpreting Compare and Merge Results* on page 74.

In Compare mode, you cannot make any changes to the model, and the **Merge** menu and toolbar functions are disabled. You can easily change between Compare mode () and Merge mode (); this means that you can begin a work session in Compare mode, and later change to Merge mode if you decide to merge the models.

Note: If you change from Merge mode to Compare mode, and then exit Model Integrator, you are not prompted to save the work you did while in Merge mode.

Merging Models

Merge mode incorporates all of the features of Compare mode, along with additional information to support the decisions you need to successfully merge model files. By models files, we mean models or sub-models (controlled units/model files).

When merging, **we strongly recommend** that you divide models into controlled units, and then merge the controlled units for finer granularity. Merging entire models will likely cause merge conflict, complex merges, and require additional memory resources.


Note: Before merging models, check each model by clicking **Tools > Check Model** in Rational Rose RealTime. If errors are encountered in a model, correct the errors before performing a merge in Model Integrator.

Model Integrator supports two types of merge functionality:

- **Automatic Merge** - Model Integrator merges all changes that do not produce conflicts.
- **Selective Merge** - Model Integrator allows the user to optionally choose the contributors for every difference found between the models being merged.

Automatic Merge

Automatic merge takes affect when Model Integrator first enters Merge mode. It creates a recipient model and automatically merges all unchanged or minor changes to nodes into the recipient model for you. The **recipient** model is the model that holds the result of a Model Integrator merge session. When merging is complete, this model is given a name and saved to disk.

If the merged model has nodes that have conflicts, Model Integrator displays an icon () at the location of the conflict in the **Browser View**. As you make choices to resolve these conflicts, Model Integrator shows you the results of your merge.

For a description of the icons that appear on the Model Integrator window after comparing or merging models, see *Interpreting Compare and Merge Results* on page 74.

Selective Merge

The selective merge feature lets you change the contributor on nodes that have differences, as well as conflicts. This can be useful when you do not want to accept all of the changes that a contributor makes to your model. It is also useful when you need to correct more complex errors, such as those discovered by semantic checking. For addition information on semantic checking, see *Semantic Checking* on page 76.

Model Integrator merges models that have a common ancestor (that is, the base model). This is necessary when you keep your model under version control, and for parallel development, when changes are made in multiple streams. Model Integrator also supports the merging of models that do not have a base model.

Merging Models Without a Base Model


Model integrator allows you to merge models without having a common base model. Model Integrator automatically creates a base model that is empty. The base model will occupy the first item in the **Files** box on the **Contributors** dialog box (for Contributor 1)

Note: The base model it is not normally displayed and you cannot accept changes from it during the merge process.

To merge two files that do not have a common base model as an ancestor:

- 1 In the **File > Contributors** dialog box, select a single file and click in the empty area of the **Files** list.
- 2 Select the **Compare/Merge Against Base Model** option so that it is not set.
- 3 Click **Merge**.
- 4 Load the models as appropriate.

Note: Because a base model was not specified and the **Compare/Merge Against Base Model** option is not set, Model Integrator lets you specify a merge session with the minimum number of files (only two contributor files).

When merging models using this feature, all nodes in the contributors that do not conflict with each other appear with the  icon indicating that they are added to the merged model.

Understanding Differences and Conflicts

Model Integrator uses the concept of a base model to identify the kinds of changes that were made to the models during compare or merge. Each contributor is first compared to the base model. Model Integrator shows additions, changes, and deletions between a contributor and its base model as differences. Symbols identify the type of differences found. These symbols display in the **C** column in the **Browser View** and the **Property View**.

In Compare mode, Model Integrator only shows differences; but in Merge mode, Model Integrator also shows conflicts. A conflict occurs when there are two or more differences at the same node of the model. When Model Integrator finds a conflict, it

cannot determine which contributor to incorporate into the recipient model. Conflicts appear in the **M** column of the **Browser View**, along with other status information about the merge.

For information on the meaning of the icons for comparing (C column) and merging (M column), see *Interpreting Compare and Merge Results* on page 74.

In Merge mode, Model Integrator automatically incorporates differences into the recipient. However, it requires that you resolve conflicts by selecting the contributor from which to accept changes.

Model Integrator also supports comparing and merging models without using a base model as a reference point. However, in this mode, every node of the model shows as a difference.

Interpreting Compare and Merge Results

Model Integrator shows you the results of comparing or merging the contributing models by displaying an icon to the left of each node in the **Browser View**. Icons indicating the results of comparing models appear in the **C** column. Icons indicating the results of merging models appear in the **M** column.

Table 4 shows the status icons for comparing models, and Table 5 shows status icons for merging models.

Note: The icons for **differences** are **yellow**, and the icons for **conflicts** are **red**.

Note: The merge results do not appear in Compare mode.

Table 4 Status Icons for Comparing Models










Symbol	Description
No Symbol	A common item (same values in all contributors).
	New item added by a single contributor.
	Item deleted in a single contributor.
	Item changed in a single contributor.
	Item moved to a new location in a single contributor.
	Item added by multiple contributors (each having different property values).
	Item deleted in a contributor, item changed by another contributor.
	Item changed in multiple contributors.
	Item moved in a contributor, item changed by another contributor.
	Item moved in a contributor, item deleted by another contributor.

Table 4 Status Icons for Comparing Models






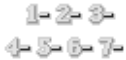
Symbol	Description
	Item moved by multiple contributors (to different locations).
	Item was resolved using merge source code.

Table 5 Status Icons for Merging Models

Symbol	Description
No Symbol	A common item and the recipient is set to the common property values.
	The recipient item is not set. This can occur because of an unresolved conflict, or after applying the Merge > Revert command.
	Item was resolved using merge source code.
	The recipient item is set with values from contributor <i>n</i> , where <i>n</i> is a number between 1 and 7.
	The recipient item is set for deletion by contributor <i>n</i> , where <i>n</i> is a number between 1 and 7 and where this contributor has no values set for the selected item, as indicated by the minus sign.

Starting a Merge

When Model Integrator starts Merge mode, it applies the **AutoMerge** procedure to the entire set of contributors. By default, Model Integrator uses automatic merge to merge all changes that do not produce conflicts into your merged model. You can also use the **Merge > AutoMerge** command to re-apply automatic merging to nodes, or a property of the model that you previously reverted using **Merge > Revert**. The **AutoMerge** procedure follows the rules illustrated in Table 6; a typical case using three contributors.

Table 6 AutoMerge Rules for Three Contributors

AutoMerge State	Contributor 1 (Base)	Contributor 2	Contributor 3	Result
No change	A	A	A	A
Added	--	A	--	A
Changed	A	A	A*	A*
Deleted	A	A	--	--

Table 6 AutoMerge Rules for Three Contributors

Conflict	a	a [*]	a ^{**}	?
Conflict	a	a [*]	--	?
Conflict	--	a [*]	a ^{**}	?

Where:

- A is a model element (for example, a class).
- a is a property (for example, a class visibility property).
- A^{*} is model element A, with one or more modified properties.
- -- indicates not present.

Note: Only the role of the base model is fixed in the **AutoMerge** procedure. The order of the other contributors does not matter. For example, changing Contributor 2 and Contributor 3 does not affect the results.

If a contributor that is not the base model introduces a change (that is, adds, modifies, moves, or deletes an object), that change is copied to the merged output instead of the original object. Model Integrator can select properties from different contributors and merge them. If two or more contributors change the same property, then the **AutoMerge** procedure does not know how to determine which one to choose. Instead it generates a conflict.

