



Team Collaboration Guide

**Rational Rhapsody
Team Collaboration Guide**



Before using the information in this manual, be sure to read the “Notices” section of the Help or the PDF available from **Help > List of Books**.

This edition applies to IBM® Rational® Rhapsody® 7.5 and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1997, 2009.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Basic concepts	1
Project complexity	1
Number of team members	2
Number of components	2
Design complexity	3
Type of content management	3
Geographical distribution	4
Team members and their roles	4
Methodologies for team collaboration	6
Share by copy without a CM tool	7
Share by reference without a CM tool	8
Conventional CM tools	10
Accessing the CM archive from Rational Rhapsody	11
Rational Rhapsody files for content management	12
Model organization and partition	15
Possible model organizational methods	15
Test considerations	17
Configuration items (CIs)	18
Considerations for dividing a project into units	18
Creating unit files	19
Removing unit files	20
Packages as units	20
Classes as units	20
Diagrams as units	20
The project file	21
Multiple Rational Rhapsody projects	21
Dividing a project into two projects	22
Two projects into one project	23
Multiple project workflow	23
Repository structure	23
Repository structure planning	24

Table of Contents

Flat repositories	25
Hierarchical repositories	26
Example of project under CM.	27
Restructuring a project	28
File and directory creation.	29
File and directory deletion.	33
Unit storage.	33
Packages in a new directory	34
File renaming	34
Package contained in its own directory renaming.	35
Control moving a file or directory.	35
When Rational Rhapsody cannot update the CM system.	36
CM and Rational Rhapsody	39
SCC versus Batch mode.	39
Configuration Items window	40
CM operations	41
Connect to Archive	41
Connect to Archive in SCC Mode	43
Show Items in Archive.	44
Run CM tool	44
Comparing with the DiffMerge tool	45
Displaying the properties of a unit.	45
Synchronize Items.	46
Autosynchronize	46
Check out operation	47
Check In operation	48
Using Add to Archive in CM operations.	49
Lock and Unlock operations	50
Fetching a unit.	50
Using Uncheckout in CM operations.	51
History/Version tree.	51
CM status of units in a project	51
CM status information in the browser	52
CM status information in the Configuration Items window	52
Property to turn off display of CM status.	53
About troubleshooting CM operations	53
CM Output window	55
Pre- and post- actions.	56
CM interface extension	58

Unresolved references	59
Units added by reference	59
Multi-site collaboration	61
Webify for collaboration	61
Rapid prototyping	62
Parallel development	63
The DiffMerge tool	63
What is a unit?	63
How do you use DiffMerge?	64
Launching DiffMerge inside Rational Rhapsody	64
Compare With operation	66
Comparing two archived versions	67
Advantages of launching DiffMerge inside Rational Rhapsody	67
Launching DiffMerge outside Rational Rhapsody	68
Select units to compare	68
Selecting units to compare outside Rational Rhapsody	68
Advantages of launching DiffMerge outside Rational Rhapsody	69
Examining “left” and “right” value selections	70
Results displayed in the DiffMerge tool	71
Differences report in the Output window	74
Difference Report display	74
Features of a Difference Report	75
DiffMerge differences	76
Differences in the browser	76
Difference categories and their icons in the browser	77
Base-aware Diff icons	78
DiffMerge tool navigation	79
The external difference/merge textual tool	80
Using your external difference/merge textual tool	80
Filtering the comparison in the DiffMerge tool	81
Inspecting differences in diagrams visually	82
Graphical differences	83
Difference Report generation	85
Printing a Difference Report	86
Graphical differences suppression	86
DiffMerge limitations	87

Table of Contents

Logical versus graphical differences	88
Example of logical difference	88
DiffMerge reports	90
Exporting DiffMerge reports	90
The Rational Rhapsody DiffMerge process	91
How does the DiffMerge tool make a match?	91
Examples of how the DiffMerge tool handles renamed elements	92
How DiffMerge performs a model comparison	96
How differences are detected in base-aware comparisons	96
Limitations for match by element ID in DiffMerge	97
How to examine only major structure differences	98
Merge units with the DiffMerge tool	101
Starting a merge operation	101
Merge renamed elements	104
Saving the merged unit	105
Merge units limitations	105
Automatic merging for base-aware comparisons	106
About making merge decisions	108
Merging diagrams graphically for most diagrams	111
Merging diagrams graphically for statecharts and activity diagrams	112
About merging sequence diagrams	114
Merge activity log	116
Producing merge reports	116
DiffMerge tool preferences	117
Changing preferences	117
Keywords	118
Colors preferences category	119
DiffReport preferences category	119
General preferences category	121
MergeLog preferences category	123
Suppressions preferences category	124
TextDiffMerge preferences category	127
Command-line options for the DiffMerge tool	129
Launching the DiffMerge tool interface using the command line	129
Launching the DiffMerge tool from the command line	129
DiffMerge command-line syntax options	130
IBM Rational Synergy	133
Setting up Rational Rhapsody for use with Rational Synergy	133
Rational Synergy and Rational Rhapsody	135
Using Rational Synergy with Rational Rhapsody	135

Connecting to the Rational Synergy archive	136
Creating new Rational Synergy tasks	137
Viewing the properties for a Rational Synergy task.	138
Working with a Rational Synergy task in Rational Rhapsody	138
Checking in Rational Rhapsody work	138
Rational Synergy and the Rational Rhapsody DiffMerge tool.	139
Customize Rational Rhapsody and Rational Synergy	139
IBM Rational ClearCase	141
Batch mode Versus SCC mode	141
The differences between the Batch and SCC modes	142
SCC mode or Batch mode?	145
SCC Mode or Batch Mode Summary	147
Setting up Rational ClearCase	148
Controlling case sensitivity in Rational ClearCase	149
About checking out Rational Rhapsody files	149
About setting up Rational Rhapsody projects for team members.	149
About adding new files to the archive	149
Rational ClearCase limitations with Rational Rhapsody	150
Rational ClearCase semantics	150
Evil twins issue	150
Integration issues	151
Hierarchical repository and Rational ClearCase	152
Changes to an existing directory structure	152
Limitations for changing an existing directory structure	153
Rational ClearCase Type Manager.	153
Setting up the Rational ClearCase Type Manager	154
Setting up the Rational ClearCase .magic file.	156
Rational Rhapsody models and changing the default properties	157
Code generation performance improvements	158
Forced check in of a package with unchanged subunits	158
When is a Rational ClearCase license consumed?	159
Customize Rational Rhapsody and Rational ClearCase	159
Checking out/Checking in a directory once.	160
Storing an existing package in a separate directory	160
Removing an existing directory for a package and reconciling its contents	161
Adding a unit to the CM archive automatically	161

Table of Contents

Serena PVCS Dimensions	163
Enabling a SCC-compliant CM tool	163
Access to Dimensions from Rational Rhapsody	163
Create the initial connection to the SCC tool	164
Creating the initial connection to the SCC tool in Dimensions	164
Add to SCC archive operation	165
Adding a unit to an SCC archive	165
Check out operation in SCC archive	166
Checking out a unit in SCC archive	166
Check in operation in SCC archive	167
Checking in a unit in SCC archive	167
Listing the archive in PVCS Dimensions	167
Fetching in Dimensions	168
Unchecking Out in Dimensions	169
Viewing the history of a unit	169
Viewing the file details for a unit	169
Customize Rational Rhapsody and PVCS Dimensions	170
Concurrent Versions System (CVS)	171
Sharing a Rational Rhapsody project in CVS	171
Checking out a Rational Rhapsody project from a CVS repository	172
Collaboration with other users in CVS	173
Repository synchronization in CVS	173
Updating a Rhapsody unit in Eclipse to the CVS repository	174
Adding a unit created in Rational Rhapsody to the CVS repository	175
Showing the history of a unit in CVS	175
Subversion (SVN)	177
Sharing a Rational Rhapsody project in Subversion	177
Checking out a Rational Rhapsody project from a Subversion repository	178
Collaboration with other users in Subversion	179
Repository synchronization in Subversion	179
Updating a Rational Rhapsody unit in Eclipse to the Subversion repository	180
Adding a unit created in Rational Rhapsody to Subversion repository	180
Showing the history of a unit in Subversion	180
IBM Rational Team Concert	181

How changes are accepted and conflicts resolved 182

Index 183

Table of Contents

Basic concepts

IBM® Rational® Rhapsody® Team Collaboration describes how multiple Rational Rhapsody users can collaborate as a team on Rational Rhapsody projects. This information is designed to assist users in a wide variety of situations. Not all topics apply to all readers. Use the topics in Team Collaboration to understand the terminology used to describe different teams and team members, and to determine where to find information that applies to your situation.

Project complexity

There are no formal definitions to classify projects in terms of size or complexity. However, a few guidelines are necessary to clarify terms used in Team Collaboration. Some topics focus on specific types of projects. This topic will help you determine the type of project you are working on and which topics will be useful to you.

Levels of complexity can be measured in all kinds of ways, including the number of team members or components, complexity of the design, integration of legacy code, type of content management used, and geographical distribution of the team.

Number of team members

There are many ways to determine the level of complexity for a project, one consideration is the number of team members involved.

Use the following table to determine the size of a project based on the number of team members.

Number of People	Project Size	Relevant Topics
1	Individual	<ul style="list-style-type: none">• Share by copy without a CM tool• Share by reference without a CM tool• Parallel development for information about the IBM Rational Rhapsody DiffMerge tool
2 to 8	Small	<ul style="list-style-type: none">• Share by copy without a CM tool• Share by reference without a CM tool• Parallel development for information about the DiffMerge tool
9 to 25	Medium	<ul style="list-style-type: none">• Conventional CM tools• CM and Rational Rhapsody
More than 25	Large	<ul style="list-style-type: none">• Conventional CM tools• CM and Rational Rhapsody

Number of components

There are many ways to determine the level of complexity for a project, one consideration is the number of team components involved.

The following guidelines determine the size of a project based on the number of components:

- ◆ **Small** for 1 to 5 components
- ◆ **Medium** for 6 to 15 components
- ◆ **Large** for more than 15 components

If you are managing medium or large projects, see [Model organization and partition](#).

Design complexity

There are many ways to determine the level of complexity for a project, one consideration is the design complexity.

The following guidelines may help you determine the complexity of a design for a project:

- ◆ **Low** for up to 5 packages, 20 classes, or 20 events
- ◆ **Medium** for up to 20 packages, 100 classes, or 100 events
- ◆ **High** for up to 100 packages, 1000 classes, or 1000 events
- ◆ **Extremely complex** for more than 100 packages, 1000 classes, or 1000 events

If you are managing medium or high complexity projects, see [Model organization and partition](#). If you are working with high and extremely complex models, see [Multiple Rational Rhapsody projects](#).

Type of content management

There are many ways to determine the level of complexity for a project, one consideration is the type of content management involved.

The following guidelines consider three categories of content management:

- ◆ **None** for a content management process that does not use a CM tool.
- ◆ **Simple** for a simple CM tool that supports basic versioning features, but does not include complex operation such as branching. See [CM and Rational Rhapsody](#).
- ◆ **Complex** for an advanced CM tool that supports branching and features such as extensibility and integration with process control. See [CM and Rational Rhapsody](#) and in particular [CM interface extension](#).

Geographical distribution

There are many ways to determine the level of complexity for a project, one consideration is the geographical distribution of the team members involved.

The following guidelines consider the proximity of team members to one another:

- ◆ **Single site** when all team members are located at the same site.
- ◆ **Multiple sites** when team members are distributed between several sites.
- ◆ **Multiple remote sites** when team members are distributed between several sites with large time differences.

For projects that are distributed between different sites, see [Multiple Rational Rhapsody projects](#) and [Multi-site collaboration](#).

Team members and their roles

To be successful, all projects from simple to complex require several organizational roles. In some cases, one person might perform more than one role. For example, the project manager might also be the architect; the configuration system manager might also be the integrator; and the developer of a component might also be the quality assurance person for that component.

Consider the following organizational roles for a project.

Project Manager

The *project manager* is responsible for assigning work, and defining and monitoring the schedule. This person defines the project scope, is involved in all key decisions, facilitates communication among team members, and might define the process or policies for CM. The project manager has the overall responsibility for the project.

The project manager will find useful information throughout Team Collaboration.

Configuration System Manager

The *configuration system manager* sets up all the aspects of the CM tool, ensures that the environment is running with Rational Rhapsody, maintains the system, and assists team members with issues concerning the CM tool. In addition, this person promotes baselines as new versions or releases arise. The configuration system manager might manage CM for more than one project or serve in other roles, depending on the organization.

The configuration system manager should review the following topics:

- ◆ [Example of project under CM](#)
- ◆ [CM and Rational Rhapsody](#) and in particular [CM interface extension](#)
- ◆ The particular CM tool: [IBM Rational Synergy](#), [IBM Rational ClearCase](#), [Serena PVCS Dimensions](#), [Concurrent Versions System \(CVS\)](#), or [Subversion \(SVN\)](#)

Architect/Lead Developer

The *architect/lead developer* is responsible for policies relating to project structure, dividing the project into several smaller projects, allocating components to projects, and allocating design units to components.

Architects should review [Model organization and partition](#).

Developer

The *developer* performs tasks from the basic (such as fixing a defect) to the more complex (such as working on a six-month project). Developer CM activities include joining a project, editing files on a local machine, verifying changes, submitting updated files to the CM system, and synchronizing a local workspace with the updates of other team members.

Developers should review the following topics:

- ◆ [Model organization and partition](#)
- ◆ [CM and Rational Rhapsody](#)
- ◆ [Parallel development](#)

Quality Manager

The *quality manager* defines quality assurance policies, develops and maintains test suites, executes testing, and takes responsibility for the overall quality of a project. This person might use the CM tool to track test cases or determine which version of the project is currently under testing analysis.

The quality manager should review [Test considerations](#).

Integrator

The *integrator* (“toolsmith”) creates the integration facilities, makefiles, and special dedicated scripts for the project. This person derives the formal, deliverable product from a CM baseline, arranges the installation and deployment procedures, and automates the process of periodic builds and tests.

The integrator might find useful information throughout Team Collaboration.

Methodologies for team collaboration

There are several ways to manage files so several team members can collaborate on the same project at the same time. Rational Rhapsody supports file management with or without the use of a CM tool.

There are two ways to manage files:

- ◆ Without a CM tool, including sharing by copy or reference
- ◆ With a CM tool, using either a CM interface included in Rational Rhapsody or a custom CM interface

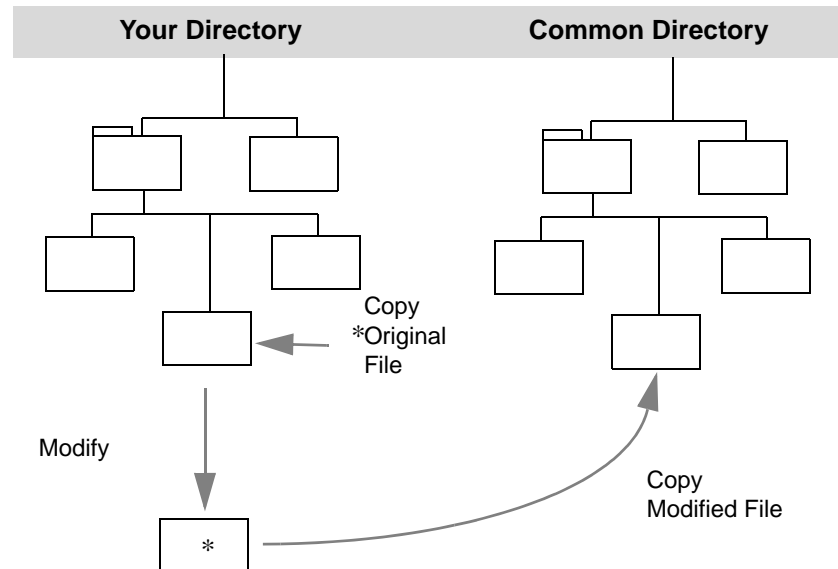
When you use a CM tool, the software takes care of most CM issues. However, a file management system can incorporate a combination of sharing by copy or reference and using a CM tool.

Software development teams with team members located remotely from one another face an additional challenge. In these cases, technologies that allow team members to collaborate over long distances (such as the Web) can be incorporated into the overall file management strategy. For more information, see [Multi-site collaboration](#).

Projects with complicated file systems and diverse teams often require creative solutions to CM that incorporate more than one method of team collaboration. For example, team members can check files in and out of a traditional CM tool, while at the same time the project can reference classes from another project or from other external specification files.

Share by copy without a CM tool

When sharing by copying, your directory contains a replica of the common directory (as shown in the following figure). You make your modifications to your local directory, then copy the file to the common directory to update the project. This strategy is appropriate for individual or small teams.



This technique has some obvious flaws. For example, there is no mechanism to prevent someone else from modifying the same file at the same time. This can cause a problem when the two team members copy files back to the common directory. The second developer overwrites the changes made by the first, thereby eliminating them. At some point, the two team members must compare their versions and merge the two sets of changes.

In addition, the copying process can be time-consuming for large projects. There is no method for parallel development or for tracking the history of changes made to the project.

This form of CM is done using the file system, not the Rational Rhapsody interface. However, you can use the IBM Rational Rhapsody DiffMerge tool to detect differences between the original file and the modified file, or to merge changes made to the same file by two different developers. For more information, see [Parallel development](#).

Share by reference without a CM tool

When sharing by reference, team members do not need to copy all project files to their local directories. Instead, they each have a copy of the `.rpy` file on their local machine. The `.rpy` file references the read-only project files on a common directory. When team members need to update a file, they add the file to the local machine, make the necessary changes, and then move the updated file back to the common directory.

By using references to the common directory, files are always up-to-date with the latest changes. However, processes that prevent two developers from localizing and modifying the same unit at the same time should be established and followed. In addition, there is no way to track changes made to a file, or to revert to a previous version when a problem is encountered.

Setting up a local workspace

To create a project that uses sharing by reference for CM:

1. Create a project directory on your local hard drive.
2. Make a copy of the `.rpy` file from the common directory to your local project directory.
3. Create a `<project>.rpy` directory inside the project directory. Your local file structure should resemble the following figure:



4. Start Rational Rhapsody.
5. Open the `.rpy` file on your local machine, selecting the **Without Subunits** check box. The project opens with all units marked as unloaded (U) and read-only (RO).
6. Open the Add to Model window. Choose **File > Add to Model**.
7. Locate the master project and select the master `.rpy` file.
8. Select the **As Reference** radio button, and click **Open** to open the Add to Model from Another Project window.
9. In the **Unit Type** drop-down list, select **All** and click the **Select All** button.
10. Select the **As reference** radio button.

11. To include associations, aggregations, dependencies, or similar relations, select the **Add Dependents** check box.
12. Click **OK**. All units in your local project are now read-only (RO), but they are no longer unloaded (U).

Editing files using sharing by reference

To edit a file in a project that uses sharing by reference:

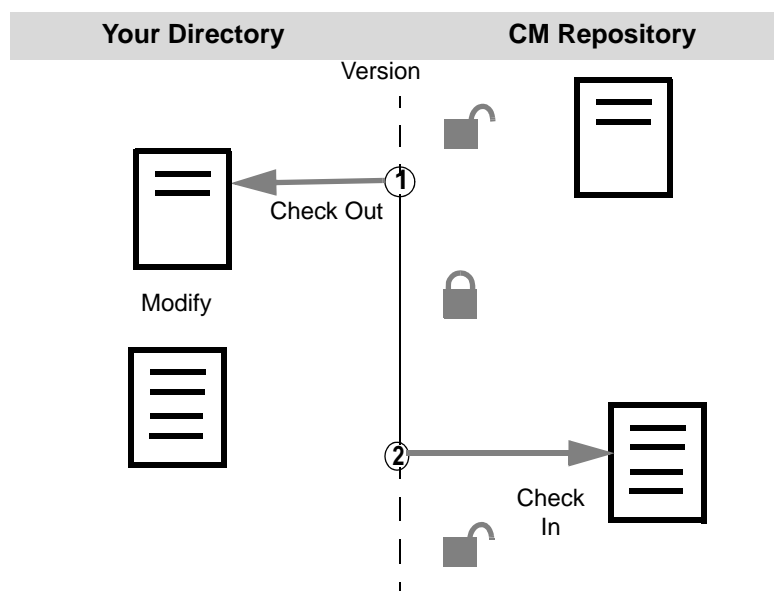
1. Start Rational Rhapsody and open your local project.
2. Open the Add to Model window. Choose **File > Add to Model**.
3. Locate the master project and select the master `.rpy` file.
4. Select the **As Unit** radio button and click **Open** to open the Add to Model from Another Project window.
5. Select the units you want to edit:
 - ◆ Use the **Ctrl** or **Shift** key to select multiple units. You can filter the units using the **Unit Type** drop-down list.
 - ◆ To include nested units, select the **Add Subunits** check box.
 - ◆ To include associations, aggregations, dependencies, or similar relations, select the **Add Dependents** check box.
6. Select the **As unit** radio button, and then click **OK**.
7. In the Add to Model window, select **Replace existing unit** and click **OK**. The selected units are loaded into your local project for editing.
8. Choose **File > Save** to save the added units on your local machine.
9. When you have completed the necessary changes, move the updated unit files to the master project in the common directory. The unit files should be removed from your local machine.