Semantic Checking

Semantic checking is a Merge mode feature that helps ensure that the merge choices you make are valid.

There are two forms of semantic checking available in Model Integrator:

- The first is performed by the **CheckMerge** function, called automatically before a merged model is saved. It cross-references all of the nodes of the recipient model to ensure that the final result is complete.
- The second form of semantic checking is an optional, real-time version of the **Check Merge** feature. This function checks references on the nodes as you access them, and it disables merge choices that would introduce errors into the model.

For example, a base model contains class A. Contributors 2 and 3 each make a change to one of the members of this class, while contributor 4 deletes the class. If you previously accepted changes from Contributor 4 to delete the class, you should not be able to accept one of the changes that Contributors 2 or 3 made to the class. However,

if semantic checking is not selected, Model Integrator allows you to make these contradictory changes. Model Integrator would not discover the problem until either you decided to save the recipient model, or you used **Check Merge** to verify the model.

In a large, complex model, it may be difficult to remember exactly which contributors present valid choices at a particular node of the model. This problem can also arise when Model Integrator makes automatic merge choices at nodes that do not have conflicts. If a node is deleted automatically, you may not be aware of that fact when you view a conflict at one of its dependent nodes. Semantic checking helps you avoid these problems by making your choices clear at each step.

When semantic checking is activated, and if you move the current selection to a new node of the model tree, the semantic checker determines which choices of contributor (if any) would result in an invalid model, if they were chosen you. These choices are then disabled in the GUI by disabling the appropriate menu items and Toolbar buttons.

When working with a very large model, you may want to make a change now and fix it later. In this case, semantic checking can be disabled and merge choices can then be made. After making the merge choices, select **Merge > Check Merge** to check and repair the model. The model is always checked for validity before saving.

Using Semantic Checking On-the-Fly

Model Integrator automatically performs semantic checking before you save the merged model (using **Check Merge**); however, you can also perform a check while you work. Click **Merge > Semantic Checking** to perform some reference checking when you select a new node in the **Browser View**. For this type of check, Model Integrator will disable those merge choices that will result in merge errors later in the session.

Click **Merge > Semantic Checking** when you want to avoid accepting changes that may produce errors. However, the checking performed by this function is not complete because it would take too long to check the entire model every time you select a different node. Consequently, Model Integrator may continue to find errors when saving, even if semantic checking is selected.

If the contributor you want to choose is currently disabled by Semantic Checking, you can either:

- Investigate the reason the choice is disabled by looking at the model elements referenced by this node (click **View > Referenced Node**) or the parents of this node, or its referenced nodes (click **View > Parent**). Typically, one of these nodes is already being deleted by another contributor. Choose a new contributor that does not delete the node, and click **View > Previous Location** to return to the original node and make the choice you want.
- Clear **Semantic Checking**, and make the choices you want. Rely on the **Check Merge** function to identify any errors when you finish your merge.

Limitations of Semantic Checking

For performance reasons, on-the-fly semantic checking is limited to checking only the nodes of the model that you are currently viewing. Consequently, it is necessary to perform a check of the entire model before saving it, and this check may reveal errors that require your attention.

References to subunits that are not loaded into the current merge session cannot be checked.

Resolving Merge Errors

Use the **Merge > Check Merge** command to check your merged model for internal consistency. Inconsistency can occur during a merge operation when, for example, one of the contributor models being merged deletes model elements used by one of the other contributors to your merged model. This can occur because of:

- Decisions you make when you resolve conflicts between contributors.
- Decisions made by the Model Integrator automatic merging feature.

The **Merge Errors** dialog box provides a set of tools that can help you find and correct errors that Model Integrator detected in the merged model. After clicking **Merge > Check Merge**, the **Merge Errors** dialog box appears only when there are errors with the merge. You must address these errors by resolving the contributor.

To resolve an error:

- 1 Select an error message in the list of errors in the **Merge Errors** dialog box.
- 2 To correct a merge error, you must select a different contributor for some node of the model. For additional information, see *Accepting Changes from Contributors* on page 81.

In the **Merge Errors** dialog box, you can find the node you need to change by selecting the following buttons:

- **View Error** - Takes you to the node of the model where the error was encountered, called the *error node*.
- **View Definition** - Takes you to the node of the model which defines the reference made by the error node.
- **View Parent** - Takes you to the parent of the currently selected node in the browser. Use this option to search for the parent of a definition node. Click this button when you have a node whose parent is deleted.
- **View Other Locations** - If the node you are viewing was moved to different locations by another contributor, **View Other Locations** will take you to one of the other locations where the node exists. Only one of these locations will actually exist in the merged output model; the other nodes are marked for deletion.
- **Refresh List** - Clears the error list and performs the **Check Merge** function again. If new errors are encountered, they will appear in the errors list. Use this command after fixing all the errors, because there may be additional errors in the merge that were hidden by the first set of errors. The same node of the model could have several errors, but only one is reported at a time. Occasionally, you will resolve one error, and that may resolve other errors as well.

The **Check Merge** function detects two types of merge errors:

- **This node references a node that is deleted.**

This message means that the error node references another node in the model that was deleted in the merged model. Click **View Definition** to display the location where the deletion occurred. You must resolve this error because the error node requires the other node to exist. To correct this error, choose either:

- A contributor at the definition node that does not delete the node.
- A contributor which deletes the error node (if one is available).

Typically, choosing the same contributor at both locations is the preferred solution.

- **This node references a node whose parent is deleted.**

This message means that the error node references another node in the model where one of the parent nodes of the defining node was deleted, rather than the defining node itself. When the parent node is deleted, all of its children are deleted as well. To change the parent node so that it is not deleted, click **View Definition** to go to the defining node in the model, then click **View Parent** to move up the model tree until you find the parent node being deleted. Choose a contributor for this node that contains a definition of the node rather than deleting it.

Merging Models with Controlled Subunits

Rational ClearCase and Model Integrator support the comparing and merging of individual model files or controlled units directly from ClearCase. This is desirable in a team environment (parallel development) because modelers only work with individual component files of the model.

For example, you can divide use cases into logical packages so that developers only need to check out the .rtlogpkg file that contains their use cases. They can branch these files privately, and subsequently merge them back into the main development branch without having to merge the entire model.

However, it can be desirable to merge the entire model because semantic checking works best when the whole model is loaded into Model Integrator. To accomplish this, construct a separate ClearCase view for each full contributor to the merge session. Each view is constructed to make the correct version of the model files for that contributor visible within the view. Check out the model files in the view which will receive the merge result.

Model Integrator starts, not from a **ClearCase** menu, but from the Rational Rose RealTime **Tools** menu or by the standard method for the system you use. The merge session proceeds in the same way it would if ClearCase were not involved. When completed, the merged model files are saved and checked back into ClearCase.


For instructions on how to configure ClearCase Integration, see **Team Development Guide > Source Control Tools > Rational ClearCase** in the online Help.


Merging Options

When merging in Model Integrator, you have the following merging options:

- *Accepting Changes from Contributors* on page 81
- *Deciding Which Contributor to Use* on page 82
- *Changing Nodes with Differences* on page 82
- *Reversing Changes to Nodes* on page 83
- *Performing a Partial Merge* on page 83

Accepting Changes from Contributors

The results of a merge appear in the main Model Integrator window. The  icon indicates a node that must be resolved before the merge can be completed. To resolve the conflict, you must specify the contributor to accept. The following commands allow you to accept changes from a contributor:

- Resolve all the remaining conflicts by clicking **Merge > Resolve All Conflicts Using**. This command lets you choose a single contributor to resolve all the remaining unresolved items. It operates over the entire merged model regardless of where you are when you select it. However, it only operates on unresolved conflicts. Nodes that you have previously accepted changes for, or nodes that are displaying only differences are not affected.
- Resolve an individual conflict or difference by selecting its node and then clicking **Merge > Resolve Selected Nodes Using** or one of the corresponding Toolbar buttons . This command copies one of the available contributor choices to the recipient. Unlike the **Resolve All Conflicts** command, this command operates on any node that shows either a conflict or a difference and overwrites previous choices.

You can also use this command with Subtree mode to resolve an entire subtree of model nodes at one time or with a set of nodes selected using the mouse and SHIFT and CTRL keys. The results of this command also affects all nodes displaying either conflicts or differences, and it changes values that were previously set.

Note: Subtree mode is very powerful. Use it with caution.

When semantic checking is selected, Model Integrator disables choices of contributors that may produce errors in the recipient model. If you want to make this change regardless of any potential errors that may result, you must click **Merge > Semantic Checking** so that it is not selected. You can always click **Edit > Undo** to undo any merge choices you make.

If you choose a contributor from the **Browser View**, all properties for that object are merged into the recipient. This becomes the default contributor for the object.

From the **Property View**, you can select and override the default contributor's property value for any desired property of the object.

Deciding Which Contributor to Use

The crucial issue in performing a merge is deciding the changes that you want to keep, and those that you want to discard. There are a few simple rules you can follow that will make this job easier:

- Merge often.
- Partition the activities and the model so that people can work on different parts without interfering with others. This will reduce the number of conflicts you have to resolve.
- Know the models that you are merging. You should try to know in advance which of the contributors you want to select for major components of the model, such as classes and diagrams. This will help guide the choices that you must make.
- You may encounter internal parts of the model that you do not necessarily understand; this means making merge decisions about these objects that are normally hidden. For the unfamiliar items, use the same contributor you selected for the items you are familiar with.


For example, you have a use-case with an associated interaction diagram, and you select Contributor 3 for this diagram (because it has the most recent set of changes). If conflicts arise among the hidden objects, such as the Mechanism or one of its components, that are also part of this use-case, select Contributor 3 for those objects as well. This will maintain consistency in the final merged model.

Changing Nodes with Differences

You can accept changes from nodes that do not have conflicts, but do have differences because Model Integrator (**AutoMerge** feature) already made a choice for you. The choice of the contributor is not shown in the **M** column of the **Browser View**, but you can see the current contributor by looking at the **Property View**. The **Recipient** column displays the values for the chosen contributor. The **AutoMerge** choice will be the contributor that is different from the others.

You can override the Model Integrator choice by selecting the node in the **Browser View**, and clicking **Merge > Resolve Selected Nodes Using** to select a different contributor. The effect of this command is to *not* accept the change because you are choosing a contributor that did not change the model. This is useful when, for example, you do not want to delete a model element deleted in one of the contributors. When you apply this command to a node with a difference, the **M** column shows the contributor that you chose for the result.

Reversing Changes to Nodes

If you change a node, you can click **Edit > Undo** to restore it to its original state. If you want to undo a change made earlier in your merge session without undoing all of the changes after this change, click **Merge > Revert Selection**. This restores a single node to the unmerged state. Clicking **Merge > Revert Selection** makes the node unresolved (whether it is a conflict or not). The **M** column for this node changes to display the  icon. For conflict nodes, this command removes your choice of contributor to resolve the conflict. For difference nodes, this command removes the **AutoMerge** choice made by Model Integrator.


The **Merge > AutoMerge Selection** command can only be applied to nodes that have been reverted. Applying the **AutoMerge** command to reverted nodes restores them to the state that they were in when the merge session started.



Performing a Partial Merge

You may have confined your editing in Rational Rose RealTime to only a part of the model, but when you load the model into Model Integrator, differences appear in other parts of the model that you did not expect. This is not an error on the part of Rational Rose RealTime or Model Integrator; it simply reflects the fact that the model is complex, and not necessarily organized in the way you might expect. However, you can restrict your merge session to a part of the model.

Note: To perform a partial merge, you must use a base model, that provides the output for the parts of the model that you do not want to modify.

To perform a partial merge:

- 1 In Model Integrator, start a new session, and specify a base model and contributors model in the **Contributors** dialog box.
- 2 Click **Merge** from the **Contributors** dialog box.
- 3 Click  (Subtree mode) on the Toolbar.
- 4 Select the root node of the model tree. This is the first node in the **Browser View**.
- 5 Click **Merge > Resolve Selected Nodes Using > Contributor 1**.

The base model is selected for all conflicts and differences in the entire model. The **M** column for the entire model changes to the  icon (nodes that were added by other contributors change to the  icon).

- 6 Select the part of the model that you want to actively merge. You can use Subtree mode if the area you want to merge consists of one or more subtrees. Otherwise, you can select portions of the model by pressing SHIFT and CTRL while clicking nodes that you want to select with the mouse.

Note: If you select with the mouse, ensure that you expand the model tree so that all nodes can be selected (click **View > Expand All**).

- 7 Click **Merge > Revert Selection** to this part of the model.

This part of the model will display the  icon for each node.

- 8 Click **Merge > AutoMerge Selection** for the same part of the model as in Step 7.

Now, you have restricted the **AutoMerge** function to a part of the model. There may be conflicts in the part of the model you reverted and automerged (if there are any at all). Complete your merge on this part of the model and save the model.

Note: **Check Merge** may find errors due to references to the parts of the model that you excluded from the merge with this procedure. If this occurs, you must resolve the reference errors; you may have to make changes outside of the area you have chosen to merge. You cannot save a merged model that has reference errors.

Differencing and Merging Model Elements

The local version of a unit may be compared to its previous versions that may exist in your source control tool. Click **Source Control > Show Differences...** to compare the local file with the most recent version under source control.

Similarly, if a unit is already checked out, a "Get" performed on that unit will prompt you to merge. To merge from the most recent version under source control, perform a "Get" on the desired checked out unit. To merge from a previous version, use the "Get" facilities provided in the **Show History** dialog box.

Contents

This chapter is organized as follows:

- *Filtering* on page 85
- *Searching for a Model Element* on page 86
- *Finding Nodes that Have Moved* on page 86
- *Finding Referenced Nodes* on page 86
- *Viewing Conflicts and Differences* on page 87
- *Viewing Conflicts and Differences with Auto Advance* on page 88
- *Viewing the Parent of a Node* on page 88

Filtering

When merging models, the filtering feature allows you to ignore small changes in the position of objects in a Rational Rose RealTime diagram.

To enable filtering:

- 1 Click **Options > Filtering**.

The **Filter Properties** dialog box shows several Rational Rose RealTime model properties.

- 2 Select any desired filter and set a corresponding value.

Differences between model files that are less than the **Filter Value** are not reported in the Model Integrator conflict and difference summaries and are not displayed with any conflict or difference icons. The value chosen for the output is the value found in the base model.

Note: Differences continue to display for the properties of view objects, other than the filtered properties. You may continue to observe differences in Rational Rose RealTime diagrams. With filtering enabled, you may lose small changes made to drawings because they are being filtered out.

Searching for a Model Element

To search for a particular node by its name in the **Browser View**, click **Edit > Search**.