The next time you open this project, these units will be missing. Add them as references until you need to edit them again.

Conventional CM tools

Using conventional CM tools, you can check files in and out of a central repository. The tool tracks who has locked the file and protects it from being written to by other users.

One concept of content management using a CM tool is shown in the following figure. A file is stored in the CM tool repository. When you want to edit the file, you check it out and place a lock on the file. You make your modifications on your local machine, then check the file back into the CM repository, removing the lock. The CM tool maintains copies of both the previous version and the new version, and assigns a unique version number to the new file.



Rational Rhapsody provides an interface to all of the following concepts supported by CM tools:

- ◆ Locking files
- ◆ Viewing available versions
- ◆ File history
- ◆ Retrieving previous versions
- ◆ Setting baselines
- ◆ Process control

See the *IBM Rational Rhapsody Readme* file for the list of CM tools supported by Rational Rhapsody. In addition, you can develop a custom interface between Rational Rhapsody and unsupported CM tools.

Accessing the CM archive from Rational Rhapsody

Rational Rhapsody works with any CM archive in Microsoft® Common Source Code Control (SCC) mode (Windows only) or other tools in batch mode.

Rational Rhapsody developers might use any of the following CM tools to manage their source files:

- ◆ [IBM Rational Synergy](#) in SCC mode
- ◆ [IBM Rational ClearCase](#) in batch mode and SCC mode
- ◆ [Serena PVCS Dimensions](#) in SCC mode
- ◆ [Concurrent Versions System \(CVS\)](#)
- ◆ [Subversion \(SVN\)](#)

Note

For Linux users, you can use batch mode configuration management, and Rational ClearCase. Rational ClearCase is supported in Linux.

For more information about SCC mode and batch mode, see [SCC versus Batch mode](#).

To access your source controlled files from Rational Rhapsody:

1. Be certain that your source control archive is accessible from the PC you are using for this comparison.
2. Start Rational Rhapsody and open a project.
3. Open the Configuration Items window. Choose **File > Configuration Items**.

Note: The look of the window depends on the type of CM system you are using.

Limitation: The **Version** number of the controlled files does not display in this window.

Rational Rhapsody files for content management

In a Rational Rhapsody project, some files contain project data and others store local information that does not need to be shared with other team members. Only files that store project data need to be included in a CM system.

The following table lists each Rational Rhapsody project file, its purpose, and guidelines for placing it under content management.

File Name	Purpose	CM Guidelines
*.rpy	Rational Rhapsody project file	Maintain under CM from Rational Rhapsody.
unit files (* .sbs, * .omd, * .cls, and so on)	Files that store Rational Rhapsody elements, such as packages, diagrams, and classes	Maintain all unit files under CM from Rational Rhapsody.
*.rpw	User-specific workspace data	Does not require CM.
*.ehl	Events history list; stores animation commands, such as event generation	Does not require CM.
*.vba	A binary file that stores VBA macros	<p>If the project uses VBA macros, maintain this file under CM outside of Rational Rhapsody (from the CM tool).</p> <p>Note that most CM tools require a specific signal from the user when archiving a binary file.</p> <p>If this file is read-only, Rational Rhapsody displays the warning message, "Failed to open document" when you open the project, and "Failed to save document" when you save the project.</p> <p>If the project does not make use of VBA macros, there is no need to apply CM to this file.</p>
store.log	A log recording when the project was saved	Does not require CM.
load.log	A log of files loaded into Rational Rhapsody	Does not require CM.
ReverseEngineering.log	A log of reverse engineering activity	Does not require CM.
filestable.dat	Internal Rational Rhapsody cache file	Does not require CM.
*.cg_info	Stores information related to incremental code generation	Does not require CM.

File Name	Purpose	CM Guidelines
*_auto.rpy	Autosave file (optional, depends on project settings)	Does not require CM.
*_bak1.rpy & *_bak2.rpy	Backup project files created by Rational Rhapsody (optional, depends on project settings)	Do not require CM.

Model organization and partition

When you plan a Rational Rhapsody project, you need a design that facilitates team collaboration. Good model organization is crucial for achieving reusability in developing frameworks and components. Once you have decided on an organization, you need to determine how the model should be partitioned into units.

System organization enables many team members to contribute to the model without corrupting it or losing previous changes. It also provides the following benefits:

- ◆ Allows team members to work with parts of the model that lie outside of their responsibility
- ◆ Manages changes to pieces of the model
- ◆ Provides for an efficient build process
- ◆ Helps developers locate and work on various model elements
- ◆ Facilitates reuse of components
- ◆ Helps in developing versions and configurations of the product

Possible model organizational methods

Depending on the type of project, you can organize your model using one of several organizational methods.

Model organization by use cases

Use cases are central to gathering requirements and are an obvious focal point for organizing the requirements and analysis model. For example, when you are working on related requirements and use cases, you typically need access to one or more use cases and actors. When detailing a use case, you work on a single use case and detailed views (a set of scenarios) and often either an activity diagram or a statechart (or some other formal specification language). When elaborating a collaboration, you must create a set of classes related to a single use case, as well as refining the scenarios bound to that use case. Packages can divide up the use cases into coherent sets (such as those related by generalization, «includes» or «Extends» relations, or by associating with a common set of actors). In this case, a package would contain a use case and its actors, activity diagrams, statecharts, and sequence diagrams.

Model organization by framework

A framework-based model organization addresses some of the limitations of the use case-based approach. It is still targeted at small systems, but it adds a framework package for shared and common elements. The framework package has subpackages for usage points (classes that will be used to provide services for the targeted application environment) and extension points (classes that are grouped into subclasses by classes in the use-case packages). Note also that there are other ways to organize the framework area that also work well. For example, frameworks often consist of sets of coherent patterns; the subpackaging of the framework can be organized around those patterns. While this is particularly apt when constructing small applications against a common framework, the scheme does hamper reuse in some of the same ways as the use case-based approach.

Model organization by logical and physical architectures

Another approach is to break up the architecture into the logical (organization of types, classes, and other design-time model elements) and physical aspects (organization of instances, objects, subsystems, and other run-time elements). The logical architecture is often organized by *domains*, whereas the physical architecture revolves around components or subsystems. If you structure the model this way, a domain, subsystem, or component becomes a CI to be assigned to a single worker or team. If the element is large enough, it can be further subdivided into subpackages based on subtopic within a domain, subcomponents, or another criterion such as team organization.

Model organization by domains

You could also divide the model based on classes. A domain, as defined in the ROPES process, is a subject area with a common vocabulary, such as device I/O, user interface, or alarm management. Each domain contains many classes, and system-level use case collaborations will contain classes from several different domains. Many domains require rather specialized expertise, such as low-level device drivers, aircraft navigation and guidance, or communication protocols. From a workflow and logical standpoint, it makes sense to group such elements together because a single person or team will develop and manipulate them. Grouping classes by domains and having the domains be CIs might make sense for many projects.

Model organization by components

A model can sometimes be organized intuitively by components. A UML™ component in Rational Rhapsody is a basic building block used to define executables, libraries, and other physical binary deliverables. Each such component is compiled of code generated from model elements. The model elements that compose the component are called the *component scope*. Using top-level packages that include all the model elements mapped to a certain component creates a simple and easy-to-use structure. Note that this approach interferes with reuse of the same design elements in multiple components. However, efficiency can be achieved by component-level reuse; that is, assigning the design-level elements to be reused into a library, then using this library in multiple components.

Model organization by team members or groups

A simple solution would be to assign one package per team member. Everything that Sam works on is in `SamPackage`; everything that Julie works on is in `JuliePackage`. For very small project teams this is a viable model. But again, it brings up the question of what Sam should work on versus Julie. It can also be problematic if Susan wants to update a few of Sam's classes while Sam is working on others in `SamPackage`. Further, this scheme adds project team organization dependencies into the model structure, making it more difficult to make changes to the project team (such as assigning team members to another task) and also limits reusability.

Test considerations

Testing workflows can dictate model organization. Although testing teams require read-only access to the model elements, they need to manage test plans, procedures, results, scripts, and fixtures (often at multiple levels of abstraction). Testing typically occurs on primary levels: unit testing, integration, and validation.

Because unit-level testing consists mainly of white box, design, or code-level tests and often uses additional model elements constructed as test fixtures, it makes sense to co-locate them with the corresponding parts of the model. So, if a class `myClass` has testing support classes, such as `myClass_tester` and `myClass_stub`, they should be kept together, either within the same package or in another if a peer will do the testing (as long as it is a different CI from that of the model elements under test).

Integration and validation tests are not as tightly coupled as unit-level tests, but the testing team might construct model elements and other artifacts to assist them. Because the creators of the model elements do not typically do these tests, independent access is required, so they should be in different CIs.

Efficiently constructing and testing prototypes is a crucial part of the development life cycle. This involves both tests against the architecture (integration) and against the requirements (validation) for the entire prototype. There can be any number of model elements specifically constructed for a particular prototype that need not be used anywhere else. It makes sense to keep these near that build or prototype. Store test fixtures, to be applied to many or all prototypes, in a locale that allows independent access from any given prototype.

Configuration items (CIs)

To implement an infrastructure, you need to determine which model elements should be individual configuration items (CIs), because certain usage policies apply only if that model element is a CI.

A CI is any element stored in a separate file. The project is always a separate file. In addition, Rational Rhapsody allows you to store components, packages, classes, and diagrams (except statecharts and activity diagrams) as individual files.

It would be extreme for the entire model to be a single CI. In that case, only one person could update the model at any given time. The other extreme would be to make every element (every class and use case) a separate CI. Again, in simple systems where there are only a few dozen model elements, it would not be difficult to explicitly check out each element. However, this method does not scale well, even to medium-sized systems where you might have to list 30 or 40 classes to work on a large collaboration realizing a use case.

UML provides an obvious organizational unit for a C, the *package*. A UML package is essentially a bag into which you can throw semantic model elements such as use cases, classes, objects, and diagrams. So, although you might want to make packages CIs in your source control or configuration management (CM) system, you need to decide which model elements should go into one package versus another.

Considerations for dividing a project into units

Using Rational Rhapsody, you control the granularity level of the CIs or *units*.

By default, every package you create is a unit (a separate file on your file system). In addition, components and diagrams are units. By default, all other design elements *are not* units and are stored in the file of the parent unit. Therefore, a Rational Rhapsody model consists of the project file (*.rpy), package files (*.sbs), component files (*.cmp), and various diagram files.

However, you can override these defaults according to the organization and requirements for a project, either on a unit-by-unit basis or by establishing new policies for creating units. For example, you can choose to make a particular class its own unit. Alternatively, you can set up Rational Rhapsody so every new class you create is a unit.

Sometimes it makes sense to override the default settings and store a class or several classes in a separate file, such as when a package is maintained by two team members, who often have usage conflicts. While team member A works on the behavior of a certain element in the package as captured in a statechart of one of the classes, team member B defines a new family of types to be used by all the classes in that package. In this case, you might want to make the class with the statechart a separate unit.

This applies to diagrams as well. Rational Rhapsody stores diagrams as units, enabling you to apply changes to the diagram without changing the actual model elements that appear in it; changes to the colors, element layout, comments, and other graphic characteristics of the diagrams can be changed regardless of the unit status (read-only or read/write) of the elements it contains. However, the diagram is often just a “view” that reflects certain aspects of the package to which it belongs. In this case, changing these aspects (for instance, changing the multiplicity of a relation, or deleting a class) requires a change in the diagram, and vice versa. Because of the associative nature of Rational Rhapsody, some changes in the diagram require the design elements to be checked out (for example, adding relations and changing relations).

Sometimes it makes sense to override the default settings in the opposite direction, to reduce the complexity, both in the number of files and in the need to perform CM operations. For example, suppose you need to define 50 events in a package. To keep the design readable, you split these 50 events into three subpackages with meaningful names and appropriate descriptions. You might want to create the design packages, without ending up with three new files. In this case, you can override the default behavior to prevent the creation of packages as units.

In addition to defining units individually, you can use *properties* to define new policies for creating units. For example, you can prevent all new diagrams from becoming separate units. Or, you can override the default policy for classes so all new classes are automatically stored as units.

Creating unit files

To create a unit:

1. Right-click the element and select **Create Unit**.
2. In the Unit window, the **Store in Separate File** check box is selected by default. You can edit the default file name, but do not add a file extension.
3. Click **OK**.

Removing unit files

To change an element so it is no longer a separate unit:

1. Right-click the unit and select **Unit > Edit Unit**.
2. Clear the **Store in Separate File** check box.
3. Click **OK**.

The unit file is not removed from the file system, but the file is obsolete. The element is now stored in its parent unit.

Packages as units

By default, Rational Rhapsody saves all packages as units. To prevent packages from being saved as separate units, set the `General::Model::PackageIsSaveUnit` property to `Cleared`.

Classes as units

Rational Rhapsody does not save classes as units unless explicitly told to do so. You can store classes as separate files on a class-by-class basis using the **Create Unit** option. To have Rational Rhapsody automatically save all new classes as units, set the `General::Model::ClassIsSaveUnit` property to `Checked`.

Diagrams as units

By default, Rational Rhapsody saves all diagrams except statecharts and activity diagrams as units. You can change a particular diagram so it is stored in its parent unit using the **Edit Unit** option. To prevent Rational Rhapsody from automatically saving diagrams as units, set the `General::Model::DiagramIsSaveUnit` property to `Cleared`.

The project file

When you save a Rational Rhapsody project, you save a project file with the `.rpy` extension as well as supporting files.

The Rational Rhapsody project file contains two types of information. The first is a list of components, diagrams, packages, and so on that constitute your project. The second is a list of properties that you have overridden at the project level.

Note

Project-level properties encompass technical aspects that apply to all elements in a project, such as the CM tool, default editor, autosave preferences, and font settings.

The project file (`<Project>.rpy`) is a unit that can be checked into a CM archive, which means you can perform CM operations on this file just like any other unit. Because the `.rpy` file contains the latest list of top-level units in your project, you will probably need to check it out for changes whenever you plan to add new packages or components to the project. (Adding new elements to a package requires a checkout of the package file, not the `.rpy` file.) In addition, you need to check out the `.rpy` file if you plan on modifying the project-level properties.

Because it is a unit that can be placed under CM, you can use the DiffMerge tool on the `.rpy` file. For more information about this tool, see [Parallel development](#).

The project file is a unique, top-level package.

Multiple Rational Rhapsody projects

As the project expands and complexity grows, you have several options for “growing the project.” In some cases, the best approach is to continue adding new elements into a single Rational Rhapsody project. In other cases, it is more efficient to create a new project.

No single solution fits all scenarios. Consider the following issues when planning for project growth and expansion:

- ◆ For a single Rational Rhapsody project, it is easy to apply a property or set of properties to all model elements. For multiple projects, the process must be done for each project.
- ◆ The CM archive associated with a project is stored in a project-level property. Therefore, if a project will be stored in more than one archive, you must create a separate Rational Rhapsody project for each archive.
- ◆ When you want to achieve a situation where all team members are aware of all the design parts, including those they are not directly involved with, having a single Rational Rhapsody project is a good approach.

- ◆ If you want to isolate the work of different team members so each member sees only the elements relating to their work, create multiple projects (one for each team member).
- ◆ Distributed teams working on different components of a system can benefit from splitting the overall project into several smaller Rational Rhapsody projects.
- ◆ If you want to reduce the complexity of a project by limiting it to a defined and encapsulated functionality, having multiple projects (each dedicated to a well-defined piece of functionality, essentially a set of components) means that all team members have access to all the elements in the project.
- ◆ When your project is practicing in a binary reuse pattern, split it into several projects. When using a model reuse pattern, do not split it.
- ◆ When the project is mapped to several binary components interleaved together to create a final product (for example, a set of libraries used by one or more executables), placing them in a single project helps the team member designing not only the internals for the component but also the relations between them using component diagrams (makefiles take into account cross-component relations).

The following issues should not affect the decision to split a project into multiple projects:

- ◆ **Time to load the project into Rational Rhapsody**

The partial load feature enables you to load only the units needed for your current task without loading the entire project.

- ◆ **The operating system**

If a single design needs to be regenerated to target several operating systems, there is no need to create multiple projects.

The decision of how to structure the overall project is not irreversible; you can easily divide a project into two projects, and you can merge two projects into a single project.

Dividing a project into two projects

To divide a Rational Rhapsody project into two projects:

1. Decide on names for the two new projects.
Ideally, a project that you are planning to divide has two separate, complex parts, with no dependencies between them.
2. Save the project under both of the new names, for example, `part1.rpy`, and `part2.rpy`.
3. For each new project, open the project in Rational Rhapsody, delete the unnecessary elements from the model, and save.

You now have two separate projects stored on your file system. You cannot modify the CM archive from within Rational Rhapsody (it must be done manually using your CM tool).

Two projects into one project

Combining two projects into one is more complex than dividing a project. When combining two projects, make sure that the project-level properties of the two projects do not collide with each other, causing problems in the composite project.

You can use the DiffMerge tool to accomplish this task. For more information about this tool, see [Parallel development](#).

Changes to the CM system must be done outside of Rational Rhapsody.

Multiple project workflow

When you have a project that consists of several Rational Rhapsody projects and you want to share information between them beyond binary reuse, while keeping the design parts separate so you cannot change one project while working on another, you can add units from one project as references in another. This is a classic, multiple-parts gray box collaboration.

Only non-referenced units (units loaded in the current model) should be archived with the project. When new units are added to a referenced project, they must be refreshed in the active project.

Repository structure

By default, Rational Rhapsody stores all the repository files (class files, package files, diagram files, and so on) in a single directory, creating a “flat” list of files that represents a tree-structured hierarchy of the UML design elements in Rational Rhapsody. This approach has certain limitations, especially when projects grow very large and complex.

Rational Rhapsody supports a paradigm known as *hierarchical repository*. With this feature, as you save your model, Rational Rhapsody maps UML packages to directories containing the design elements included in their scope. You are not required to use the hierarchical scheme; by default, the tool uses the flat structure. You have the choice of moving some or all of your packages to a hierarchical structure.

Restructuring the project can be done as a one-time effort, or by gradually moving several packages at a time to their own directories, reducing complexities in specific areas of the model.

Repository structure planning

The structure of your repository depends on the organization of your project and, like other project decisions, involves trade-offs.

On one hand, keeping an entire model in a single file prevents any conflict but practically disables parallel development. On the other hand, making every single design element and diagram a unit can be overkill, and leads to a large number of files and much larger overhead in terms of the number of files you need to check out in order to accomplish a task.

Analogously, when structuring a Rational Rhapsody project repository, keep in mind that moving from one extreme (where you have hundreds of Rational Rhapsody files located in a single directory) to the other extreme (where you have hundreds of directories containing a single file each) is probably not an effective use of the hierarchical repository feature.

For example, consider a project **A** that contains 200 units, out of which 120 are package (*.sbs) files and 80 are other types of unit files, and project **B** with 200 units, out of which 20 are packages.