The search starts at your current location in the **Browser View** and proceeds through all the nodes in the model that display in the **Browser View**. Use the **Edit > Expand All** to display every model object in the **Browser View**.

If the string is found, the browser window scrolls to display the desired node, and its properties display in the **Property View**. If the string is not found, you will hear a beep.





You do not have to specify the full name you want to find; Model Integrator performs the search by matching the string you specified against any part of the model element name. The search is not case sensitive.

Finding Nodes that Have Moved

Model Integrator can detect when a node in one contributor has moved to a new location in another contributor. You can move back and forth between all the locations where a node has moved.

Viewing Model Elements that Have Moved

When you merge models with elements that have moved, Model Integrator will display all the locations where the model elements could be placed by the different contributors. However, you can only keep one of these locations in the merged file.

When you see one of the status icons indicating that an item has been moved (, , , ), you can navigate between the different locations by clicking **View > Other Locations**. Every time you select this command, it cycles to the next location where a contributor placed the model element that you are viewing. If the model element has only one location, this command is not enabled.

Click **View > Previous Location** to quickly return to the node that you were previously viewing.

Finding Referenced Nodes

Many of the nodes in a Rational Rose RealTime model make references to other nodes in the model in one or more ways. Model Integrator provides commands to make it easy for you to find the parent of a node, and to find references from the current node in the model that are not in the immediate subtree containing the current node.

Viewing Nodes Referenced by a Node

It is not uncommon for a particular node of a Rational Rose RealTime model to reference other nodes in the model. To ensure consistency in your merged model, you may want to view these referenced nodes while making a decision about which contributor to select to resolve a given conflict. Also, if Semantic Checking is enabled and a choice of contributor is disabled, viewing the referenced nodes can often reveal why. The **View > Referenced Node** command makes it easy to view nodes that have one or more of the three common types of references: *client*, *supplier*, and *quidu*.

Note: For our purposes, these reference types are not important. They are used internally within the Rational Rose RealTime model and their meaning changes depending on the node viewed. The only real significance they have in Model Integrator is that they link two different objects in the model together.

Nodes that have these references appear in the **Property View**. To the right of the reference name is the name of the referenced node. You could scroll through the **Browser View** to find this node but clicking **View > Referenced Node** locates it more quickly and accurately.

When a node in the model contains any of these references, the **View > Referenced Node** command is active for the type of reference (*client*, *supplier*, or *quidu*). The context menu for each type of reference contains an entry for the recipient and each contributor (since the referenced nodes may be in different places in different models - one of the contributors may have moved them). If you or Model Integrator have already accepted a change for the referenced node, the **Recipient** menu becomes active. Typically, you choose to view this one because it will be saved in the merged model. If the Recipient choice is not active, that means the referenced node is an unresolved item.

Viewing Conflicts and Differences

Model Integrator includes commands to move you from one conflict or difference to the next, skipping over the intervening nodes that do not change. The **View** menu contains a number of options for navigating through conflicts and differences. Use these commands to help you view all the conflicts and differences in the merged model. These commands automatically expand the Browser view hierarchy to make the next conflict visible.

Viewing Conflicts and Differences with Auto Advance

Use **Auto Advance** to automatically move to the next conflict or difference after you accept a change. The function has three modes of operation:

- **Conflict** - Advances to the next conflict.
- **Differences** - Advances to the next difference.
- **None** - Does not auto advance.

You can change the **Auto Advance** setting by selecting your choice from the **Options > Auto Advance** menu.

The **Auto Advance** setting also affects the functioning of the commands for viewing conflicts and differences.

For additional information, see *Understanding Differences and Conflicts* on page 73.

When you load a set of models, the **Auto Advance** function is set automatically. If the models have conflicts, then the **Conflict** mode is set. If the model has no conflicts, but has differences, the **Differences** mode is set. If there are no conflicts or differences, the **None** mode is set.

Viewing the Parent of a Node

Except for the first node, every node in the model has a parent node. Usually, there is an important relationship between a node and its parent. For example, the parent of a **State** node is a **State Machine**.

While merging models, you may need to view the parent of a node that you are viewing, however, if the model is large, the parent node may not be visible on the screen. Click **View > Parent** to quickly bring the parent node into view. You can click **View > Previous Location** to quickly return to the node that you were previously viewing.

Using the Rational ClearCase Diff Merge Tool

10

Contents

This chapter is organized as follows:

- *Overview* on page 89
- *Merge Source Code Example* on page 90
- *Recommendations* on page 100

Overview

The Rational ClearCase **Diff Merge** tool enables you to compare two or more files by graphically representing the differences between them. Diff Merge enables you to resolve differences (for example, merging source code differences line-by-line) between contributors graphically and merges the results into a single file. This means that you can merge such things as state transition action code, guard code, capsule and class operations, state entry and exit code, and capsule header prefaces with greater detail (granularity).

Note: To use the ClearCase **Diff Merge** tool with Model Integrator (and to access the **Merge Source Code** command and button), you must have Rational ClearCase installed and configured on your computer.

When Model Integrator is used from the command line or as a ClearCase type manager, it will automatically resolve non-conflicting textual differences. Model Integrator can use the Rational ClearCase **Diff Merge** tool to resolve user-written textual changes, such as code and documentation, within a model file.

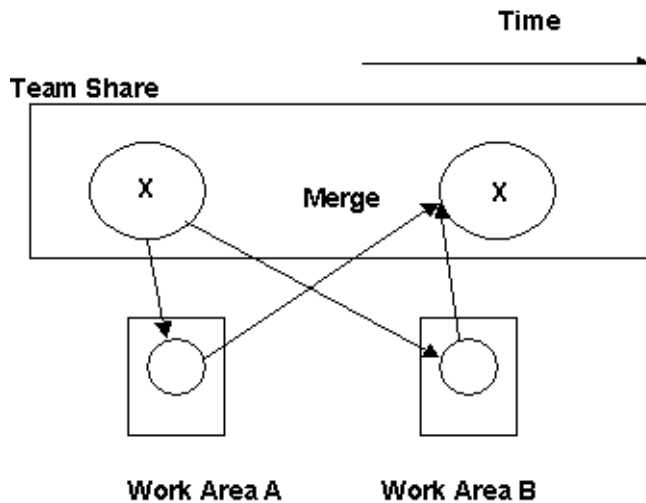
Merge Source Code Example

Note: Although the example used in this chapter merges models, we strongly recommend that you divide your models into controlled units and then merge.

The purpose of this example is to illustrate the flow of events for using the Model Integrator ClearCase **Diff Merge** tool. In this example, a software engineer uses Model Integrator to merge two models (or contributors) to a common root model (that is, the base contributor). Model Integrator will then call the ClearCase **Diff Merge** tool (Merge Source Code) to resolve the textual differences.

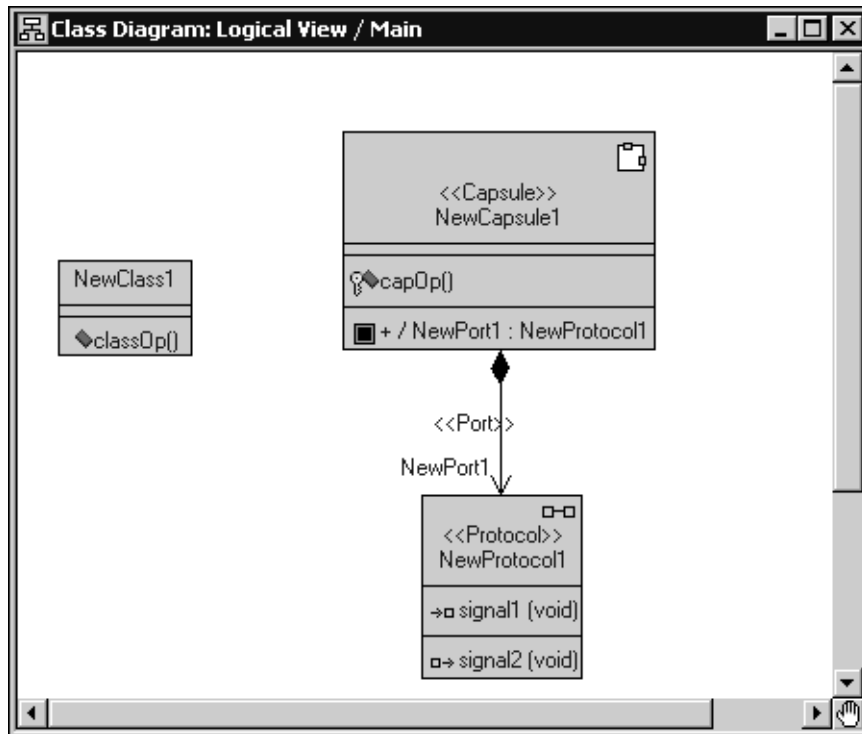
We want to use Model Integrator to apply a combined set of changes to a single base file. Figure 11 shows the merge scenario for our example; two different and conflicting model changes made to the same file. A file X (**modelBase**) is modified by in stream A. This modified file is called **modelContributor1**. A version of the same file is also modified in stream B. This modified file is called **modelContributor2**. In our example, we will demonstrate how to merge changes made to code for a transition.

Figure 11 Merging Changes - Parallel Development

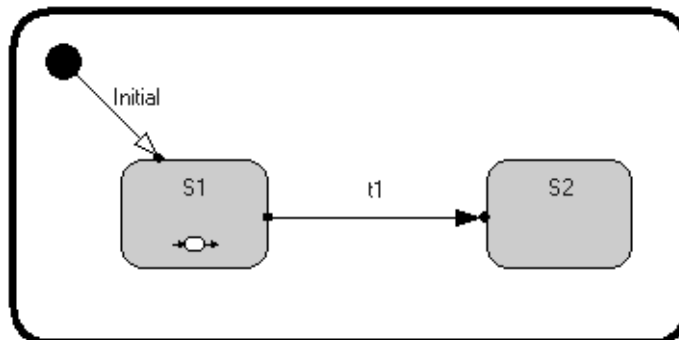


In our example, we use the following models:

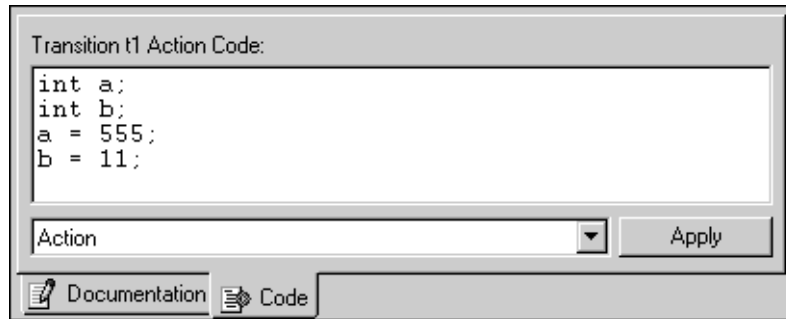
- 1 A **base model** - This model contains a single capsule.



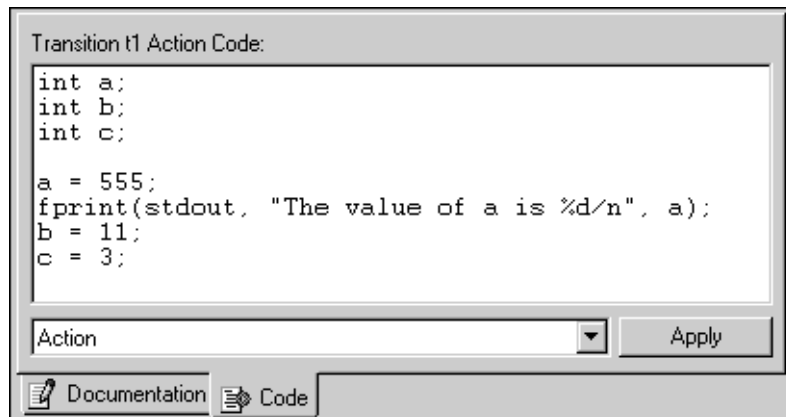
The **State Diagram** has an initial transition to the state called **S1**, and another transition named **t1**.



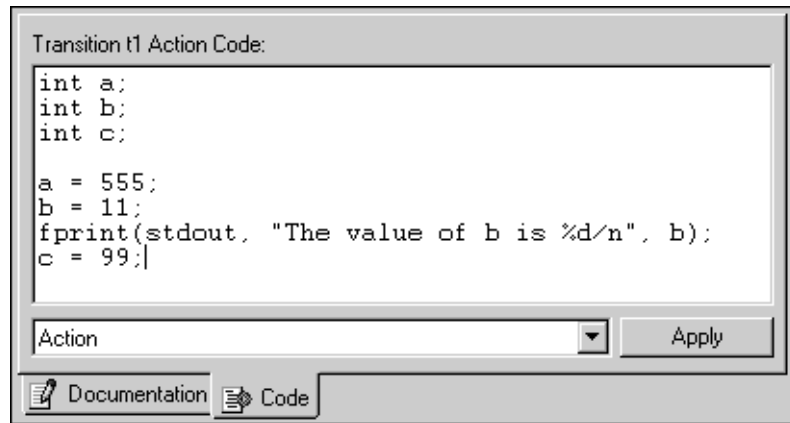
The code for the **t1** transition declares and initializes two integers as follows:



- 2 **modelContributor1** - This model is a different version of the base model with the following change (the **fprint** statement and initialization value for variable **c**) made to the **t1** transition:



- 3 modelContributor2** - This model is also a version of the base model with another change (a different `fprint` statement and a different initialization value for variable `c`) made to the code for the `t1` transition:



```
Transition t1 Action Code:
int a;
int b;
int c;

a = 555;
b = 11;
fprintf(stdout, "The value of b is %d/n", b);
c = 99;
```

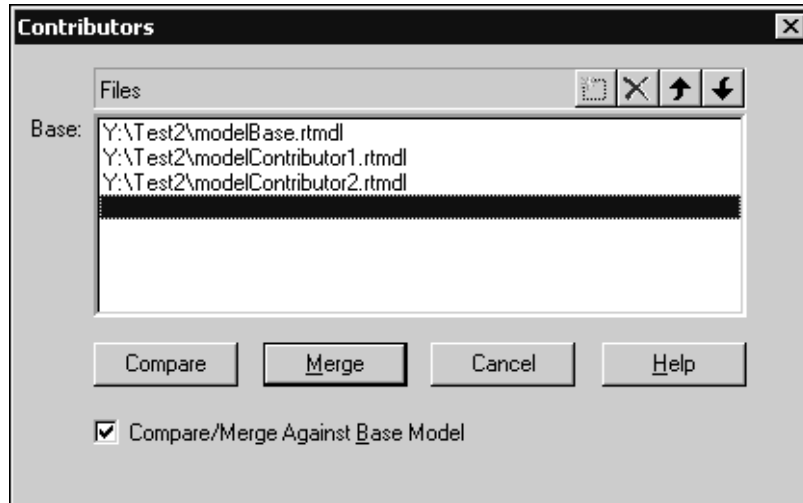
Specifying Contributors

In our example, we will select `modelbase.rtmidl` as the base model, and `modelContributor1.rtmidl` and `modelContributor2.rtmidl` as the contributor files.