If moved to a pure tree structure (where every UML package is mapped to a directory in the file system), each directory in project **A** will have (on average) 1.66 files; many directories will contain a single file and many others will have two files. Such a structure provides very little advantage, and might increase project overhead.

On the other hand, if you moved project **B** to a pure tree structure, an average directory would contain 10 files with interrelated functionality, which is a reasonable number. This solves many of the problems faced by flat repositories, while adding very little overhead in terms of new directories.

Consider these factors when you decide whether to store a project in a hierarchical structure. Note that the entire project does not have to have the same structure, some packages can be stored in a hierarchy, whereas others remain in a flat structure in their parent folder.

Flat repositories

A flat repository works best for smaller projects that have many packages that contain very few or no unit files and subpackages. In these cases, dividing packages into folders does not simplify the project, and might make it more complex.

In contrast, large models typically encounter problems when stored in flat repositories, especially when the model contains many units and subpackages.

The following list describes typical problems encountered with flat repositories:

- ◆ **Visibility** where a single directory containing hundreds of files cannot be viewed in a clear way in a single window.
- ◆ **Focus** where in a directory containing hundreds of files, you cannot focus on a single functional area of the project and detect which elements belong to which functional area. This is important when you want to check if all the files are present in your view, find the latest modification date, and so on.
- ◆ **Portability** where copying and sending several Rational Rhapsody files representing a single functional area to a co-worker or support person poses a serious challenge because of the difficulty in determining which files are really needed.
- ◆ **Branching** where when checking out several different parts of the model to a branch, it is difficult to identify the required functionality and check in all related files.
- ◆ **Integration with other tools and processes** where some CM tools assist with process and control issues by setting permissions on directories. For example, you can lock a directory called `engine` or `core` to prevent modifications, but keep the `application` directory open to changes. A flat repository cannot take advantage of these options, but a hierarchical repository easily facilitates the implementation of a directory-driven process.

Hierarchical repositories

By dividing a project into hierarchical directories based on model organization, you can avoid or resolve many of the problems encountered with flat repositories.

With a hierarchical repository, you can easily identify the correct directory for a particular functional area. You can then view all related files in a single window, copy them or extract them for distribution, check them into or out of an archive, and take advantage of CM features that operate on directories.

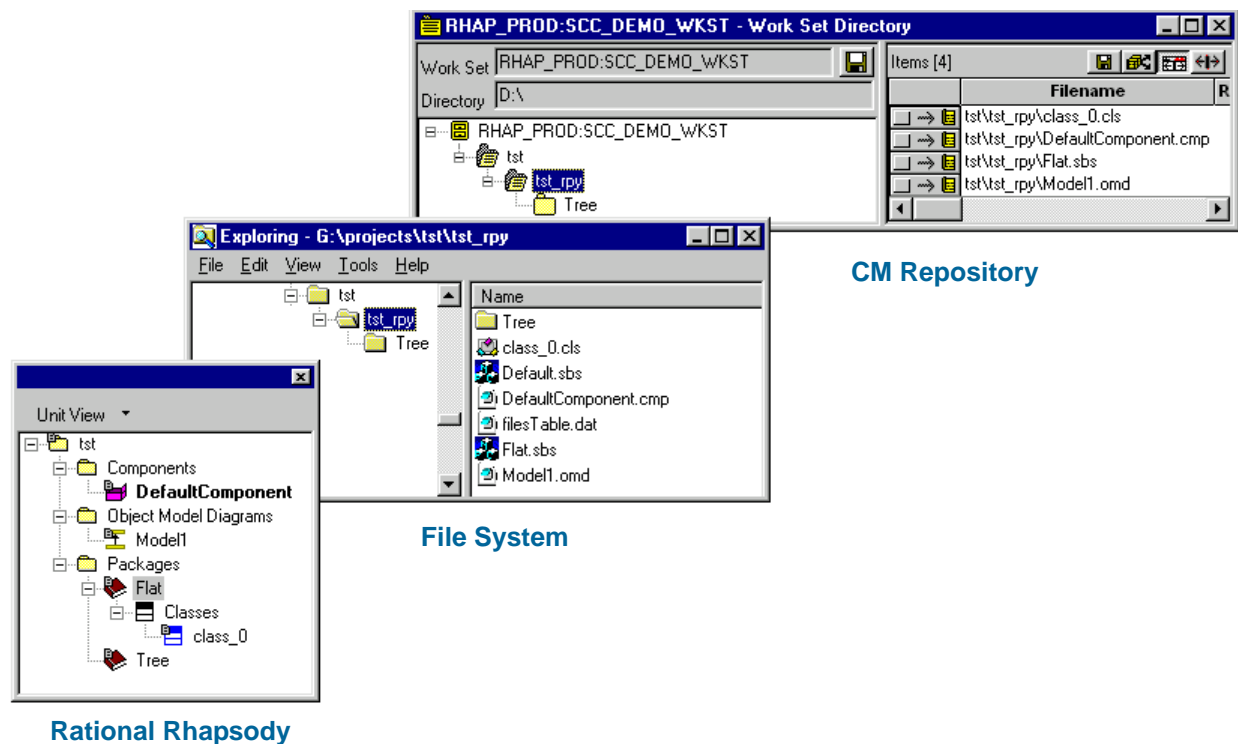
However, maintaining directories adds a level of complexity to file management. Operations such as renaming or moving a package can be more difficult to execute within the CM tool, depending on the type of tool you are using. Careful analysis of the model will determine whether the added organization of a hierarchical repository is worth the trade-off of increased complexity.

The repository structure is controlled by the `General::Model::DefaultDirectoryScheme` property.

Example of project under CM

A project consists of Rational Rhapsody elements stored as files on the local file system and archived in a CM repository.

Ideally, both the CM repository and file system mirror the structure of the project in Rational Rhapsody. The following figure shows a Rational Rhapsody design, file structure, and CM repository synchronized with each other.



In some cases, changes you make to your Rational Rhapsody model result in changes to the directories and files stored on the file system. Some examples include creating new units, renaming units, and moving units to new locations. Rational Rhapsody can update some of these changes in the CM system automatically, completing the three-way synchronization without any additional steps. Use the CM tool to manually make the changes that cannot be accomplished using the CM tool from within Rational Rhapsody.

In other cases, it is easier to make changes to the structure of your project outside of Rational Rhapsody, particularly when moving files from a flat structure to a hierarchical structure. You can make these changes to your file system and your CM system directly. When you next open Rational Rhapsody, you are asked to locate the missing units. Rational Rhapsody prompts you with two options for restoring synchronization:

- ◆ Update the repository with the new location, so the new structure is preserved going forward.
- ◆ Move the files from their current location back into the expected location.

Note

If the CM system has been updated with the new location of the file, it is important that you *not* select the Copy unit to project path option. This might affect the integrity of the system. Instead, update your Rational Rhapsody model with the current location of the file so it matches the CM system.

Restructuring a project

To make changes to your file system and CM system outside of Rational Rhapsody:

1. Update the file structure on your file system.
2. Update the structure of your CM system outside of Rational Rhapsody using your CM tool.
3. Open Rational Rhapsody.
4. In the Search for File window, locate the new location of the file.
5. Select **Update model—Keep unit in current location** from the Next Location of Missing Unit window.

File and directory creation

When you create a new unit in a model, Rational Rhapsody creates a new file in the file system for that unit. This new file is not automatically placed under CM. Use the Add to Archive operation to add the file to the CM tool.

When creating a new package unit, if the project uses a hierarchical directory structure, both a file and a directory are created in the Rational Rhapsody project. You need to create this new directory in the CM file structure.

Directories in SCC mode

To add a new package and package directory to a CM archive in Microsoft® Common Source Code Control (SCC) mode, add the unit file to the repository using the Add to Archive operation. The Add to Archive operation creates directories automatically in the CM system, as necessary.

Directories in batch mode

When you use a CM tool in batch mode, directories are not automatically created in the CM archive. However, Rational Rhapsody assists you with creating new directories using the `ConfigurationManagement::<CM tool>::MakeCMShadowDirActivation` property.

The possible values are as follows:

- ◆ `Disable` means Rational Rhapsody does not create a directory. You need to create the directory in the CM archive outside of Rational Rhapsody.
- ◆ `UserConfirmation` means Rational Rhapsody asks if you want to create a new directory in the CM structure whenever you save a project that contains a new directory.
- ◆ `Automatic` means Rational Rhapsody automatically creates the directory in the CM structure whenever new directories are created by a save in Rational Rhapsody. Note that Rational Rhapsody creates the directory in the CM archive for every new directory. There exists the potential for the creation of unneeded (and probably unwanted) directories for packages that you do not intend to add to the archive.

Once you have created the new directory in the CM archive, you can archive the new package unit and any of its child unit files using the Add to Archive operation.

Keyword expansion in batch mode

When performing CM operations in batch mode, Rational Rhapsody expands keywords.

The following table lists the keywords and their expansion:

Administrative Keywords		
Keyword Name	Expanded to	Comments
\$Operation	The name of the CM operation being executed by Rational Rhapsody	Appears in CMOperationStartSeparator & CMOperationEndSeparator
\$Time	The current system time	
\$Date	The current system date	
\$User	The logged in user name	
\$temp	The current system temporary directory as specified in the TEMP environment variable	
\$OMROOT	The value of OMROOT as loaded into Rational Rhapsody (most likely from the .ini file)	
\$projectname	The name of the Rational Rhapsody project	
Directories, paths, file names		
Keyword Name	Expanded to	Comments
\$projectunitdirname	Project unit directory name (_rpy). This is not the full path to the unit directory, but just the name of the directory.	This keyword is different from the \$rhpdirectory, which expands to full path to the project unit directory.
\$FileName	This expands to repository file specified during connect to archive operation.	
\$fulldir	Name of the CM shadow directory to be created in the archive	To be used in \$MakeCMShadowDir property.
\$parentdir	The name of the parent directory of the CM shadow directory to be created in the archive	
\$rhpdirectory	Name of the project units directory (_rpy)	

Administrative Keywords (Continued)		
Directories, paths, file names (continued)		
Keyword Name	Expanded to	Comments
\$SubDirs	This is the sub directory of a CM unit with respect to its archive root director.	
\$targetDir	Name of the directory where the file is put during Fetch operation	
\$unit	Name of the CM unit file name	
\$UnitDirectory	This is the sub-directory of a CM unit with respect to project units directory (_rpy).	This is the "PersistsAs" sub-directory of the CM unit.
\$UnitDirPath	Full directory name of the CM unit	
\$UnitPath	Complete path name of the CM unit file name	
\$currentdirectory	Current working directory name	
\$archivedirectory	Directory name of the CM archive	
\$ArchivePath	Directory name of the CM archive	
\$dir	These is the name of the directory which contains the CM units whose name is changed.	To be used in "Rename," "RenameDirectory," "Move," "MoveDirectory" property.
\$oldname	Previous name of the CM unit file name it can be also a directory name	
\$newname	Changed name of the CM unit file name it can be also a directory name	
\$olddir	Name of the directory from which CM unit has been moved	To be used in "Move" & "MoveDirectory" property
\$newdir	Name of the directory where CM unit is moved	
\$archive	Name of the archive file specified in "Connect2Archive" operation	
\$ArchiveRoot	NOT USED	NOT USED

Administrative Keywords (Continued)		
Arguments to CM commands		
Keyword Name	Expanded to	Comments
\$mode	Locked or unlocked check box	Reflects value entered by the user in the UI
\$log	Comment string	
\$label	Label identifier string	
\$newdir	The new directory in which a unit will be stored	Used when moving a unit in tree structure
\$olddir	The old directory name where unit used to be stored	
\$newName	The new name for a unit file	Used when renaming a unit
\$oldName	The old name for a unit file	
\$dir	This is the name of the directory, which contains the CM units whose names changed.	To be used in "Rename," "RenameDirectory," "Move," "MoveDirectory" property
\$oldname	Previous name of the CM unit file name it can be also a directory name	
\$newname	Changed name of the CM unit file name it can be also a directory name	
\$olddir	Name of the directory from which CM unit has been moved	To be used in "Move" & "MoveDirectory" property
\$newdir	Name of the directory where CM unit is moved	
Keywords for the Rational Rhapsody DiffMerge Tool		
Keyword Name	Expanded to	Comments
\$DiffInvocation	The command line used to launch the Rational Rhapsody DiffMerge tool	
\$source1	The full path name of the first file to be compared	
\$source2	The full path name of the second file to be compared	
\$output	The name of the file used by the merge tool	

File and directory deletion

When you delete a unit from a project in flat mode (where all units are in one directory), Rational Rhapsody performs a Delete from CM operation. This process uses properties in batch mode and the SCC API in SCC mode.

In SCC mode, when you delete a package that is stored in a separate directory, only the package file is deleted; in Rational ClearCase, both the package file and the package directory are removed from the CM repository.

The following properties control file deletion (including files of descendant units):

- ◆ `ConfigurationManagement::SCC::DeleteActivation` (SCC mode)
- ◆ `ConfigurationManagement::<CM tool>` (batch mode)

Note

For this operation, the Rational Rhapsody Undo command (**Ctrl+Z**) works only in the model, not in the archive. You cannot undo a delete operation in the archive.

In some cases, Rational Rhapsody cannot delete a file from the CM archive. When this occurs, Rational Rhapsody removes the unit from the model (but does not delete the file on the hard drive or remove the file from the CM system) and displays an error message informing you that the unit was not completely removed. Check the Rational Rhapsody Output window for additional messages.

Unit storage

By definition, units are stored in separate files. However, you can move a unit back into the file of its parent (usually a package) using the Edit Unit window. When a unit is moved back into the parent file, the unit file becomes obsolete. Therefore, Rational Rhapsody deletes the unit file (without descendants) using the Delete from CM command, as described in [File and directory deletion](#).

Packages in a new directory

When a package is saved in its own directory, a new directory is created in its parent folder. When a package is moved out of its own directory back into the parent folder, a directory is removed from the project. However, when either of these operations occur, Rational Rhapsody does not make any changes to the CM system. You must manually add or remove the extra directory in the CM system.

In these cases, Rational Rhapsody displays an error message stating that the operation could not be performed in the CM system. For more information, see [When Rational Rhapsody cannot update the CM system](#).

File renaming

When you rename a unit in your model, Rational Rhapsody uses the Rename in CM operation. This operation is based on properties in batch mode and the SCC API in SCC mode.

The following properties control renaming a file:

- ◆ `ConfigurationManagement::SCC::RenameActivation` (SCC mode)
- ◆ `ConfigurationManagement::ClearCase::RenameActivation` and `Rename` (ClearCase)

Note

For this operation, the Rational Rhapsody Undo command (**Ctrl+Z**) works only in the model, not in the archive. You cannot undo a rename operation in the archive.

In cases where Rational Rhapsody cannot rename the file in the CM system, Rational Rhapsody changes the unit name in the model, but not in the CM system. For more information, see [When Rational Rhapsody cannot update the CM system](#).

Package contained in its own directory renaming

In Rational ClearCase, when you rename a package in its own directory, both the file name and the directory name are changed (see [File renaming](#)). Note that the `ConfigurationManagement::ClearCase::RenameDirectory` property controls renaming directories.

For other CM tools, when you rename a package located in a separate directory, Rational Rhapsody does not make any changes to the CM system. Instead, Rational Rhapsody displays an error message stating that the operation could not be performed in the CM system.

If you change the model only, Rational Rhapsody renames the package in the model but *does not* change the file or directory name in either the file system or CM system. If you change the model and the file system, Rational Rhapsody changes the names of the model element, file, and directory. For more information, see [When Rational Rhapsody cannot update the CM system](#).

Control moving a file or directory

When you move a file from one location to another in the file system, Rational Rhapsody activates the Move in CM operation. This operation is based on properties in batch mode and the SCC API in SCC mode.

The following properties control moving a file:

- ◆ `ConfigurationManagement::SCC::MoveActivation` (SCC mode)
- ◆ `ConfigurationManagement::ClearCase::MoveActivation` (batch mode)
- ◆ `ConfigurationManagement::ClearCase::MoveDirectory` (ClearCase)

Note

For this operation, the Rational Rhapsody Undo command (**Ctrl+Z**) works only in the model, not in the archive. You cannot undo a move operation in the archive.

When you rename a package that is in a separate directory, Rational Rhapsody does not make any changes to the CM system. Instead, Rational Rhapsody displays an error message stating that the operation could not be performed in the CM system.

If you change the model only, Rational Rhapsody moves the package in the model but *does not* change the file system or CM system. If you change the model and the file system, Rational Rhapsody changes both the model and file system. For more information, see [When Rational Rhapsody cannot update the CM system](#).

When Rational Rhapsody cannot update the CM system

In some cases, Rational Rhapsody cannot update the CM system to match the changes made to the file system. If this occurs, Rational Rhapsody displays a message window.

You have the following options:

- ◆ Change the model structure, but keep the physical layout on the hard drive as-is. Rational Rhapsody does not make any modifications to the CM system. For more information, see [Model only changes](#).
- ◆ Change the model structure and hard drive layout. This requires modifications to the CM system outside of Rational Rhapsody. For more information, see [Model and the file system changes](#).
- ◆ Cancel the operation. For more information about this tool, see [Canceling a change](#).

To set your current selection as the default (and not display this window in the future), select the **Don't ask me again, use my current selection as default** check box.

The `ConfigurationManagement::General::CMConflictResolution` property stores the default selection for this error message. Set the property to `AskUser` if you want this error message to appear whenever this situation occurs.

Model only changes

Select the **Change only model** option to maintain synchronization between the model and CM system (units do not lose their “CM history,” you do not encounter unrecognized units that are already under CM, and so on).

When you select this option, you do not need to perform any maintenance operations on the CM tool outside of Rational Rhapsody and workflow is not interrupted. However, the model structure does not match the physical layout of the files on the hard drive. This is the simplest solution, but can lead to problems in later development, especially for large projects. The disparity between the model layout and the file system can be confusing.

To set this option as default, set the property as follows:

```
ConfigurationManagement::General::CMConflictResolution to ModelOnly
```


Model and the file system changes

If you update both the file system and model, a model/CM synchronization problem might result (some of the units might lose their “CM history,” some units cannot be checked in because they are not recognized as CM elements, and so on). In addition, you must perform maintenance operations on the CM tool outside of Rational Rhapsody to synchronize the changes on the hard drive with the CM system.

The benefit of this option is that the model structure, as seen in the Rational Rhapsody browser, matches the physical layout on the hard drive. Therefore, this option requires more work to implement, but results in a better organized system that is less likely to cause problems in the future.

To set this option as default, set the property as follows:

```
ConfigurationManagement::General::CMConflictResolution to ModelAndFileSystem
```

Canceling a change

Click **Cancel** on the error message box to cancel the changes that caused the CM conflict to occur.

CM and Rational Rhapsody

Rational Rhapsody supports collaboration among several developers or teams by allowing projects to be divided into multiple files, called *units*, that can be worked on concurrently. It also has a built-in interface that connects with several common source control or configuration management (CM) tools.

This subject describes how to manage units from within Rational Rhapsody using a CM tool.

SCC versus Batch mode

Rational Rhapsody supports common CM operations, such as Connect to Archive, Add Member, Check In, and Check Out, for a wide variety of CM tools. The Windows version of Rational Rhapsody also supports SCC operations, such as Get, Un-Check Out, and History, for CM tools that conform to the SCC standard.

Batch mode is the traditional method of interacting with CM tools that do not conform to the SCC standard. In this mode, Rational Rhapsody has a custom set of properties for each tool that calls tool-specific commands for the CM operations. The Lock and Unlock operations are supported only in batch mode.

SCC mode is an alternate method of interacting with CM tools that conform to the SCC standard. In SCC mode, you need set only one property to interface with any of dozens of SCC-compliant CM tools, without further customization. You interact directly with the GUI elements of your CM tool to perform SCC-supported operations. Return status information, or error information in the case of failure, comes directly from the CM tool. Thus, you have more direct CM tool interaction (and receive more complete feedback on CM operations) in SCC mode. The Fetch and Properties operations are supported in SCC mode only.