To specify the contributor files:

- 1 Start Model Integrator.
- 2 Click **File > Contributors**.
- 3 Select the first contributor file. The first contributor has special significance to Model Integrator; it is the base model used for comparing the differences between the other contributor models (`modelContributor1.rtmidl` and `modelContributor2.rtmidl`).

- 4 Ensure that **Compare/Merge Against Base Model** is selected.

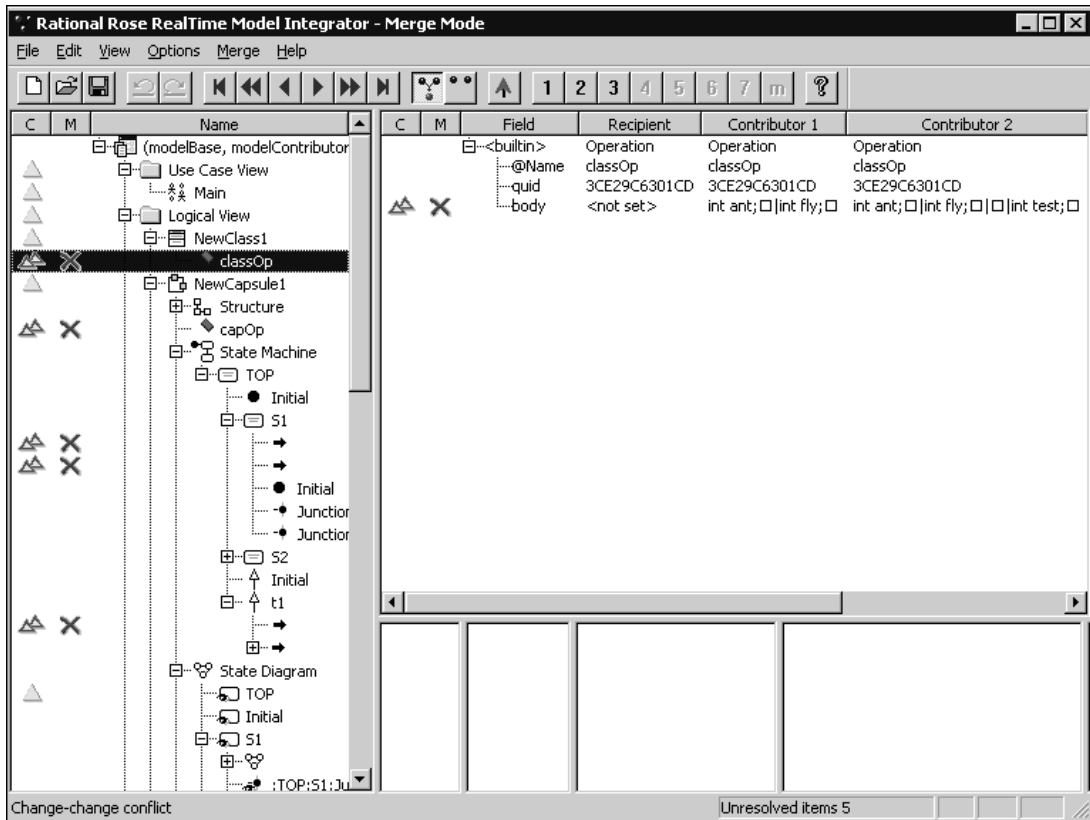


- 5 Click the **Merge** button.

Note: If the **Subunits** dialog box appears, load or unload the appropriate units. For information on how to load and save subunits, see *Loading Subunits* on page 66 and *Saving Subunits* on page 70.

After the main window appears, you are ready to begin working. Figure 12 shows the Model Integrator window after merging modelContributor1.rtmld and modelContributor2.rtmld with the base model modelbase.rtmld.

Figure 12 Model Integrator Window for Merge Mode



Note: The bottom right corner of the Model Integrator window indicates that it encountered five conflicts during the merge process.

Model Integrator shows you the results of comparing or merging the contributing models by displaying an icon to the left of each node in the **Browser View**. Icons indicating the results of comparing models appear in the **C** column. Icons indicating the results of merging models appear in the **M** column.

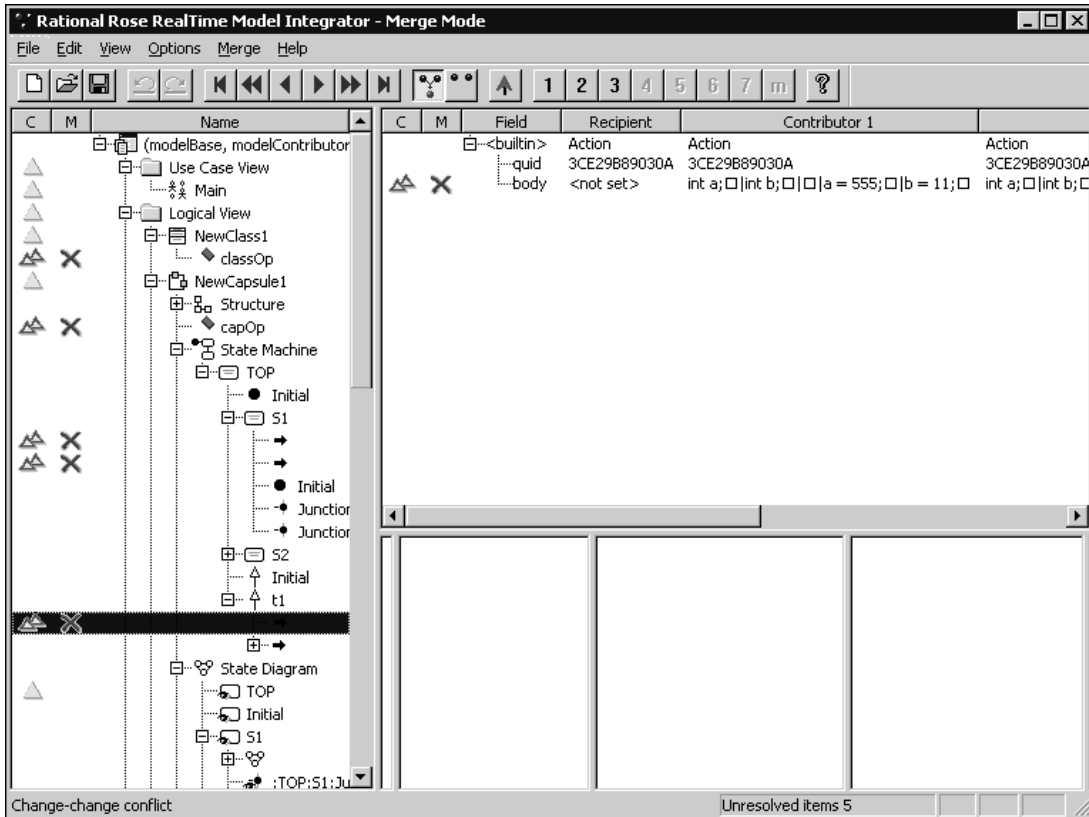
In our example, Model Integrator encountered a number of nodes that have model conflicts in the merged model (identified by the **X** icon at the location of each conflict). In particular, let us examine the changes made to the transition identified in our example. The software engineer performing the merge requires line-by-line examination of the detail code for the transition to determine the appropriate action to take. To achieve this type of granularity when merging, Model Integrator uses the Merge Source Code tool - ClearCase **Diff Merge**.