Note

As an SCC-compliant IDE, Rational Rhapsody can communicate with any CM tool that conform to the SCC standard. Note, however, that IBM Rational Synergy, IBM Rational ClearCase, and Serena PVCS Dimensions are the only SCC tools supported by Rational Rhapsody.

Configuration Items window

To begin any CM operation in Rational Rhapsody, open the Configuration Items window. Choose **File > Configuration Items**.

Note that the available operations depend on the CM tool you are using and whether you are running in batch or SCC mode. See [CM operations](#) for the list of CM operations supported in either mode.

The buttons in the Configuration Items window apply commands directly to the CM tool. The response of the specific tool is displayed in the Rational Rhapsody Output window or, in the case of SCC tools, in message boxes specially designed for this purpose.

Note

It is important to observe messages from the CM tool because the commands can sometimes fail. See [CM Output window](#).

CM operations







There are a number of CM operations included in the standard Rational Rhapsody interface. You can also assign additional operations to any of four customizable buttons. For more information, see [CM interface extension](#).










Connect to Archive

The Connect to Archive operation connects the project in your workspace to a CM archive. In addition, it permanently sets properties associated with the archive to be referenced internally whenever the project needs to communicate with the CM tool. You need to perform this operation only once for the lifetime of the project.

A Connect operation also triggers actions that should be performed before any other CM operation.

In batch mode, you must tell Rational Rhapsody which CM tool you are using by setting the `CMTTool` property. In SCC mode, you need to set a different property because the `CMTTool` property is ignored. For more information, see [Enabling a SCC-compliant CM tool](#).

Button	CM Operation	Mode	Description
	Connect to Archive	Batch and SCC	Connects the project to an archive. You need to perform this task only once for the life of the project.
	Show Items in Archive	Batch and SCC	Displays units that have been archived. (The Configuration Items list displays all units, archived or not.)
	Comparing with the DiffMerge tool	Batch and SCC	Compares a unit in the model with its archived unit in the configuration management (CM) archive.
	Synchronize Items	Batch and SCC	In Rational ClearCase, this operation synchronizes your model with the current view. In batch mode, this operation synchronizes the model with the latest versions in the CM tool. In SCC mode, this operation synchronizes the model with your local file system.
	Run CM tool	Batch and SCC	Launches the CM tool assigned to this project. It is controlled by the <code>RunCMTToolCommand</code> property.
	Check Out Branch	Batch and SCC	Checks out all configuration items in a branch.

Button	CM Operation	Mode	Description
	Checking out a unit	Batch and SCC	Checks out the file from the archive. The model element becomes a read/write (RW) unit that can be edited.
	Checking in a unit	Batch and SCC	Checks in new versions of configuration items. The model element becomes read-only (RO).
	Using Add to Archive in CM operations	Batch and SCC	Adds new units to the archive.
	Locking/Unlocking a unit	Batch (except Rational ClearCase)	Locks units to prevent them from being modified by others.
	Locking/Unlocking a unit	Batch (except Rational ClearCase)	Unlocks units that you have previously locked so others can edit them.
	Fetching a unit	SCC	Loads the checked out unit into your model.
	History/Version tree	SCC and Rational ClearCase	In SCC mode, this operation displays the history of the file in the archive. For Rational ClearCase in batch mode, this operation lists all the versions in the archive.
	Displaying the properties of a unit	SCC	Enables you to set the properties of the file in the CM tool.
	Uncheck Out	SCC and Rational ClearCase	Rolls the file back to the latest version in the archive. The file is unlocked in the archive, and the local copy is replaced by the archived version, as if the file had never been checked out.

Connect to Archive in SCC Mode

In SCC mode, you must connect to the archive only once (there is no need to reconnect the project). Once the project is connected to the archive, you are prompted to log into the CM system when you perform a CM operation.

For more information, see [Creating the initial connection to the SCC tool in Dimensions](#).

Connecting to a different archive

To connect to a different archive, you must first clear the property information from the currently displayed archive.

To perform this clean-up operation:

1. Choose **File > Project Properties**.
2. In the Features window, on the **Properties** tab ensure that the View All option is selected.
3. Navigate to the `ConfigurationManagement::SCC` group of properties.
4. Clear any information in the **AuxProjPath** box.
5. Clear any information in the **ProjName** box.
6. Click **OK**.

Configuring a CM tool Batch mode

To configure a CM tool in batch mode:

1. Choose **File > Project Properties**.
2. On the **Properties** tab, set the `ConfigurationManagement::General::CMTool` property to one of the following values:
 - ◆ `ClearCase` (In batch mode only.)
 - ◆ `None` (You are not using CM.)

Connecting a project to the archive

To connect the project to the archive:

1. Open the Configuration Items window. Choose **File > Configuration Items**.
2. Click the Connect to Archive button. Note that in batch mode, the Connect to Archive window opens. When you use the **Browse** button, if the archive is represented as a directory (as in RCS) rather than a file, you must select a file within this directory and edit the path in the window. Alternatively, type the entire path in the text box.
3. Once you have set the archive, click **OK**.

Note

Setting the archive might involve additional activities, depending on how your CM tool has been set up.

The CM tool connects your working project to the archive. If successful, the Output window displays a confirmation message.

Show Items in Archive

The Show items in Archive operation lists units that have been added and checked into the CM archive. You must perform a List Archive before checking out a unit that does not currently exist in your workspace.

To list the archive, click **Show items in Archive** in the Configuration Items window. The Archive window opens, listing all the units in the archive.

For an example of using the List Archive operation in SCC mode, see [Listing the archive in PVCS Dimensions](#).

Run CM tool

You can use this user-defined button to launch your CM application from the Rational Rhapsody interface. It is controlled by the `RunCMToolCommand` property. For more information, see [CM interface extension](#).

Comparing with the DiffMerge tool

The DiffMerge tool in Rational Rhapsody allows you to compare two units (or two versions of the same unit) and merge them, if you want.

To compare the two units of a model:

1. In the Configuration Items window or the Archive list, select the unit you want to compare.
2. Click **Diff with Rhapsody**.

You can launch the DiffMerge tool to compare two archived files with the same name from inside Rational Rhapsody. If you want to compare units of the same type, but with different names or entire Rational Rhapsody projects, you must launch the DiffMerge tool outside Rational Rhapsody. For detailed instructions to use this tool, see [Parallel development](#).

Displaying the properties of a unit

The Properties operation is an SCC operation that retrieves the file details for a unit (such as the file name and date it was created). The Properties operation is not available in batch mode.

To display the details of a unit that is a member of an archive:

1. Open the Configuration Items window. Choose **File > Configuration Items**.
2. Highlight a unit that has been added to the archive.
3. Open its Properties window. Click the Properties button.
4. Examine the file details for the highlighted unit.

Synchronize Items

Depending on the type of CM tool, the Synchronize option enables you to synchronize your model (workspace) with the local file system or CM archive.

To synchronize a unit in your workspace with the archive, click **Synchronize Items**. Rational Rhapsody displays one of the following windows:

- ◆ **Synchronize with View.** When you use Rational ClearCase in batch mode, the Synchronize with View window synchronizes your model with the current Rational ClearCase view.
- ◆ **Model files have been modified outside of Rhapsody.** In SCC mode, the synchronize operation synchronizes elements in your local model (workspace) with your local file system.

Note

The Synchronize window does not display controlled files when there are newer versions of these files in the CM system.

Autosynchronize

The `AutoSynchronize` property is a Boolean value that determines whether Rational Rhapsody does synchronization. When this property is `Checked`, each time Rational Rhapsody gets the focus (for example, if you leave Rational Rhapsody to read e-mail, then switch back to Rational Rhapsody), Rational Rhapsody calls the synchronize functionality. The synchronize can be a synchronization with the files on the file system, view, or CM archive, depending on the environment.

To set this property, set the `General::Model::AutoSynchronize` value to `Checked`. For more information about this property, see the **Properties** tab of the Features window for it.

Check out operation

In batch mode, the Check Out operation can fetch a unit from the archive with or without a lock. However, in SCC mode, the Check Out operation always fetches the unit with a lock.

In both batch and SCC modes, the Check Out operation always performs implicit Add to Model and Update operations using the appropriate CM information (such as version, CM header, and so on).

Before checking a unit out of the archive, use the List Archive option to confirm that the unit has been properly added to the archive, and that at least one version of it has already been checked in.

Checking out a unit

To check out a unit from the archive:

1. In the Rational Rhapsody browser, right-click the unit or units you want to check out and select **Configuration Management > Check Out**. The Check Out window opens with information displayed for the selected items. If a selected item is reserved by another user, the **Reserved** check box is selected and the check out operation cannot be performed.
2. Select the **Include descendents** check box if you want the units checked out with nested units. For example, if a package that is a unit has a nested package that is also a unit, this option checks out both packages.
3. Select the **Include corresponding source artifacts** check box, if available, if you want to check out corresponding source artifacts. (Code respect information, such as mapping, ordering, and code snippets, of an element is defined in a *SourceArtifact* element, which is typically created by reverse engineering or roundtripping.)
4. If your CM tool has advanced options, you can click an **Advanced** button to open its Advanced Options window. The Advanced Options window that opens is provided by your CM tool.
5. If you opened an Advanced Options window, click **OK** to close it after you make your selections.
6. Click **OK** on the Check Out Options window to confirm your settings and check out the units.

Depending on what changes you make to a unit that is contained inside a package (for example, a class), the package information might also change accordingly.

In the following situations, you must check out the containing package and the nested unit:

- ◆ You change the package to which the unit belongs.
- ◆ You move a nested unit from one unit to another within the package.
- ◆ You change the name of a unit.

Check In operation

The Check In operation copies a unit from your working project into the CM archive.

Checking in a unit

To check a unit into the archive:

1. In the Rational Rhapsody browser, right-click the unit or units you want to check into the configuration management system.
2. Select **Configuration Management > Check In**. The Check In window displays with information displayed for the selected items. If a selected item is locked by another user, the **Locked** check box is selected and the check in operation cannot be performed.
3. Select the **Include descendents** check box if you want to check in nested units.
4. Select the **Include corresponding source artifacts** check box, if available, if you want to check in corresponding source artifacts. (Code respect information, such as mapping, ordering, and code snippets, of an element is defined in a *SourceArtifact* element, which is typically created by reverse engineering or roundtripping.)
5. Type a **Revision Description** explaining the changes in this revision. Note that most CM tools ignore the description the first time you check in a unit.
6. If your CM tool has advanced options, you can click the **Advanced** button to open its Advanced Options window; otherwise the button is disabled. The Advanced Options window that opens is provided by your CM tool.
7. If you opened an Advanced Options window, click **OK** to close it after you make your selections.
8. Click **OK** on the Check In Options window to confirm and check in the units. Rational Rhapsody saves the project (including concatenating CM headers and footers) before the CM tool checks the unit into the archive.

9. Click **OK** to dismiss the confirmation message.

Note

In SCC mode, the Check In operation can be successfully completed only if a unit is already checked out. If a unit that you are trying to check in is not already checked out, by default, the Check In operation is ignored. A message to this effect is displayed on the **Configuration Management** tab of the Output window.

Using Add to Archive in CM operations

You can add a unit into a CM archive only if the unit is not already in it.

To add one or more units to a CM archive:

1. In the Configuration Items window, select the units you want to add to the archive.
2. Click the Add to Archive button.
3. If your CM tool has advanced options, you can click the **Advanced** button to open its Advanced Options window; otherwise the button is disabled. The Advanced Options window that opens is provided by your CM tool.
4. If you opened an Advanced Options window, click **OK** to close it after you make your selections.
5. Click **OK** on the Add to Archive Options window.

Rational Rhapsody saves the project and the CM tool adds the units to the archive. If the operation is successful, a confirmation message is displayed. After units are added to the archive, the CM tool rereads information stored with the unit files, begins load metering, and processes the information rapidly.

Lock and Unlock operations

The Lock and Unlock operations are available in batch mode only. When you lock a unit, its permission becomes read/write for you and read-only for others. When you unlock a unit, its permission becomes read-only for all users.

Locking/Unlocking a unit

To lock or unlock one or more units:

1. In the Configuration Items window, select the currently unlocked (RO) or locked (RW) units.
2. Click **Lock** or **Unlock**, whichever is applicable.
3. Click **OK**.
The permission for the unit is changed to read/write (RW) if you locked or to read-only (RO) if you unlocked.

Fetching a unit

The Fetch operation is an SCC operation that fetches a unit from the archive without a lock (RO). The alternative is the SCC Check Out operation, which always fetches a unit with a lock (RW). The Fetch operation is not available in batch mode.

To fetch a unit:

1. In the Configuration Items window, select the units that you want to check out as unlocked.
2. Click the Fetch button.
3. If your CM tool has advanced options, you can click the **Advanced** button to open its Advanced Options window; otherwise the button is disabled. The Advanced Options window that opens is provided by your CM tool.
4. If you opened an Advanced Options window, click **OK** to close it after you make your selections.
5. Click **OK** on the Get Options window.

Using Uncheckout in CM operations

The Undo Check Out Options is an SCC operation that reverses the effect of a Check Out, releasing the lock on a unit (making it RO), and reverting to the file version before the last Check Out operation. The Undo Check Out operation is not available in batch mode (except with Rational ClearCase).

To undo a check out:

1. In the Configuration Items window, select a unit that has been checked out (whose mode is RW).
2. Click the Uncheckout button.
3. If your CM tool has advanced options, you can click the **Advanced** button to open its Advanced Options window; otherwise the button is disabled. The Advanced Options window that opens is provided by your CM tool.
4. If you opened an Advanced Options window, click **OK** to close it after you make your selections.
5. Click **OK** on the Undo Check Out Options window.

History/Version tree

The History operation is an SCC operation that opens an archive for a unit so you can review the history for the unit and access previously archived revisions if you want. The History operation is not available in batch mode.

To view the history of a unit, click **History** in the Configuration Items window.

If you are using Rational ClearCase in batch mode, use this button to view the version tree. The version tree lists all versions in the Rational ClearCase archive.

CM status of units in a project






If you are using an SCC-compliant configuration management tool in conjunction with Rational Rhapsody, Rational Rhapsody can keep track of and display the CM status of all units in the Rational Rhapsody project. This information is displayed in both the Rational Rhapsody browser and the Configuration Items window.

Note

This feature is available only for CM tools that implement the SCC API.

CM status information in the browser

In the browser, Rational Rhapsody displays an icon that represents the CM status for the unit along with the icon for the element. The CM status icons reflect the possible statuses:

-  means the unit is not in source control
-  means the unit is checked in
-  means the unit is checked out by the user
-  means the unit is checked out by another user
-  means the unit is deleted from CM

The status displayed in the browser is updated when:


- ◆ you connect to the CM archive
- ◆ a CM operation is completed (status is updated only for the items affected by the operation)
- ◆ you select **Configuration Management > Refresh State**

The CM status of project units will be updated from the CM repository when a project is opened if the `ConfigurationManagement::SCC::RefreshCMStatusAtProjectOpenup` property is set to `Yes`. This property can also be set to `No` and `Ask User`.

CM status information in the Configuration Items window

The Configuration Items window includes two columns that are used to reflect the CM status of a unit:

- ◆ **Controlled by CM** can show Yes, No, or Deleted
- ◆ **Checked Out** can show Yes, No, Out by other user

You can click the Refresh Status button  to refresh the CM status displayed for the items listed in the window.

Property to turn off display of CM status

If you do not want Rational Rhapsody to display the CM status of project units, set the `ConfigurationManagement::SCC::ShowCMStatus` property to `Cleared`. This property is set at the project level. When the value of this property is set to `Cleared`:

- ◆ Statuses are not displayed in the browser or Configuration Items window.
- ◆ The Refresh Status button is not displayed in the Configuration Items window.

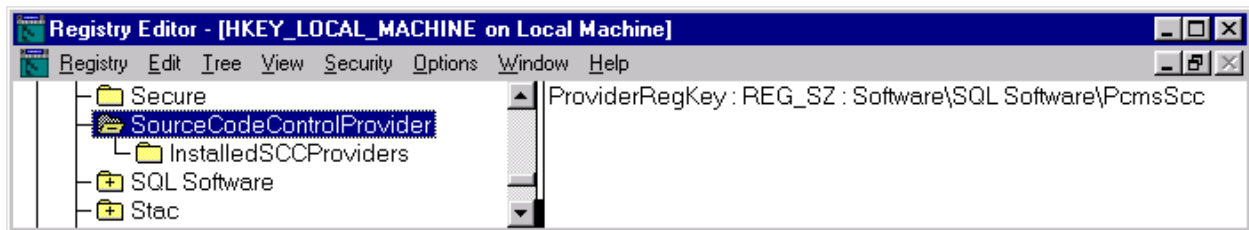
The above items will also not be displayed if the `ConfigurationManagement::General::UseSCCtool` property is set to `No`.

About troubleshooting CM operations

This topic contains common problems encountered with CM operations.

Unable to connect to SCC-Compliant CM tool (SCC mode)

When you install an SCC-compliant CM tool, a tool-specific DLL is installed on your PC. When invoking a CM operation, Rational Rhapsody looks for an entry for this DLL in the system registry. If it finds one, it loads the DLL and performs the CM operation with that tool. If Rational Rhapsody cannot find the DLL for the correct SCC tool, it generates an error message. The `ProviderRegKey` value of the `SourceCodeControlProvider` key in the system registry (under `HKEY_LOCAL_MACHINE\SOFTWARE`) stores the location of the DLL for the default CM tool.



Note

The SCC interface is currently supported on Windows platforms only.

It is possible to have multiple SCC-compliant tools installed on the same system. In this case, there are multiple entries under the `InstalledSCCProviders` key (under `HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider`). For example:

```
PVCS Dimensions: REG_SZ: Software\SQL Software\PcmsScc
Microsoft Visual SourceSafe: REG_SZ: Software\Microsoft\
    SourceSafe
ClearCase: REG_SZ: Software\Atria\ClearCase
```

If the `ProviderRegKey` value does not correspond to the CM tool you want, select one of the installed SCC-compliant CM tools using a registry editor (for example, `Regedt32`) and edit the `ProviderRegKey` value of the `SourceCodeControlProvider` key with the information from `InstalledSCCProviders` for the SCC tool you want to use.

For example, to use PVCS Dimensions, edit the `ProviderRegKey` value with the following string:

```
Software\SQL Software\PcmsScc
```

Unable to create process message (Batch mode)

In some cases, there are problems with the standard `echo` command on the Windows 98/NT operating system.

If you receive an “Unable to create process” message while connecting to an archive because of an `echo` (either during a `Connect` operation or as a subtask of another CM operation), try to execute the same operation from a command prompt.

If the result is another message similar to “Unrecognized command,” correct the `PATH` variable for your environment. The `PATH` variable should include the full path in which the command to be executed is located.

CM Output window

Rational Rhapsody captures messages generated by the CM tool and displays them in the CM output window. It is important to observe messages from the CM tool because the commands sometimes fail. For example, if you try to check out a unit with a lock when the unit is already locked, the operation will fail.

To view the CM output window, click the **Configuration Management** tab on the Rational Rhapsody Output window.

The messages reported in response to CM operations, whether in the Output window or a special message box, are always from the CM tool. Error conditions must be corrected before you can proceed with any further CM operations.

To see all messages received by Rational Rhapsody from the CM tool during the session, scroll up or down using the scroll bar on the Output window.

To clear the output window of all CM messages, right-click in the window and select **Clear**.

The following properties control the appearance of text used to separate CM output messages in the CM output window:

- ◆ `ConfigurationManagement::General::CMOperationEndSeparator`
- ◆ `ConfigurationManagement::General::CMOperationStartSeparator`

To see the definition for an individual property, open the Features window and select the property on the **Properties** tab. The box at the bottom portion of the **Properties** tab shows the definition for the property selected in the upper left column of the tab.

Pre- and post- actions

Rational Rhapsody assumes a minimal configuration for all CM operations and takes certain actions either before (*pre-actions*) or after (*post-actions*) CM operations.

In SCC mode, post-actions are performed only if the operation is successful. In batch mode, Rational Rhapsody performs all post-actions regardless of whether the CM operation succeeds.

The following table lists the pre- and post-actions for SCC operations.

CM Operation	Pre-Action	Post-Action
ConnectToArchive	None	None.
CheckIn	Save	Update CM information.
CheckOut	Save (controlled by a property)	Load the unit into the current workspace.
AddMember	Save	Update CM information.
ListArchive	None	None.
Fetch	None	Load the unit into the current workspace.
Diff	None	None.
History	Save	None.
Properties	Save	None.
DeleteMember	Save	None.
RenameMember	Save	None.
MoveMember	Save	None.
Uncheckout	None	Load the unit into the current workspace.

The following table lists the pre- and post-actions for batch operations.

CM Operation	Pre-Action	Post-Action
ConnectToArchive	Save	None.
ListArchive	None	None.
DiffWithCM	None	None.
DiffWithRhapsody	None	None.
CheckIn	Save	Update CM information.
CheckOut	Save (controlled by a property)	Load the unit into the current workspace.
SCCAddMember	Save	Update CM information.
Lock	Save	Update CM information.
Unlock	Save	Update CM information.
DeleteMember	Save	None.
RenameMember	Save	None.
MoveMember	Save	None.
CheckoutBranch	None	None.

CM interface extension

User-defined buttons enable you to extend the Rational Rhapsody interface to support CM operations, which are not included in the standard Rational Rhapsody interface. Using Rational Rhapsody properties, you can assign commands to these buttons, such as submit, report defect on a version, extract, send by mail, and so on.

There are four properties, one for each definable button, located under `ConfigurationManagement::General::UserDefCommand_1` through `UserDefCommand_4`.

Items selected from the Configuration Items window are passed as arguments. Rational Rhapsody expands the keywords for each item selected in the Configuration Items list.

The format of the property is as follows:

```
Command ["ARG=
[$Path|$Ver|$Archive] [additional_parameters] "
[additional_parameters]
```

The keywords are as follows:

- ◆ `$Path` for full path + file name
- ◆ `$Ver` for Version
- ◆ `$Archive` for Archive session-specific string. This applies only to SCC model; it is empty in batch mode.

Note

Rational Rhapsody does not know if the operation completed successfully. The standard timeout applies in the case of a hanging script.

For example:

- ◆ To run the Rational ClearCase version tree on a selected item, set the property to the following value:

```
start cleartool.exe lsvtree -graph "ARG= $Path"
```
- ◆ To execute a batch file that uses the full path, version number, and archive string of a list of units, set the property as follows:

```
mycommand.bat "ARG= $Path, $Ver, $Archive ;"
```

Note

Activating various tools from a property is operating system-dependent. On Windows NT, use the following command:

```
start notepad.exe
```

Unresolved references

Every Rational Rhapsody configuration item can include references to other configuration items (*CI cross-references*). Once a configuration item is moved to another workspace, some of its references might no longer exist in the new context. These dangling references are called *unresolved references*. For example, if you replace an existing package with a newer version and the new package does not contain an item that appears in a certain view, references to that item will be unresolved.

The only CM operation allowed on unresolved elements is Check Out (and Fetch in SCC mode). The Synchronize operation can resolve an unresolved element.

Units added by reference

CM operations cannot be performed on units that have been added to the model as references.

Multi-site collaboration

Multi-site collaboration can be accomplished through Webify and rapid prototyping.

Webify for collaboration

Besides Web-enabling a Rational Rhapsody model for the purposes of controlling it and monitoring it remotely, Web-enabling can also serve as part of your development process. As a development tool, Web-enabling makes both remote and local collaboration possible and facilitates building and testing within a rapid prototyping approach.

Collaborating on the development of a Web-enabled model through the Web interface for the model enables developers to look right into the model's behavior and view its behavior as it is controlled. As if accessing the device from different windows, developers can collaboratively view the behavior of a model, or build from a working prototype, whether through the Rational Rhapsody interface or the Web interface for the model. Members of development teams with access to the application for the Web server can trigger events and change writable, Web-exposed element values. Anyone on the development team with access to the Web server for the model can take an immediate look at, and affect the status of, a model by opening a Web browser. When changing element values via the Internet through the Web GUI for the model, those changes occur real-time; immediately, within the animated diagrams of the Rational Rhapsody interface, local developers can see the model behavior as it is controlled through the Web interface by a remote team member.

In this way, you can use Web-enabling a model as a real-time use case, modeling a scenario of device usability, mimicking the device during different processes and testing its performance using the system itself to observe the system's behavior.

Rapid prototyping

To facilitate *rapid prototyping* of the application during development and testing of the device behavior in the lab on-the-fly, the Web interface can be customized to include other useful pages, such as an e-mail link for communicating errors and solutions or reporting bugs easily from within the Web interface.

Parallel development

This subject describes how multiple users and distributed teams can work in parallel with the use of the IBM Rational Rhapsody DiffMerge tool. These teams often have a source control tool or configuration management (CM) software, such as [IBM Rational ClearCase](#), to archive project units, but not all files might be checked into CM during development.

Engineers in the team need to see the differences between an archived version of a unit and another version of the same unit or a similar unit that might need to be merged. To accomplish these tasks, they need to see the graphical differences between the two versions, as well as the differences in the code. However, source control software does not support graphical comparisons.

Note

Many of the operations for the DiffMerge tool can be run from a command-line interface to automate some of the tasks associated with software development (for example, to schedule nightly builds). For more information about this feature, see [Command-line options for the DiffMerge tool](#).

The DiffMerge tool

The Rational Rhapsody DiffMerge tool supports team collaboration by showing how a design has changed between revisions and then merging units as needed. It performs a full comparison including graphical elements, text, and code differences.

What is a unit?

A Rational Rhapsody unit is any project or portion of a project that can be saved as a separate file.

Here are some examples of Rational Rhapsody units with the file extensions for the unit types:

- ◆ Class (.cls)
- ◆ Package (.sbs)
- ◆ Component (.cmp)
- ◆ Project (.rpy)
- ◆ Any Rational Rhapsody diagram

How do you use DiffMerge?

You can operate the DiffMerge tool inside and/or outside your CM software to access the units in an archive. There are two locations from which to launch the DiffMerge tool:

- ◆ [Launching DiffMerge inside Rational Rhapsody](#)
- ◆ [Launching DiffMerge outside Rational Rhapsody](#)

The DiffMerge tool can compare two units or two units with a base (original) unit.

The units being compared only need to be stored as separate files in directories and accessible from the PC running the DiffMerge tool. In addition to the comparison and merge functions, this tool provides these capabilities:

- ◆ Graphical comparison of any type of Rational Rhapsody diagram
- ◆ Consecutive walk-through of all of the differences in the units
- ◆ Generate a Difference Report for a selected element including graphical elements
- ◆ Print diagrams, a Difference Report, Merge Activity Log, and a Merge Report

Launching DiffMerge inside Rational Rhapsody

When launching the DiffMerge tool from inside Rational Rhapsody, you can compare two units in either of these environments:

- ◆ An archived version of the unit with the version currently displayed in Rational Rhapsody
- ◆ Two archived versions


Rational Rhapsody works with any CM archive in Microsoft® Common Source Code Control (SCC) mode or other tools in batch mode. Rational Rhapsody developers can use any of the following CM tools to manage their source files:

- ◆ [IBM Rational Synergy](#) in SCC mode
- ◆ [IBM Rational ClearCase](#) in batch mode and SCC mode
- ◆ [Serena PVCS Dimensions](#) in SCC mode
- ◆ [Concurrent Versions System \(CVS\)](#)
- ◆ [Subversion \(SVN\)](#)

Note

For Linux users, you can use batch mode configuration management, and Rational ClearCase. Rational ClearCase is supported in Linux.

To launch the DiffMerge tool inside Rational Rhapsody:

1. Be certain that your source control archive is accessible from the PC you are using for this comparison.
2. Start Rational Rhapsody and open a project.
3. Verify that the unit you want to compare (for example, a class) is a unit. A Rational Rhapsody element is a unit if a small red file appears in the lower left corner of its standard icon in the browser, as shown in this Class icon .
4. If the element is not yet a unit, change it into a unit by right-clicking the element and selecting **Create Unit**, and then click **OK**.

Note that you must check in any new unit into your CM system before it can be recognized by the DiffMerge tool.

5. Open the Configuration Items window. Choose **File > Configuration Items**.

Note: How this window looks depends on your CM system.

Compare With operation


Use the Compare With operation to compare an archived unit to the current version.

Comparing an archived unit to the current version

To compare the archived version of a unit to the current version of the same unit that has not been archived:

1. In the Configuration Items window, highlight a unit (for example, a class) in the displayed list that you want to compare to the archived version in your source control management system.



2. Click the Diff with Rhapsody button . Depending on what CM tool you have, the following actions might occur:
 - ◆ If you have Rational Synergy, the DiffMerge tool opens and compares the current Rational Synergy version with the current version in Rational Rhapsody so that the engineer can determine which version of the unit is going to be archived next.
 - ◆ If you have a CM tool other than Rational Synergy, the Compare With window opens:
 - Type either the revision or source control management label of the archived unit you want to compare to the one currently selected in the Rational Rhapsody model.
 - Select the **With Descendant** check box if you also want to compare any nested units inside the current unit to those of the archived version.



Note: The term *descendant* in the source control management refers to a unit that is nested inside another unit. For example, if P is a package, it might have a nested package Q and a global function $f()$ as its descendants. Q might be stored either in the same file as P or in its own file. In the latter case, Q is a descendant unit for source control management and comparison purposes. Taking P with descendants will also include Q (and its file). Taking P without descendants **will not** include Q , and the result of the merge will be two files, one for P and one for Q . The global function $f()$ cannot become a unit because it is a function, and must always “come and go” with P .

- Click **OK**.

The DiffMerge tool compares the two versions of the unit so that the engineer can determine which version of the unit is going to be archived next.

Comparing two archived versions

To compare two archived versions of the same unit that are both in the archive:

1. In the Configuration Items window, click the Show items in Archive button  to open the Archive window. The buttons that display in this window depend on what CM system you are using.
2. Select the unit with the two archived versions in the source control system.
3. Click the Diff with CM button  to open a Diff window.
 - ◆ Enter the **Revision/Label** information for the unit being compared.
 - ◆ Select the **With Descendant** check box if you also want to include any nested units in the comparison.
 - ◆ Click **OK**.

The DiffMerge tool compares the two versions so that the engineer can make the necessary decisions.

Advantages of launching DiffMerge inside Rational Rhapsody

Launching the DiffMerge tool inside Rational Rhapsody is particularly useful if you want to compare one version of an archived unit quickly with another version of the unit inside or outside the archive. You can quickly compare the two units using the DiffMerge tool inside Rational Rhapsody because you already have Rational Rhapsody and the CM system open.

To use the DiffMerge tool inside Rational Rhapsody, follow these rules for the units being compared:

- ◆ At least one of the units must be stored as a separate file in a CM system, such as Rational ClearCase.
- ◆ Units must be the same type and have exactly the same name since you select only one unit name to set up the comparison.

Compare with [Advantages of launching DiffMerge outside Rational Rhapsody](#).

Launching DiffMerge outside Rational Rhapsody

To launch the DiffMerge tool outside Rational Rhapsody, use any of these methods:

- ◆ From the Windows Start menu, choose **All Programs > IBM Rational > IBM Rational Rhapsody *version number* > Rational Rhapsody DiffMerge**.
- ◆ Use Windows Explorer to navigate to the Rational Rhapsody installation folder (for example, `<Rhapsody installation path>\Rhapsody75`), and double-click the `DiffMerge.exe` file.

Select units to compare

Use the Select Files window to specify the units to be used in a comparison in the Rational Rhapsody DiffMerge tool.

Selecting units to compare outside Rational Rhapsody

To specify the units to be used in the comparison:

1. To be certain you are comparing units of the same type, check the file extensions. They must be the same. For example, you can compare a project (`.rpy` file) only to another (`.rpy`) project, but not with a component (`.cmp` file). For more information, see [What is a unit?](#)
2. From the DiffMerge menu bar, open the Select Files window. Choose **File > Compare**. Note that the units might have different names.
3. In the **Left side Rhapsody unit** box, type the name including the path of the first unit or browse to the location. This file will be referenced as the “Left” in the comparison results.
4. In the **Right side Rhapsody unit** box, type the name including the path of the second unit or browse to the location. This file will be referenced as the “Right” in the comparison results.
5. Select the **Compare with descendants** check box if you want to include the nested units for these two units in the comparison.
6. Select the **Base-aware mode** check box if you want to browse to select the base or parent unit of the two previously selected files to use as the base-line for the comparison. If you use this third unit in the comparison, this creates a *base-aware comparison*. For more information about this type of comparison, see [For three units](#).

Note that for more information on this report, you might also want to see [Difference Report generation](#). Displaying base-aware information in this report is controlled by several [DiffReport preferences category](#) preferences.

7. Click **OK**. The DiffMerge tool compares the selected units and displays the results, as shown in [Examining “left” and “right” value selections](#).

Advantages of launching DiffMerge outside Rational Rhapsody

The advantage to launching the DiffMerge tool outside Rational Rhapsody is that the tool has greater flexibility. When you are using the DiffMerge tool outside Rational Rhapsody, you must locate the files to compare and bring them into DiffMerge. Launching the DiffMerge tool outside Rational Rhapsody is particularly useful if your comparison has any of these characteristics:

- ◆ Units are stored as separate files in a CM system or a directory that is not under source control.
- ◆ Units are the same type, but they might have different file names because the selection step allows you to enter different file names.

Note

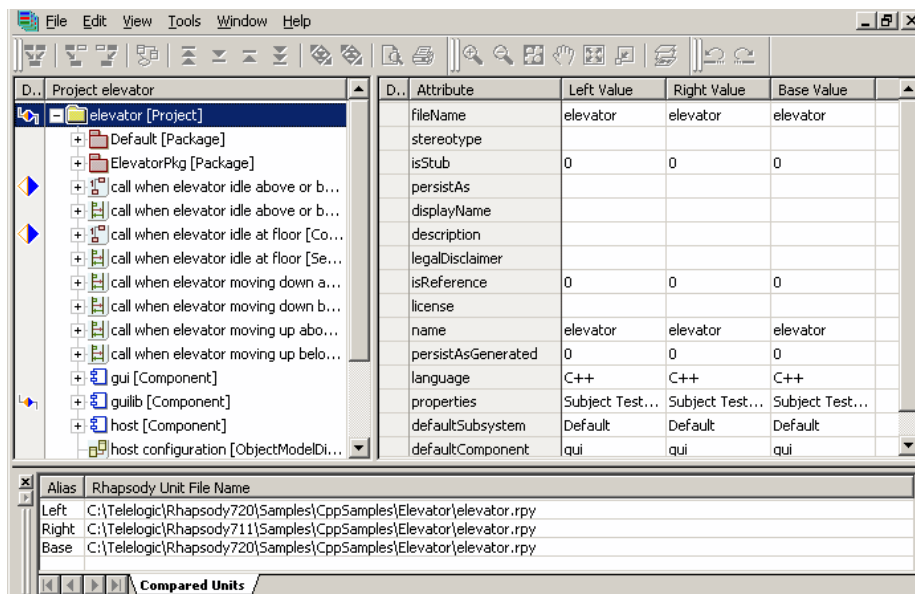
Always use the DiffMerge tool outside Rational Rhapsody to compare two units to a base unit (three units; see [For three units](#)). If you are comparing two units, you can use the DiffMerge tool inside or outside Rational Rhapsody depending on whether or not the units have different names.

Compare with [Advantages of launching DiffMerge inside Rational Rhapsody](#).

Examining “left” and “right” value selections

To examine the filenames and paths for the selected files:

1. With the compared units displayed in the DiffMerge tool, choose **View > Compared Units File Names**.
2. The Compared Units list appears at the bottom of the DiffMerge window, as shown in the following figure. Examine the following information:
 - ◆ Paths and file names of the selected units to be certain you have selected the correct files
 - ◆ Notice the “Alias” assigned to each unit (in the left column)
 - ◆ Base unit selected is correct for the comparison of the other two units.

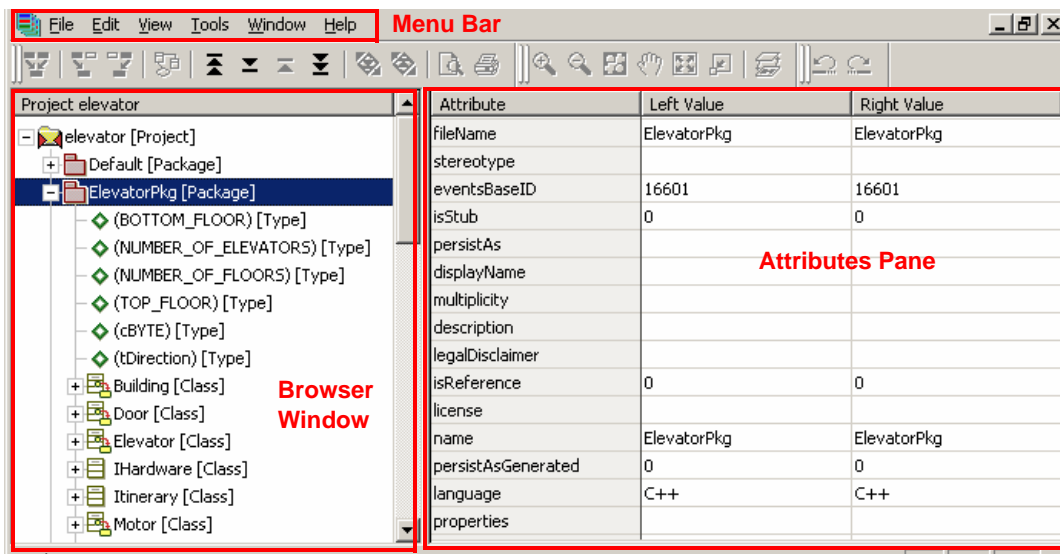


You can use **Compared Units File Names** at any time to see which file is the “Left” and which is the “Right.”

Results displayed in the DiffMerge tool

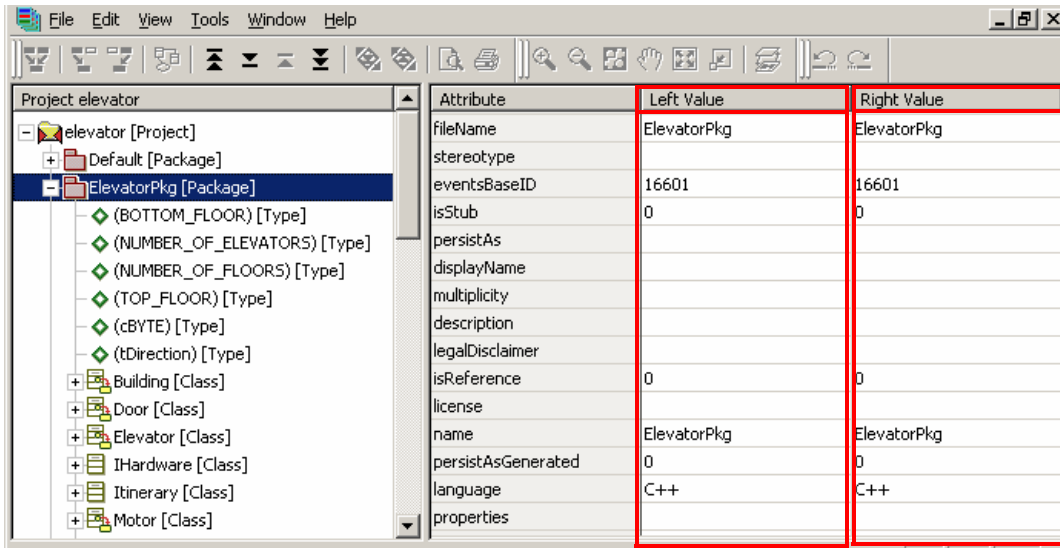
The DiffMerge tool contains a menu bar and two work areas, as shown in the following figure:

- ◆ **Browser** (left side) displays comparison results a tree structure. To see all the items in the browser, choose **View > Expand All**.
- ◆ **Attributes pane** (right side) provides text descriptions of unit elements.