Starting the ClearCase Diff Merge Tool - Merge Source Code

After the Model Integrator main window appears, you are ready to begin working. For our example, we want to merge the code for the **t1** transition for the contributor models.

To merge source code:

- 1 In **Browser View**, select the conflict generated by the source code for transition **t1**.



The upper right window displays the properties conflict for the selected transition.

Note: If you select **AutoMerge**, Model Integrator will automatically call the ClearCase **Diff Merge** tool, without displaying the GUI, to resolve non-conflicting text merges.

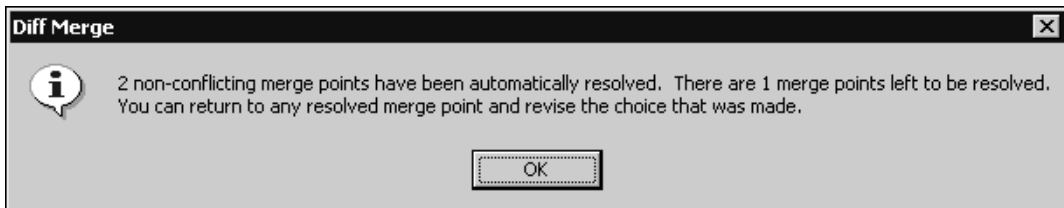


- 2 Select the property called **body**. The **m** button on the Toolbar is then enabled.
- 3 To launch the ClearCase Diff Merge tool, do one of the following:
 - Click **m**.
 - Click **Merge > Merge Source Code**,
 - Using the shortcut CTRL+SHIFT + M.

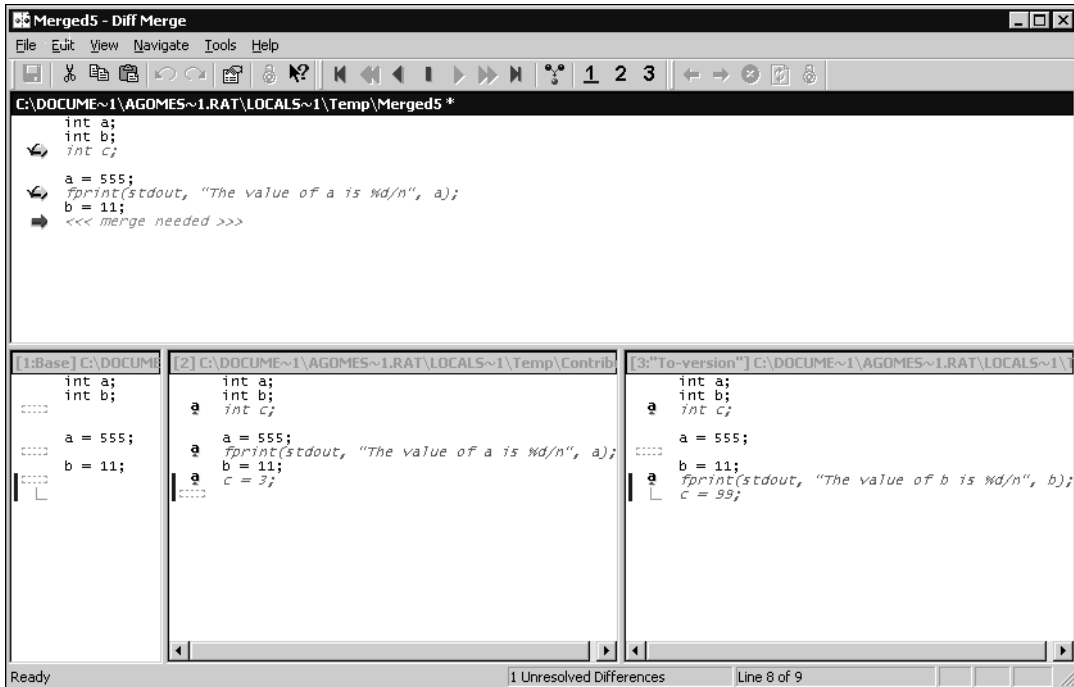
Now, you can begin merging the source code line-by-line for this transition.

The ClearCase **Diff Merge** tool prompts you with a dialog box indicating the number of conflicts that it was able to merge without requiring user intervention, as well as the number of merge points that need to be resolved.

Figure 13 Diff Merge Notification



4 Click OK to continue.



For information on using the ClearCase **Diff Merge** tool, see the Diff Merge online Help.

- 5 In our example, we will use the changes from contributor 3. Select the line containing <<< merge needed >>> and then click **3** from the Toolbar.

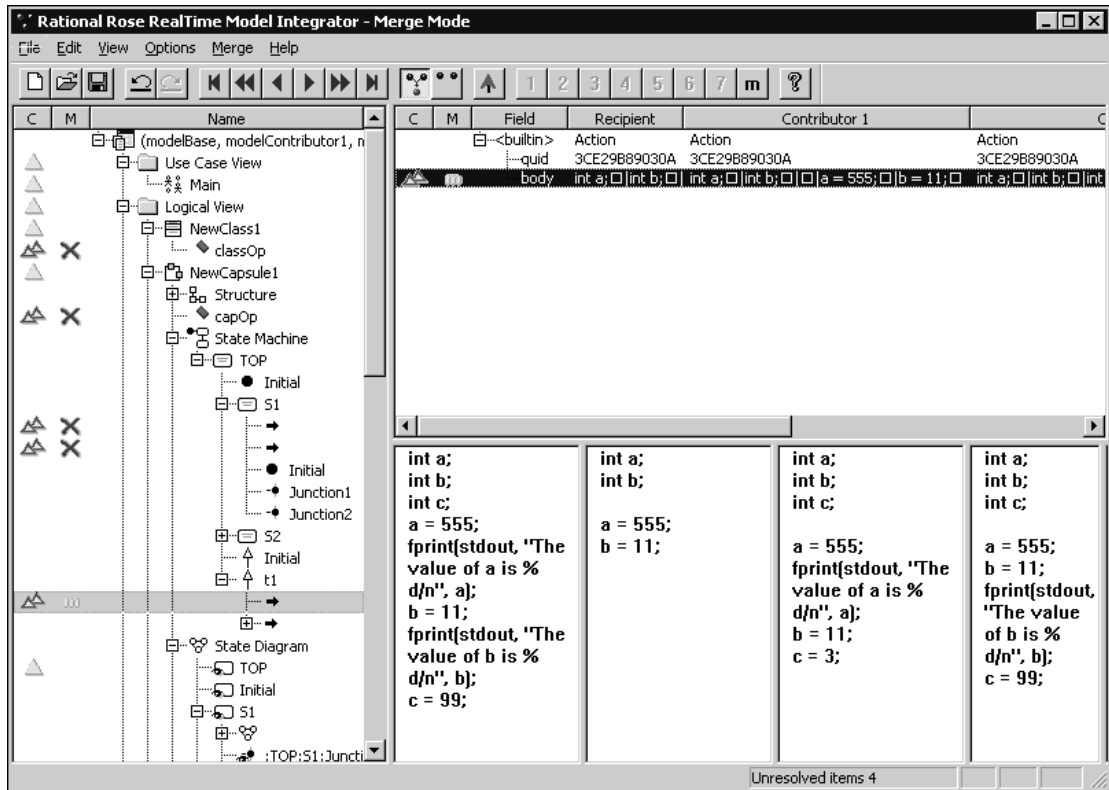
You can also resolve the conflicting lines by selecting the desired line from the **To-version** for a contributor, or you can add additional lines directly.

- 6 Click **File > Save** to save the merge source results, and then close the Diff Merge Tool.

If you close the window without saving the changes, the conflict will not be marked as resolved. If you save the results, the control returns to Model Integrator and the conflict is resolved when you close the **Diff Merge** tool. The resolved node is marked with an "m" (indicating merged source code), and the **Recipient** column in the Property window displays this merged source code.

The Recipient model will contain the following merged source code for the **t1** transition:

Figure 14 Model Integrator Merge Mode Window



Recommendations

- To obtain the best possible merge source code results, we recommend that the person performing this type of merge be an experienced programmer with basic knowledge of usage of the Rational ClearCase Diff Merge tool.
- After merging the source code, we recommend that you open the recipient model created by Model Integrator in Rational Rose RealTime to ensure that the model loads with no errors.
- Review the merge results and verify that the results are what you expected.
- Merge often.
- We strongly recommend that you always divide your model into controlled units.

Index

Symbols

- > DO NOT SAVE 66
- > LOAD 66
- > SAVE 70
- > SAVE SHARED UNIT 66
- > SAVE UNIT 66
- @Petal property 43

A

- accepting changes 81
- Actual Path 67
- adding
 - dependency issues 47
- all 23
- /all 23
- ask 23
- /ask 23
- Auto Advance 88
- Automatic Merge 72
- AutoMerge
 - rules 75

B

- base model 59
 - merging without 73
- basic objects 42
- Browser View 19

C

- changing language semantics 48
- childDirName 70
- ClearCase
 - Diff Merge tool 89
 - starting Diff Merge 96
 - using Model Integrator with 24

- ClearCase Diff Merge tool
 - recommendations 100
- code
 - merging 90
- command line
 - options list 23
 - starting Model Integrator (UNIX) 22
 - starting Model Integrator (Windows) 22
- command line options
 - commands list 22
 - Model Integrator 22
- command options 23
- commands
 - starting Model Integrator from the command line 22
- compare 23
- /compare 23
- Compare mode 61
 - Quick Start 21
- comparing
 - models 71
- composition of model files 41
- conflict 87
- considerations
 - for merging 45
 - improving Model Integrator performance 17
- contacting Rational customer support xiii
- contributor
 - types 57
- contributors 57
 - base model 59
 - determining which ones to use 82
 - merging without a base model 73
- controlled units 44, 65
 - (see also subunits) 44
 - file types 44
 - types 44

D

- dependency issues 47
- diagram objects 42
- difference 87
- differencing model elements 84

E

- enabling
 - filters 85

F

- filters
 - enabling 85
- finding
 - nodes that moved 86

G

- generating
 - unique identifiers for all elements 53
- GUI Features 20

I

- identifiers for all elements 53

L

- language semantics
 - changing (parallel development) 48
- loaded 66
- loading
 - errors (subunits) 68
 - status types for 66
 - subunits 66

M

- memory requirements for merging 16
- merge 23
- /merge 23
- merge
 - types 72
- merge errors
 - resolving 78
- Merge mode 61
 - Quick Start 21
- merging
 - before using Model Integrator 50
 - considerations 45
 - memory requirements 16
 - models 72
 - models with controlled units 80
 - options 81
 - performance 16
 - performing a partial merge 83
 - resolving errors 78
 - source code 90
 - starting a merge 75
 - types 72
 - using unique Ids 52
 - when is it necessary 50
 - without a base model 73
- merging model elements 84
- merging options
 - accepting changes from ontributors 81
 - changing nodes with differences 82
 - performing a partial merge 83
 - reversing changes to nodes 83
- model
 - generate unique identifiers for all elements 53
 - mechanism 42
 - recipient 72
 - unique Id 52
- model elements
 - differencing 84
 - find ones that moved 86
 - merging 84
 - searching for 86

- model files
 - base 59
 - composition of 41
 - types 44
 - versions 43
- Model Integrator 21
 - command line options 22
 - compare mode 61
 - comparing models 71
 - GUI 18
 - GUI for Compare Mode 18
 - GUI for Merge Mode 19
 - improving performance 17
 - merge mode 61
 - merging models 72
 - merging options 81
 - nodes 59
 - overview 15
 - starting 17
 - Subtree mode 62
 - using with ClearCase 24
 - view mode 62
 - views 19
 - when is merging necessary 50
- model specification
 - generate unique identifiers for all
 - elements 53
- ModelintRT 22
- modelintRT 22
- models
 - comparing 71
 - merging 72
- modes
 - Compare 61
 - Merge 61
 - Subtree 62
 - View 62

N

- Next Conflict 61
- nodes 59
 - changing 82
 - finding 86

- parent 88
 - referenced 86
- none 23
- /none 23
- not a unit 66

O

- objects
 - basic 42
 - diagram 42
 - quids 42
 - references 42
 - unnamed 43
 - view 42
- options
 - all 23
 - ask 23
 - command line 22
 - compare 23
 - merge 23
 - none 23
 - out 23
 - xcompare 23
 - xmerge 23
- out 23
- /out 23

P

- parallel development 45
- parent 88
- partial merge 83
- path names 67
- Pathmap 67
- pathmap
 - virtual 69
- PathmapsModel Integrator
 - Pathmaps 68
- performance 16
 - Model Integrator 17
- Property View 20

Q

Quick Start

Comparing models 21

Merging Models 21

quid 42

R

Rational customer support

contacting xiii

recipient model 72

referenced nodes 86

resolving

merge errors 78

merge wrrors 78

subunit loading errors 68

rtclass 44

rtcmp 44

rtcmpdgm 44

rtcmppkg 44

rtcollab 44

rtdeploy 44

rtdeploydgm 44

rtdev 44

rtlogpkg 44

rtmdl 44

rtprcsr 44

S

saving

status types for 66

searching 86

Selective Merge 72

semantic checking 76, 77

limitations (Model Integrator) 78

setting

context (subunits) 67

source code

merging 90

starting

Merge Diff 96

Merge Source Code 96

Model Integrator 17

Model Integrator from command line 22

starting ClearCase Diff Merge 96

status

subunit 66

types for loading 66

types for saving 66

Subtree mode 62

subunit

status 66

subunit file 67

subunit status

> DO NOT SAVE 66

> LOAD 66

> SAVE SHARED UNIT 66

> SAVE UNIT 66

loaded 66

not a unit 66

unloaded 66

subunits 21, 44, 65, 94

(see also controlled units) 44

loaded 21

loading 66

loading (status types) 66

loading errors 68

saving (status types) 66

setting new context 67

unloaded 21

T

team development 45

Text Views 20

tools

Merge Diff 89

U

unique identifiers for all elements 53

Unique Ids

correct merge scenario 55

incorrect merge scenarios 54

size impact 53

unique Ids 52

- unique ids
 - generating 53
- unloaded 66
- unnamed objects 43

V

- versions
 - model file 43
- View mode 62
- view objects 42
- views
 - Browser 19
 - Property 20
 - Text 20
- Virtual Path 67
- virtual pathmaps 69

X

- xcompare 23
- /xcompare 23
- xmerge 23
- /xmerge 23