For two units

With two units being compared (no base unit), the unit name is displayed in the title bar for the window and in the title bar for the DiffMerge browser. The first version of the unit selected, when setting up the comparison, is described in the **Left Value** column in the Attributes pane. The second unit selected has its information listed in the **Right Value** column. For descriptions of the browser icons, see [Difference categories and their icons in the browser](#).



The Attributes pane lists specific information about the meta-elements (attributes) that are different in the units being compared, such as the class names and properties. Elements with differences are displayed in boldface, dark red text.

For three units

If the comparison includes a base unit, the **Base Value** column is added to the columns in the Attributes pane to create a *base-aware comparison*. This type of comparison is used when a baseline for a Rational Rhapsody unit is needed. For example, during a maintenance release the changed Rational Rhapsody unit files are checked in on a different branch and after the maintenance release changed unit files needs to be merged back to the main branch.

During this process, the developer finds it easier to make decisions if the two Rational Rhapsody unit files versions are compared the base (common ancestor) version. The developer can scroll through any non-trivial differences (described in [Trivial Versus Non-trivial Differences](#)) using these toolbar icons.



Next non-trivial difference



Previous non-trivial difference

In the following figure, the base unit for the Elevator project is compared to two different versions of the project. In addition to the standard browser symbols, shown in the two unit comparison, both the browser and the Attributes pane include an additional Diff icon column with visual representations of each difference located in the base-aware comparison. For definitions of the Diff icons, see [Differences report in the Output window](#)

D.	Project elevator	D.	Attribute	Left Value	Right Value	Base Value
	randomElevator		concurrency		Sequential	Sequential
	randomFloor		stereotype			
	randomTime		preferBodyToActivityGra...		1	1
	theElevator		protection		!Public	!Public
	~Building		returnTypeIsOnTheFly		0	0
	Door		returnType		PredefinedTypesCpp::...	PredefinedTypesCp..
	Elevator		constant		1	0
	IHardware		displayName			
	Itinerary		description		returns either Elevator..	returns either Elevat..
	Motor		Static		1	0
	call elevator		abstract		0	0
	enter elevator		ItsBody		return (rand() % 2);	return (rand() % 2)
	evAtFloor		Virtual		0	0
	evCall		name		randomElevator	randomElevator
	evChangeDirection		properties			
	evChanged		final		0	0
	evClosed					
	evGotObstacle					
	evGoto					

Differences report in the Output window

To display the Difference Report, use one of the following methods:

- ◆ Right-click an element in a diagram and select **Report Differences**.
- ◆ Choose **Tools > Report Differences > All** to compare all the elements or **Tools > Report Differences > Selected** to compare only the selected elements.

Difference Report display

DiffMerge displays the differences in the **Difference Report** tab of the Output window, as shown in the following figure for a simple three-unit comparison.

The screenshot shows the Eclipse IDE with the 'Project elevator' project selected. The 'Difference Report' tab is active, displaying the following content:

```

=== Reporting differences for Project elevator (cannot be merged automatically) ===
>> Project elevator:
  >> Package ElevatorPkg:
    >> Class Building:
      # PrimitiveOperation configure changed on both sides (non-trivial diff)
      : # Attribute "ItsBody" changed on both sides (non-trivial diff)
      # PrimitiveOperation dispatch changed on both sides (non-trivial diff)
      : # Attribute "ItsBody" changed on both sides (non-trivial diff)
      - PrimitiveOperation randomTime deleted on the left (trivial diff)
      % PrimitiveOperation randomElevator deleted on the left and changed on the other side (non-trivial diff)
      : % Attribute "Static" deleted on the left and changed on the other side (non-trivial diff)
      : % Attribute "constant" deleted on the left and changed on the other side (non-trivial diff)
  
```

The background shows a table of differences for the 'Building' class:

D.	Attribute	Left Value	Right Value	Base Value
	concurrency		Sequential	Sequential
	stereotype			
	preferBodyToActivityGra...		1	1
	protection		iPublic	iPublic
	returnTypesOnTheFly		0	0
	returnType		PredefinedTypesCpp::in...	Predefinex
	constant		0	0
	displayName			
	description		random time between 2 ...	random ti
	Static		0	0

Features of a Difference Report

The following illustration identifies important features of a Difference Report.

The screenshot shows a Difference Report for the class `AcmeFactory`. The report is structured as follows:

```

===Reporting differences for Class AcmeFactory===
<> Differences found for Class AcmeFactory
  fileName
  name
  properties
  > Dependency AcmeHeater exists only on left side
  > Dependency AcmeJet exists only on left side
  > Dependency AcmeTank exists only on left side
  > Generalization AbstractFactory exists only on left side
  < PrimitiveOperation theFactory exists only on right side
  <> Differences found for PrimitiveOperation createTank
    ItsBody
    Virtual
    properties
  <> Differences found for PrimitiveOperation createJet
    ItsBody
    Virtual
    properties
  <> Differences found for PrimitiveOperation createHeater
    ItsBody
    Virtual
    properties
  
```

Callouts from the right side of the image point to specific features:

- Difference Report header**: Points to the line `===Reporting differences for Class AcmeFactory===`.
- The AcmeFactory class has differences from the other class, AbstractFactory.**: Points to the red header `<> Differences found for Class AcmeFactory`.
- In addition to the differences in the class names, DiffMerge found differences in the properties attribute.**: Points to the `properties` sub-section under `AcmeFactory`.
- The 3 dependency and 1 generalization differences only exist in the AcmeFactory.**: Points to the blue lines indicating differences only on the left side.
- This difference exists only in AbstractFactory (right side).**: Points to the green line indicating a difference only on the right side.
- Three methods have these differences: implementations (in ItsBody), Virtual values, and properties.**: Points to the nested difference sections for `createTank`, `createJet`, and `createHeater`.

The bottom of the window shows a tab labeled "Difference Report" and other tabs for "Merge Activity Log" and "Merge Report".

The Output window uses the following colors to distinguish between the difference categories:

- ◆ **Red** denotes a difference element.
- ◆ **Blue** denotes elements that exist only on the left side.
- ◆ **Gray** denotes a *nested difference*. A nested difference is an element without any differences, but that contains an element with either a difference, left-side only, or right-side only element.
- ◆ **Black** denotes a no difference element.
- ◆ **Green** denotes elements that exist only on the right side.

You can control the text used to mark each difference category, write the output to a file, and customize the DiffMerge tool by setting the appropriate preferences. For more information, see [Colors preferences category](#).

Notice that the Output window also displays the [Merge activity log](#) and [Producing merge reports](#).

DiffMerge differences

You can right-click any item in the DiffMerge browser and select **Browse from here** to limit or focus the scope of the current view of the browser and Attributes pane. Doing this opens the Browse From Here browser and associated Attributes pane.

The item you select in the browser specifies the commands that are displayed in the pop-up menu.

For example, if a developer selects a diagram in the browser, additional options for diagrams are available so that all of the possible options are displayed in the pop-up menu:

- ◆ **View all diagrams** lets you review all the diagrams to help you decide from which side you want to take your changes.
- ◆ **View left diagram** lets you view only the diagram from the left side.
- ◆ **View right diagram** lets you view only the diagram from the right side.
- ◆ **View both diagrams** lets you view both the diagrams for this selection from the right and left.
- ◆ **View base diagrams** let you view the base diagram.
- ◆ **Merge graphically** lets view all versions of the object model diagram, including the newly created merged object model diagram in the Merge window.
- ◆ **Browse from here** opens a new browser and Attributes pane for the selected element.
- ◆ **Report Differences** creates the Difference Report for the selected element.
- ◆ **Report Merging Differences** creates the Merge Report for the selected element.
- ◆ **Next diff** moves the selection in the browser to the next difference in the browser.
- ◆ **Prev diff** moves the selection in the browser to the previous difference in the browser.

Differences in the browser

When selecting the **Browse from here** command, the DiffMerge tool opens another browser that displays that element at the top of the browser and the associated attributes in the Attributes pane. The more focused browser is called the Browse From Here browser.

For more information about the standard icons in the browser, see [Difference categories and their icons in the browser](#).

Note

After browsing a selected element, you can display the original comparison (if you closed it) at any time by selecting **View > New Diff Browser**.

Difference categories and their icons in the browser

When comparing two units, triangle overlays are displayed in the browser items whether two or three units were compared. If you want to display the triangles on the icons indicating whether the difference is on the right, left, or both sides of the comparison, set the [ShowDMMarksInBaseAwareMode](#) preference to be `Checked` (check box is selected).

- ◆ **No difference element** means the element, including all its nested elements, is identical on both sides of the comparison. In the browser, a no difference element has the same icon as in Rational Rhapsody, as in this example:



- ◆ **Right-only element** means the element exists only on the right side of the comparison. For example, the element is new or was deleted from the left unit. In the browser, a right-only element has a left-facing, green arrow overlaid on the icon, as in this example:



- ◆ **Left-only element** means the element exists only on the left side of the comparison. For example, the element is new or was deleted from the right unit. In the browser, a left-only element has a right-facing, blue arrow overlaid on the icon, as in this example:



- ◆ **Difference element** means the element exists in both sides of the comparison, but some of its fields or properties are different. In the browser, a difference element has dual-facing, red arrows overlaid on the icon, as in this example:



- ◆ **Nested difference** means the element exists on both sides of the comparison, but some of its subelements are different. There is no two-unit comparison icon for the Nested difference, but there is a Nested difference icon in a base-aware comparison.

Base-aware Diff icons

In the Diff column in base-aware comparisons, the icons illustrate the types of differences between the two units and the base. Move your mouse over an icon to see a definition for it.

The basic design features of these icons are as follows:

- ◆ Modification on the right side with a blue triangle (pointing right)
- ◆ Modification on the left with an orange triangle (pointing left)
- ◆ Deletions have a minus sign on the side that has the deleted item
- ◆ Additions have a plus sign on the appropriate side

The icons and what they mean are as follows:



means the item contains at least one nested difference



means the item was modified on both sides of the comparison



means the item was modified on the left and deleted on the right



means the item was added to the left side only



means the item was modified on the left side



means the item was modified on the right side







means the item was deleted from the right side

To see an example of a DiffMerge browser and Attributes pane with these icons, see [For three units.](#)

DiffMerge tool navigation

The following buttons let you navigate quickly in the browser and diagrams by changing the view:

Button	Button Name	Explanation
	First difference (diagrams only)	Moves the selection to the first difference in the displayed diagrams.
	Prev difference	Moves the selection to the previous difference in the browser or displayed diagrams.
	Next difference	Moves the selection to the next difference in the browser or displayed diagrams.
	Last difference (diagrams only)	Moves the selection to the last difference in the displayed diagrams.

Using the commands in the View menu, you can open additional views to compare units in different ways. In the **View > Diff** mode, choose:

- ◆ **View > View all** to view all model elements, including those that are the same in both units.
- ◆ **View > View diff** to view only those model elements that are different in the two units.
- ◆ **View > View Conflicts** to view only those model elements with non-trivial differences that will require manual merging.
- ◆ **View > View undecided** to view differing elements that are neither in nor out of the merge. Note that this option is available only in merge mode. For more information, see [Undecided view](#).
- ◆ **View > View in merge** to view only those elements that are currently in the merge. Note that this option is available only in merge mode. For more information, see [View in merge](#).

You can also navigate in the list of differences in the Attributes page. Right-click in the Attributes pane to open the pop-up menu, which contains the following commands:

- ◆ **Diff text** launches the external textual difference/merge tool (such as TkDiff)
- ◆ **View all** displays all the model elements, including those that are the same in both units
- ◆ **View diff** displays only those model elements that are different in the two units
- ◆ **Next diff** displays the next difference in the list
- ◆ **Prev diff** displays the previous difference in the list

See [Results displayed in the DiffMerge tool](#) for a description of the Attributes pane.

The external difference/merge textual tool

For some attributes, the space provided in the Attributes pane is insufficient to view them properly (for example, a multi-line box). In such cases, you can launch an external textual diff tool.

To open a textual diff tool, in the Attributes pane, right-click an element and select **Diff text**. By default, Rational Rhapsody opens the TkDiff diff tool to display information about the selected element.

Using your external difference/merge textual tool

To use your favorite textual diff tool in Rational Rhapsody:

1. Open the Preferences window. Choose **View > Preferences**.
2. Expand the **TextDiffMerge** category.
3. Change the **DiffInvocation** and **DiffMergeInvocation** preferences, as described in [TextDiffMerge preferences category](#).

You can also edit the **DiffInvocation** preference to launch a different Diff editor than TkDiff, assuming that the editor can be launched from the command line.

To use the textual diff tool supplied with [IBM Rational ClearCase](#), modify the **DiffInvocation** and **DiffMergeInvocation** preferences, as described in [TextDiffMerge preferences category](#).

Filtering the comparison in the DiffMerge tool

You can ignore certain meta-elements during the comparison by modifying the settings in the Preferences window for the DiffMerge tool.

To filter the comparison in the DiffMerge tool:

1. Open the Preferences window. Choose **View > Preferences**.
2. Expand the **Suppressions** category.
3. Change the value for the **DiffAttributesFilter** preference. The default preference is as follows:

```
id, lastID, ImportData, cmheader,  
state, RequirementTracabilityHandle,  
isSaveUnit, isUR, isNameGenerated,  
isReadOnly, errorStatus, version,  
baseVersion, defNumber, directoryName,  
CPUtype, icon, isTemplate, typeID,  
stereotypeID, DependsOnID, DependsOnImportData
```

4. You can edit the **DiffAttributesFilter** preference to add other elements to ignore, if you want. Preferences set in **DiffAttributesFilter** affect the DiffMerge tool if you change them in the `site.prp` file before DiffMerge is launched.

For more information, see:

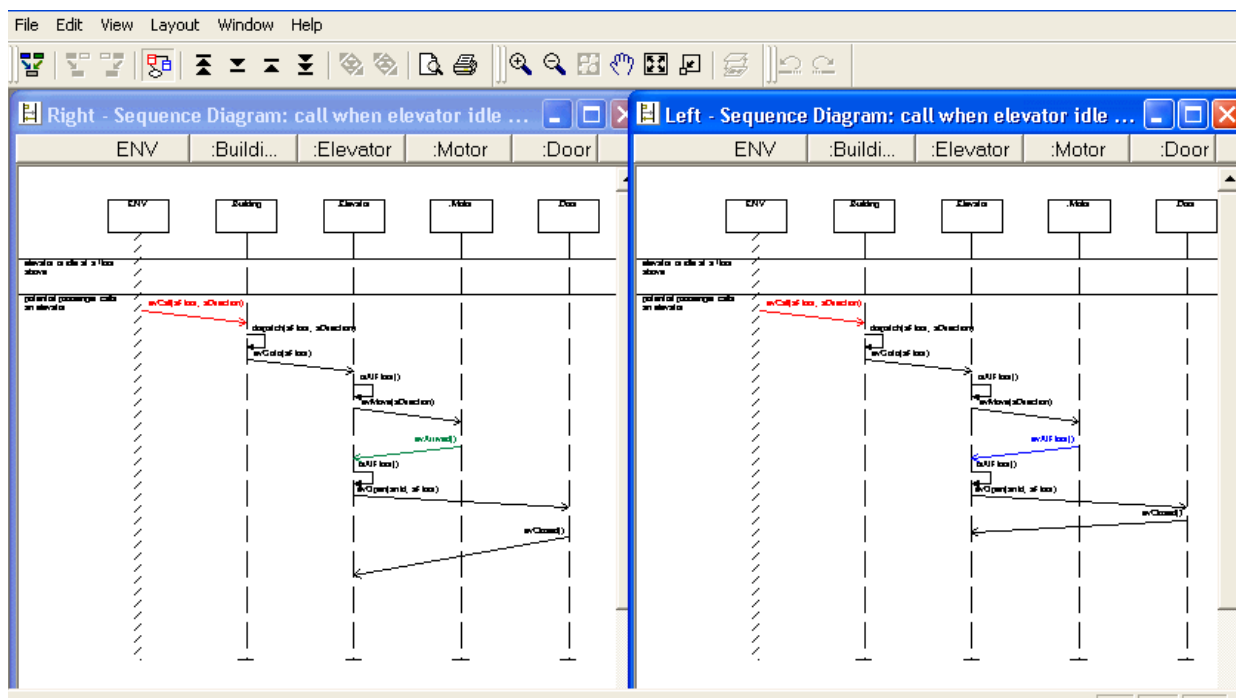
- ◆ [DiffMerge tool preferences](#)
- ◆ [Changing preferences](#)
- ◆ [Suppressions preferences category](#)

Inspecting differences in diagrams visually

The DiffMerge tool identifies visual differences underlying the model elements.

To show a graphical differences between two diagrams:

1. After displaying the units to be compared in the DiffMerge tool, open the browser tree to show the diagrams included in the comparison.
2. In the browser, right-click the diagram of interest and select **View Diagrams**. Read-only views of the two diagrams open in individual windows.
3. Click the highlighted item. The following illustration displays a comparison of two versions of a sequence diagram from the same project.



Differences in diagrams are highlighted according to the color scheme described in [Differences report in the Output window](#).


Graphical differences

When comparing the graphical features of diagrams, the user can use two features to help identify the differences between the two diagrams:

- ◆ Highlight Differences
- ◆ Walk-through Differences

Switching on the difference highlighting

For diagrams that exist on both sides of comparison and a diagram view is active, to use colors to highlight all different elements of the diagram:

1. Select a diagram in the DiffMerge browser to use for the highlighted comparison.
2. Choose **View > Highlight Differences** or the toolbar icon .

Note

Differences are highlighted using the settings in [Colors preferences category](#).

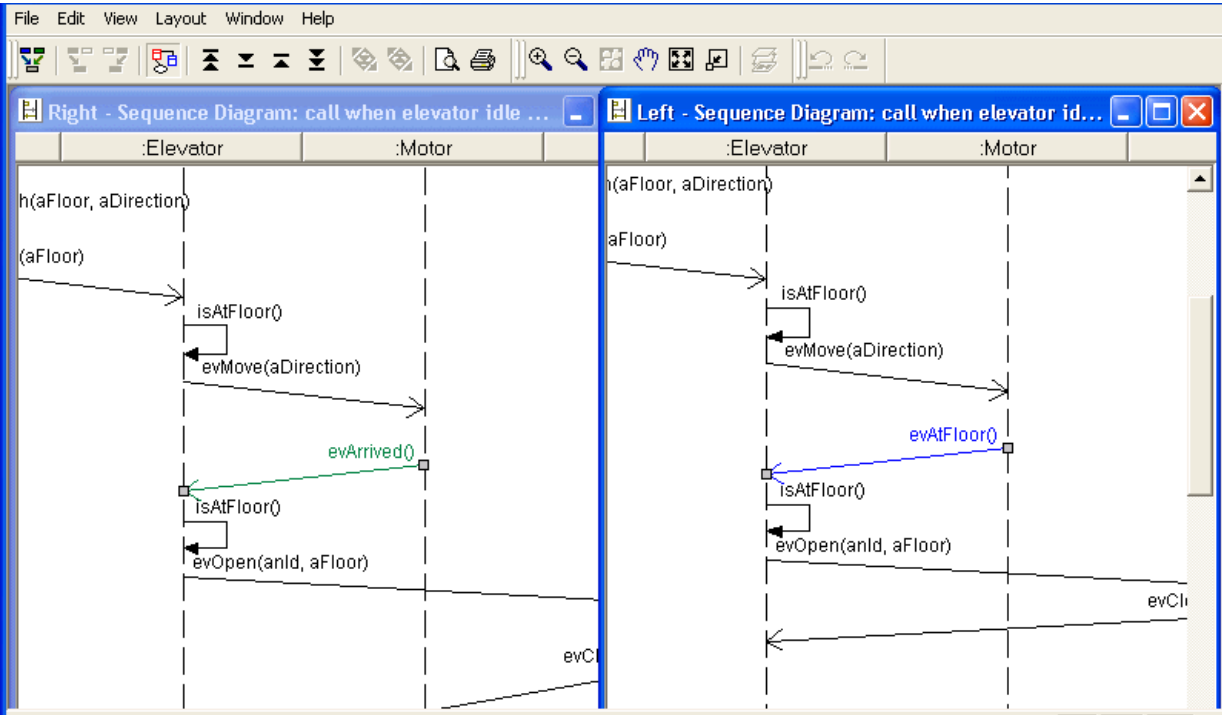
Walking through diagram differences

For diagrams that exist on both sides of comparison, you can walk-through the diagram differences.

To switch on the walk-through features:

1. Select a diagram in the DiffMerge browser to use for the walk-through comparison.
2. Right-click and select one **View right diagram**, **View left diagram**, or **View both diagrams** to place the selected diagrams in the Attributes pane.

3. Select **First**, **Next**, **Previous** or **Last** difference options or use the shortcut keys to walk through the graphical differences in the displayed diagrams, as shown in the following figure:



Note

At first each difference is highlighted with a heavier, colored line, and the highlighted line blinks to draw the user’s attention to the item. Then the heavier width of the line disappears, but the highlighted color of the items remains, as shown in this sequence diagram comparison. You might want to see about [Switching off element blinking](#) for this feature.

The walk-through menu options have the following results:

- ◆ **First** displays the first graphical difference located in the diagrams.
- ◆ **Next** marks the next difference.
- ◆ **Previous** steps back to show that difference displayed before the currently highlighted one.
- ◆ **Last** marks the last different diagram element.

You can also use the following shortcut keys to navigate through the graphical differences.

- ◆ Press **Alt-Home** key to go to the first difference
- ◆ Press **Alt-Right** arrow to go to the next difference
- ◆ Press **Alt-Left** arrow to go to the previous difference
- ◆ Press **Alt-End** key to go to the last difference

Switching off element blinking

If you prefer to switch off the blinking feature:

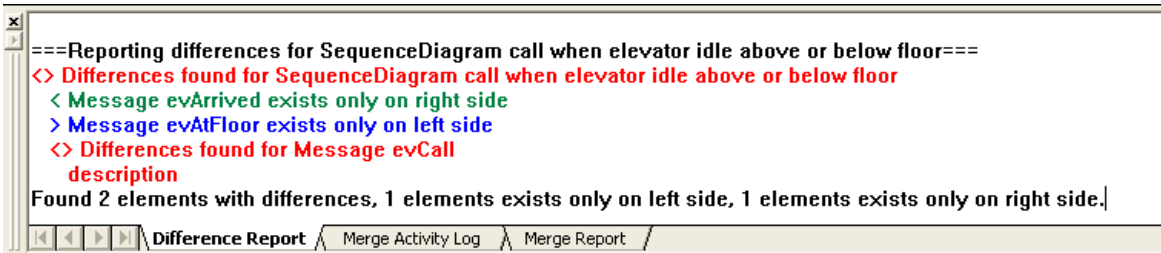
1. From the DiffMerge menu bar, open the Preferences window. Choose **View > Preferences**.
2. Expand **DiagramViews**.
3. Clear the check box for the **BlinkWalkingThroughDiffs** preference.

Note

Other preferences are described in [Changing preferences](#).

Difference Report generation

To generate a list of the differences for a selected diagram element during graphical comparison, right-click the diagram element in the DiffMerge browser and select **Report Differences**. The differences for that element display in the Output window, as shown in the following figure:




```
====Reporting differences for SequenceDiagram call when elevator idle above or below floor====
<> Differences found for SequenceDiagram call when elevator idle above or below floor
  < Message evArrived exists only on right side
  > Message evAtFloor exists only on left side
  <> Differences found for Message evCall
    description
Found 2 elements with differences, 1 elements exists only on left side, 1 elements exists only on right side.
```

Printing a Difference Report

To print a report showing the differences displayed in the Output window:

1. Decide whether or not you want to ignore graphic differences in the report, choose **View > Ignore Graphical Differences** as many times as needed.

A check mark to the left of **Ignore Graphical Differences** means this option is selected.

2. Display the information in the Output window:
 - ◆ Right-click an item in the browser select **Report Differences**, or
 - ◆ Choose **Tools > Report Differences > All** or **Selected**
3. Click the Printer button  to print the information in the Output window.

Note

The Difference Report does not specify details about graphical differences. Only the existence of a graphical difference is reported so that the developer can perform any required analysis.

Graphical differences suppression

If the graphical differences between two units are not important to the comparison, you can suppress the graphical differences from the comparison and the differences report. With the compared units displayed in the DiffMerge tool, to suppress the graphical differences, choose **View > Ignore Graphical Differences** to make a check mark appear to the left of **Ignore Graphical Differences**.

Note

You use **Ignore Graphical Differences** to toggle this feature on and off, but only when the tool is in comparison mode. As soon as you start merging, this menu option is disabled. Therefore, you must determine whether or not to ignore graphical differences *before* starting a merge operation.

To merge two classes with different statecharts, use the browser (see [Starting a merge operation](#)) or the graphical DiffMerge function (see [Merging diagrams graphically for statecharts and activity diagrams](#)).

DiffMerge limitations

Note these limitations:

- ◆ In terms of the display, DiffMerge only provides partial support for specialized profiles such as SysML. For example, the icons displayed in the DiffMerge browser are the standard Rational Rhapsody icons, not the specialized icons included in the profile.
- ◆ When comparing sequence diagrams, DiffMerge reports message re-ordering as logical differences (see [Logical versus graphical differences](#)). However, only message re-ordering is detected, while sequence modifications of other sequence diagram elements (such as Condition Mark, Interaction Occurrence, and Destruction Event) are disregarded.

Logical versus graphical differences

For the purposes of DiffMerge model difference detection, a logical difference is one that changes the logic of a model. This is also known as a model difference. In comparison, a graphic difference is a visual difference that does not have an effect on a model.

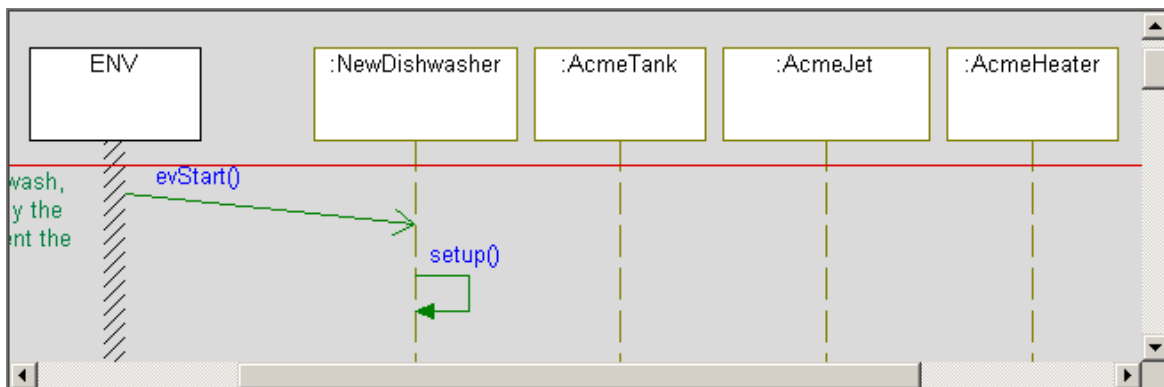
DiffMerge always reports logical differences, while it might ignore graphical differences (see [Graphical differences suppression](#)).

Example of logical difference

The following examples shows a logical difference on a sequence diagram.

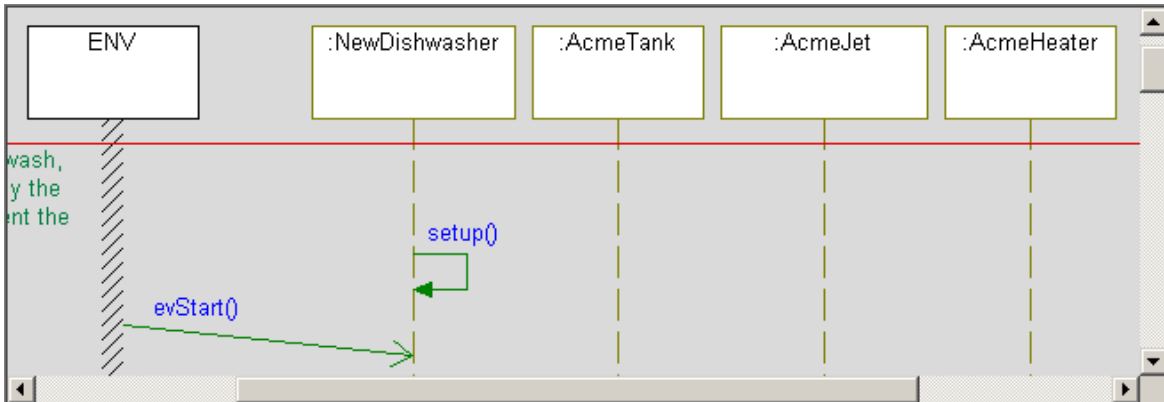
Example 1a: Sequence diagram with evStart()

The following figure shows a partial view of the Dishwasher Cycle sequence diagram for the Dishwasher sample project provided with the Rational Rhapsody product. Notice the location of `evStart()` above `setup()`. For aesthetic reasons, you could reposition `evStart()` (say, move it down slightly so that it does not touch the partition line) but leave it still above the position of `setup()` and this movement is ignored by DiffMerge because it does not affect the logic of the model.



Example 1b: Sequence diagram with evStart() Moved

Notice the location of `evStart()`, which has been moved below `setup()`. This change has an effect on the logic of the model.



Example 1c: DiffMerge

The following figure shows in DiffMerge a comparison of the sequence diagrams shown in Example 1a (Left Value column) and 1b (Right Value column) in base-aware mode (Base Value column). As you can see, the logical difference is noted in the Right Value column.

D..	Attribute	Left Value	Right Value	Base Value
	stereotype			
	ReturnVal			
	ActualArgs			
	FreeText			
	displayName			
	Type	EVENT	EVENT	EVENT
	description			
	name	evStart	evStart	evStart
	properties			
	Order		Moved Down on the Right	

=== Reporting differences for Message evStart (can be merged automatically) ===
 * Message evStart changed on the right (trivial diff)
 : * Attribute "Order" changed on the right (trivial diff)
 — Found 1 elements with differences, 0 elements exists only on left side, 0 elements exists only on right side —

DiffMerge reports

DiffMerge allows you to export reports that summarize the differences found between compared units.

Note

These reports summarize only the differences found between the compared units, not any merges that were subsequently made.

Exporting DiffMerge reports

To export a report:

1. Choose **Tools > Export Report > Rich Text Format** or **Tools > Export Report > CSV Format**.
2. When prompted, provide the path where you would like the report to be saved.

The CSV format is useful when you want to perform further analysis on the difference data.

The reports contain information regarding the following types of differences:

- ◆ element-level differences
- ◆ attribute-level differences
- ◆ diagram differences
- ◆ code-level differences

Note

The content of these reports is not identical, the Rich Text Format (RTF) report contains a greater level of detail, while the CSV format report focuses on the kind of information that you would want to use for statistical analysis.

In RTF report reports:

- ◆ When differences are found in diagrams, the report displays the different versions of the diagrams, using color to indicate the differences between the versions.
- ◆ When differences are found in attributes of type text, the differences are displayed line-by-line, using the following symbols: <> (exists on both sides), ++ (exists on this side only), __ (does not exist on this side).
- ◆ Code-level differences are reported as attribute-level differences (the attribute name is `IsBody`).

When exporting reports, remember that:

- ◆ You cannot create more than one report at a time.
- ◆ You should not close DiffMerge until creation of the report has been completed.

The Rational Rhapsody DiffMerge process

This topic provides you with an overview of the Rational Rhapsody DiffMerge process. The Rational Rhapsody DiffMerge tool makes a comparison of two units and looks for matches of the elements within their respective units. The units can be from the same Rational Rhapsody project or two different versions of the same unit. In the case of base-aware mode, you can also designate a third unit, which is the base unit. See [What is a unit?](#)

How does the DiffMerge tool make a match?

By default, the Rational Rhapsody DiffMerge tool tries to make a match by the name of elements and then by ID if there is no name match. This means the DiffMerge tool can detect and report if an element has been renamed and has a different name on each comparison side. This method makes it clearer for a developer/engineer what the differences are so that they can more easily decide which name from which side to take as the merge result for a particular element.

If you prefer, you can set the DiffMerge tool to only make matches by name only. This means the name of an element in one unit must match the name of an element in the other unit for a match to be made. If you prefer to compare the elements in units with this method, you can specify this in the [ElementMatchRule](#) preference.

For examples of results of the two compare methods, see [Example 1: Element is renamed](#).

Note

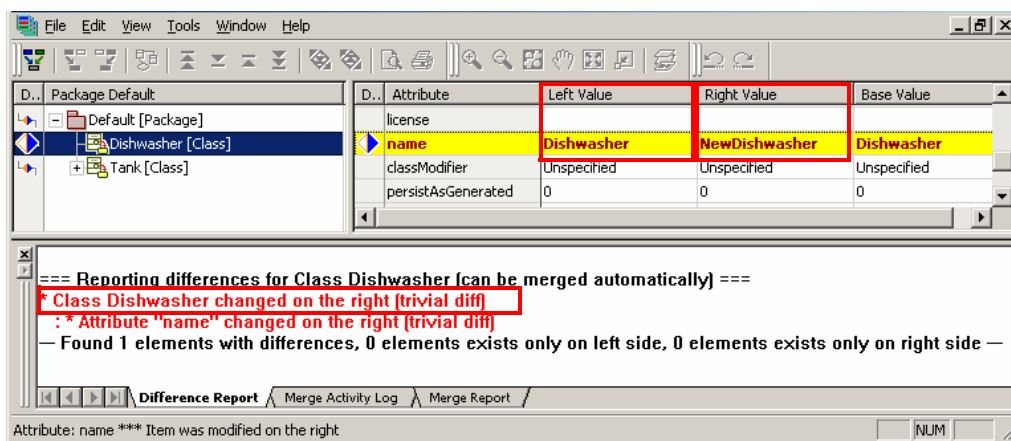
When a name change is a trivial difference, DiffMerge automatically merges it. See [Trivial Versus Non-trivial Differences](#).

Examples of how the DiffMerge tool handles renamed elements

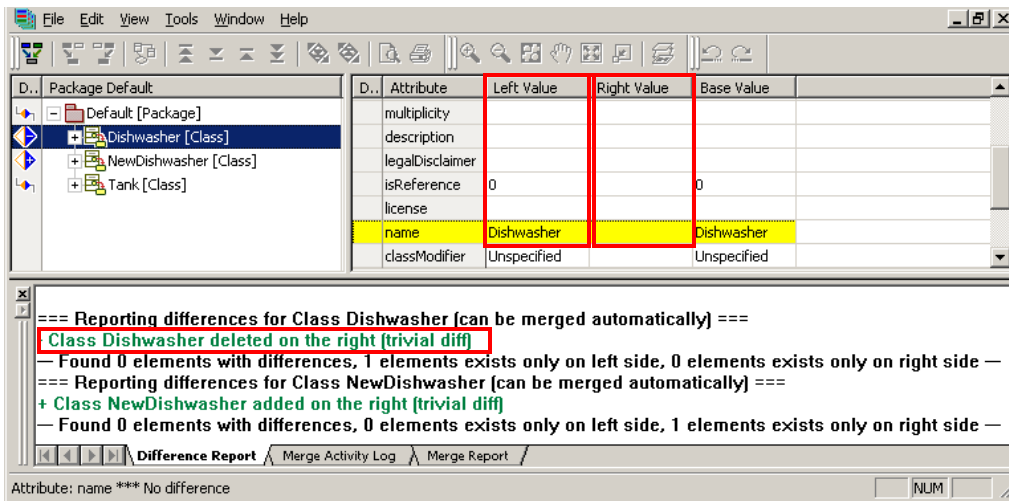
The following examples show you various scenarios of how the DiffMerge tool handles renamed elements.

Example 1: Element is renamed

The following figure shows a comparison that found a match by element ID when a name match could not be found. This is the default method used by the DiffMerge tool. As you can see, the **Dishwasher** class (in the Left Value column) has been renamed to **NewDishwasher** (in the Right Value column). The Difference Report shows the change as a trivial diff.

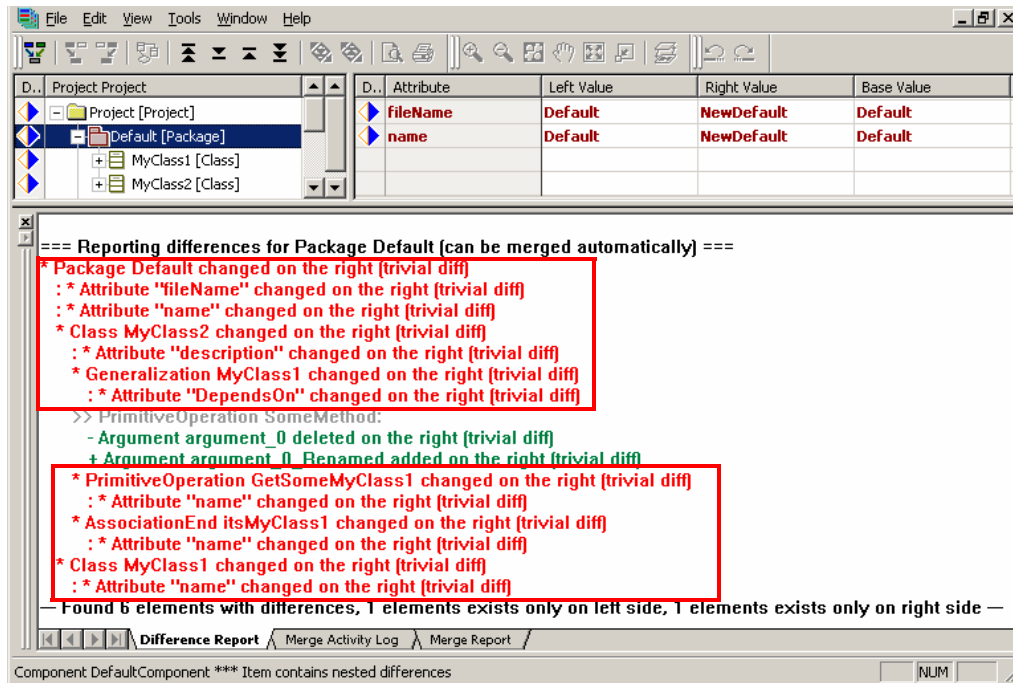


The following figure shows a comparison by name of element only. As you can see, the renamed class (**NewDishwasher**, as shown in the Right Value column in the previous figure) does not appear in the Right Value column in this comparison method. Notice that the Difference Report reports that the unit (**Dishwasher**) has been deleted, which is untrue. As you can see from the previous figure, the class was only renamed.



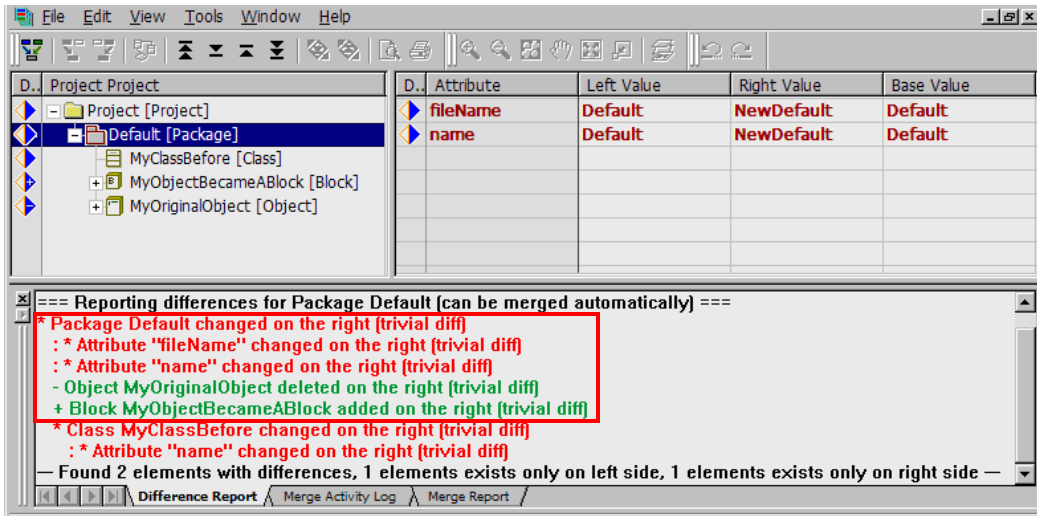
Example 2: More than one element is renamed

The following figure shows a comparison where more than one element has been renamed. In addition, nested elements have been renamed.



Example 3: Element is renamed and type is changed

The following figure shows a comparison that found matches by element IDs where the **MyOriginalObject** element was renamed to **MyObjectBecameABlock**, and then was converted into a block. Notice that the DiffMerge tool reports two different elements as before.



How DiffMerge performs a model comparison

The Rational Rhapsody DiffMerge tool makes a comparison of two units and looks for matches of the elements within their respective units. The units can be from the same Rational Rhapsody project or two different versions of the same unit. In the case of base-aware mode, you can also designate a third unit, which is the base unit. See [What is a unit?](#). The compare algorithm works as follows:

- ◆ When two model elements have the same type (or, as known in Rational Rhapsody, metaclass), name, and parent model element, then they are considered to be a match. If an element has no match by name, DiffMerge tries to find a match by ID. Matching elements always belong to the same parent and have the same metaclass.
- ◆ When two model elements are matched, then all their attributes and references to other model elements are compared to check if there are any differences.
- ◆ When two model elements are matched, then their aggregated model elements are compared the same way to check for differences. They are matched by metaclass and name or ID, their attributes and relations are checked for differences, and their aggregates (if any) are compared the same way. Basically, the process goes recursively through all aggregates.

You might also want to see [Automatic merging for base-aware comparisons](#) and [Trivial Versus Non-trivial Differences](#).

How differences are detected in base-aware comparisons

In two-way comparison, the ability for the DiffMerge tool to describe difference is straightforward. It reports the difference, but does not explain why.

In base-aware mode, the DiffMerge tool reports differences with more details that describe what caused the differences. DiffMerge is able to do so because it takes into account the third unit, the base unit. So each element (in the Left Value column) is matched not only with an element from another side (in the Right Value column) but also a base element (in the Base Value column).

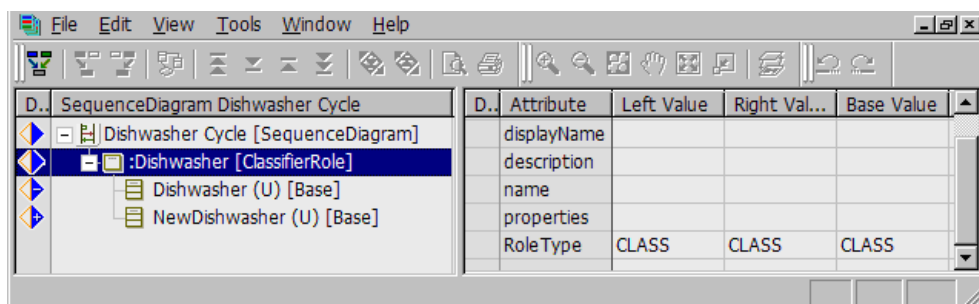
For more information about base-aware difference detection, see [Base-aware Diff icons](#).

All differences identified will be compared and put together to determine if they are trivial or non-trivial differences. See [Trivial Versus Non-trivial Differences](#).

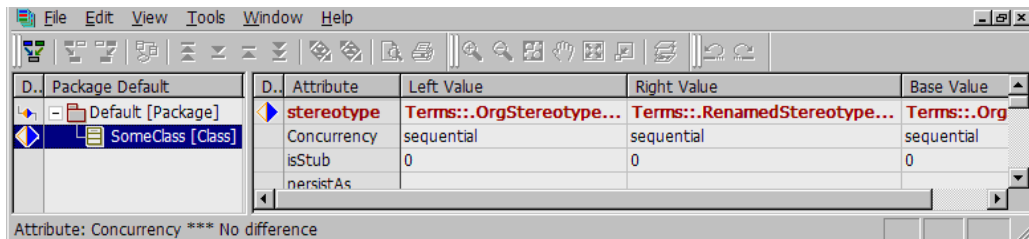
Limitations for match by element ID in DiffMerge

Note these limitations for the match by element ID function in the DiffMerge tool:

- ♦ DiffMerge will neither detect renamed nor suppressed propagated renamed differences when the renamed object is out of the comparison scope. (For information about suppressing propagated renamed differences, see [How to examine only major structure differences.](#)) For example:
 - When comparing sequence diagrams, the base classes for a classifier role are displayed as “single” nodes though it is the same class with modified name, but this class is not within this comparison, as shown in the following figure:



- When comparing packages, a class stereotype difference is not suppressed though it is the same stereotype having different names. That might be because the stereotype belongs to a package named, for example, Terms, that is out of the scope of this comparison.



- ◆ When performing a merge of units containing renamed elements, in some cases, DiffMerge does not properly update merged attributes values. This happens with the following attributes:
 - `DependsOn` in generalizations and dependencies when the related class is renamed.
 - `Stereotype` in all model elements when the stereotype is renamed.
 - `Type` when the type is renamed, and in all model elements when type is applicable, including: methods and operation return types; arguments, variables, and attributes types; template instance parameter types, and basic types of a `typedef` type.

How to examine only major structure differences

A manager for a project might want to examine at a high level the differences for a project, while a developer/engineer will want to see all the differences so that they can determine which elements should be merged. You can set the [SuppressRenamePropagatedDiffs](#) preference to specify whether propagated differences related to detected renaming should be filtered out.

Note

The **SuppressRenamePropagatedDiffs** preference is intended for comparison or reporting purposes. While this preference is active, to avoid any possible unintentional merge decisions and model corruption, all commands to merge units are disabled.

Comparison of propagated differences view and major structure differences view

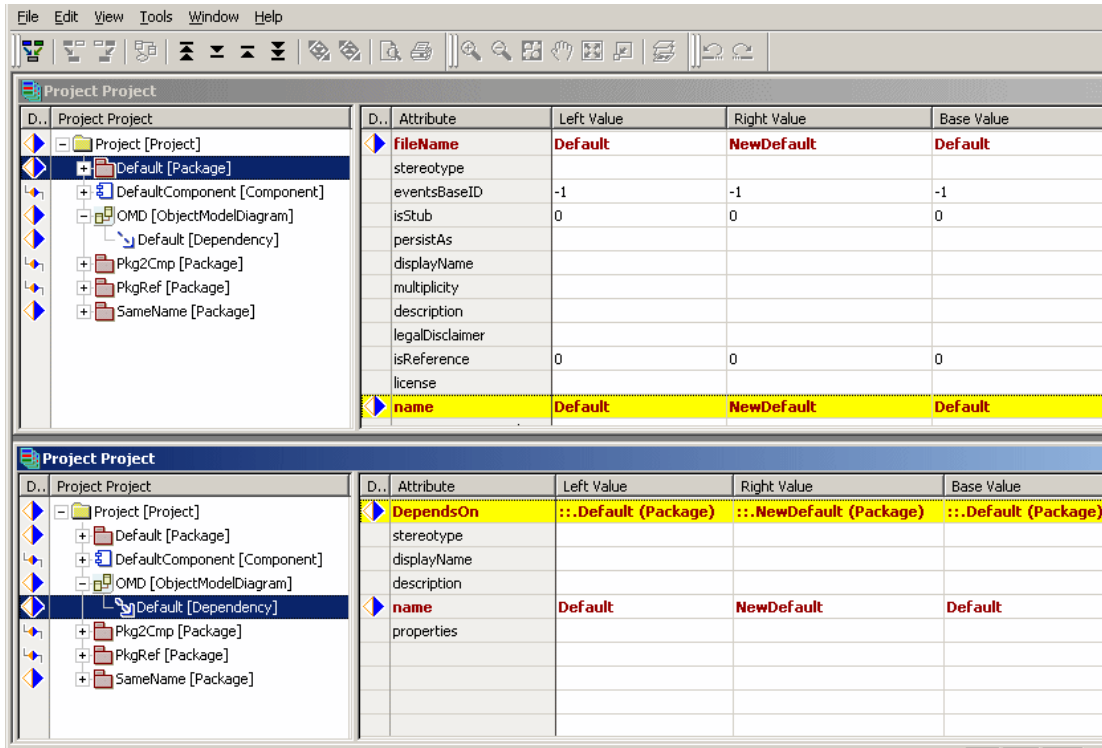
The illustrations in this topic compare the DiffMerge view of showing all propagated differences versus the view of showing only major structure differences.

Show all propagated differences

The following illustration shows all propagated differences. This means that the [SuppressRenamePropagatedDiffs](#) preference is turned off (the check box is cleared), which is the default. For example, you renamed a package. When you show all propagated differences, this means all elements related to the package will show as secondary differences. Therefore, in this example:

- ◆ The top DiffMerge browser in the following illustration shows that the **Default** package was renamed to **NewDefault**.

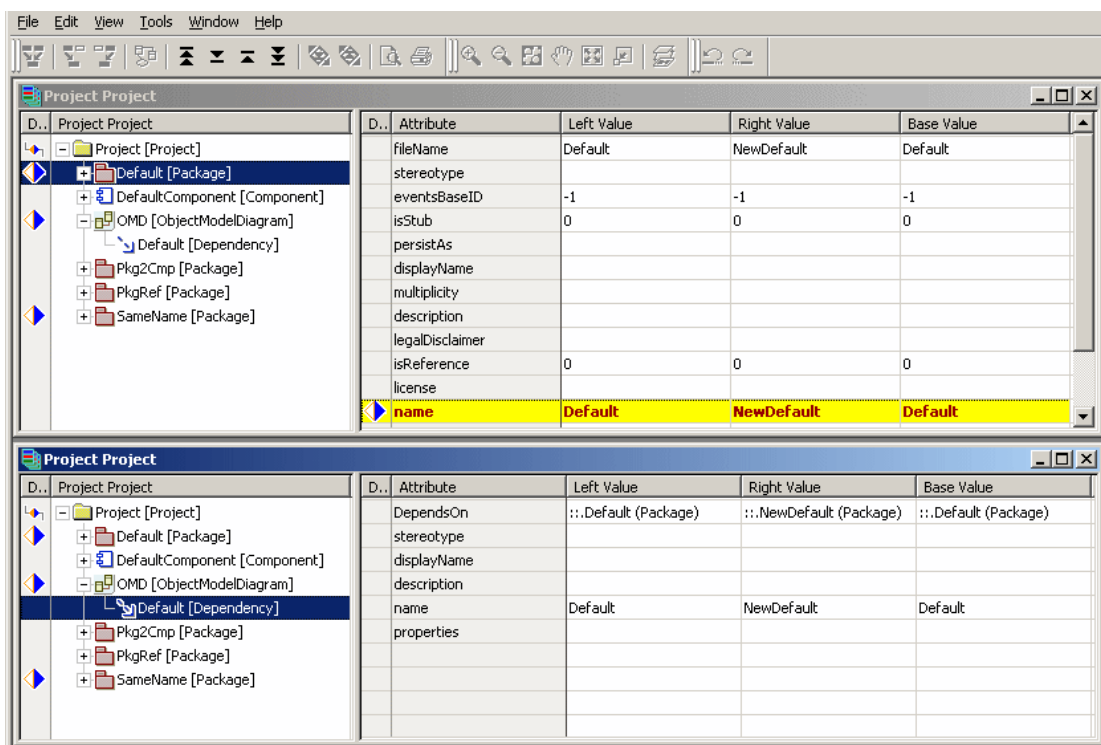
- ◆ The bottom DiffMerge browser shows that the object model diagram has a dependency to the **Default** package. Therefore, the DiffMerge tool reports a change for the dependency also.



Show only major differences

The following illustration shows only major structure differences. This means that the [SuppressRenamePropagatedDiffs](#) is turned on (the check box is selected), which means that all elements related to the package **will not** show as secondary differences. Therefore, in this illustration:

- ◆ The top DiffMerge browser shows that the **Default** package was renamed to **NewDefault**.
- ◆ The bottom DiffMerge browser shows that the object model diagram has a dependency to the **Default** package, but the DiffMerge tool **does not** report a change for the dependency. This is because when the **SuppressRenamePropagatedDiffs** setting is turned on, all secondary differences are suppressed.



Merge units with the DiffMerge tool

You can merge the two compared units into a third unit to create a new entity of the same type as the original two units or merge features from one unit into the other. By default, the initial merge (created when you start the merge) is taken from the unit on the left side. To take the initial merge from the right side, set the following flag in the General section of the `Diffmerge.ini` file:


```
"ReverseDiffOrder=TRUE"
```

Note

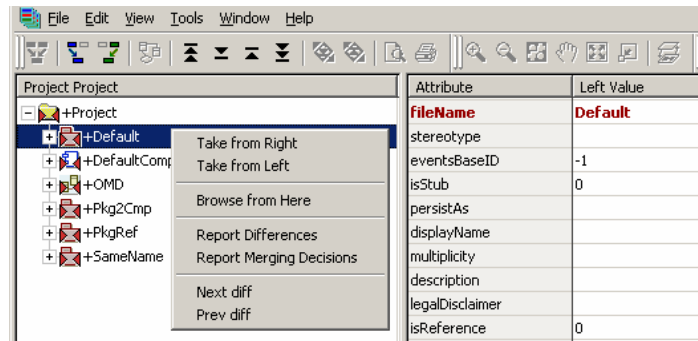
Before beginning the merge, you must also decide whether or not to include the graphical differences, as described in [Graphical differences suppression](#).

Starting a merge operation

To merge specific elements listed in the browser:

1. Choose **Edit > Start merge** or click **Start merge**  in the DiffMerge toolbar. The **Edit > Take from left**, **Edit > Take from right**, and **Edit > Take from base** arrows then become active in the DiffMerge toolbar, when applicable.
2. If you are working with a base-aware comparison ([For three units](#)), DiffMerge identifies all [Trivial Versus Non-trivial Differences](#) and displays a window asking if you want to merge all trivial differences automatically. You can select to not display this window again if you always handle trivial differences in the same manner.
3. Whether you are using a base-aware comparison or a comparison of two units, at this point you can navigate to a difference using the up and down keyboard arrows.
4. To restrict the view to only the conflicting items for either comparison, choose **View > View Conflicts**.

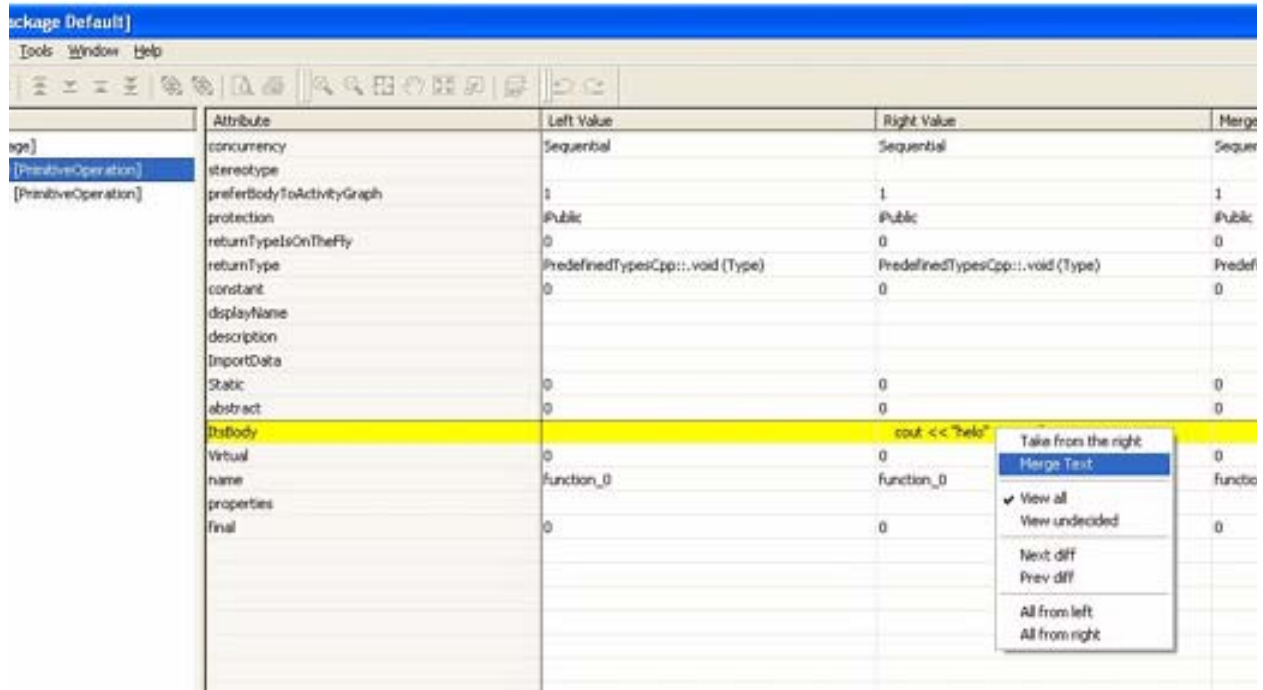
5. If you are comparing two or three units, for each element that exists on both sides of the comparison determine whether you want to perform these operations:
 - a. Take the element from the left or right along with its descendants (aggregates and associations which are nested elements in the browser).





- b. In addition, you can optionally take a value for a particular attribute for the element from the left or from the right (or from the base when performing a base-aware comparison). This might be done when you have chosen to take an element from the left in Step a, but for the value for a particular attribute, you prefer to take the right side.

Note that you might also follow this step for any descendants of the element you chose in Step a.


- c. Alternatively, if the particular attribute you want to change is a textual attribute, such as code and description, you could select **Merge Text** and the external textual DiffMerge tool (`tkdiff` by default) opens.



6. Use the **Edit > Take from Left** or **Take from right** options or click one of the following toolbar buttons to select a difference that you want to merge. Note that these buttons operate on the selected element, as well as all of its nested elements.

Button	Button Name	Explanation
	Take from left	Adds the element from the left unit to the right unit.
	Take from right	Adds the element from the right unit to the left unit.

7. If you are using a base-aware comparison for the merge, you can scroll through any non-trivial difference using the following toolbar icons when you want to make a merge decision.

 Next non-trivial difference

 Previous non-trivial difference

8. If you want to include or exclude an element manually, right-click the element select **Include from merge** or **Exclude from merge**.

9. Once a difference is resolved for either type of comparison, the difference arrow or arrows on the browser icon turn gray. Therefore, all gray items are not going to be displayed in an [Undecided view](#) of the merged elements.

Before Merge:



After Merge:



Note

You cannot exclude from the merge any elements that are in both the left and the right units. See [Rules for merging from a two-unit comparison](#).

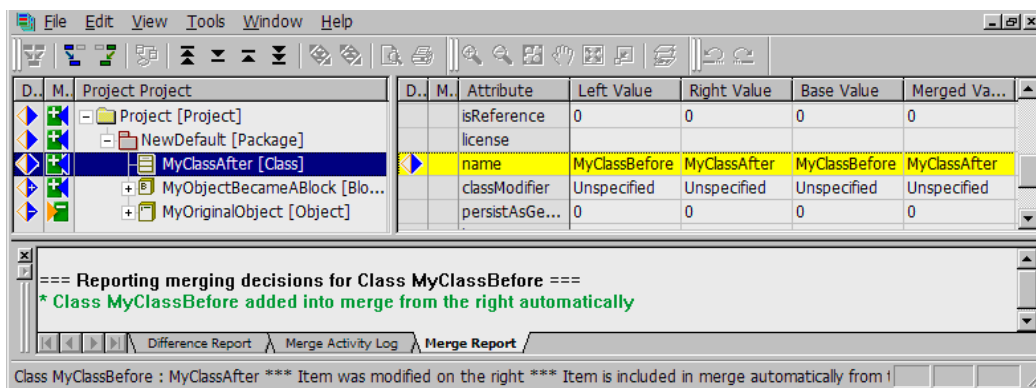
Merge renamed elements

In the DiffMerge browser, the DiffMerge tool displays element names from the left (column) comparison side by default. The left side has more priority than the right side.

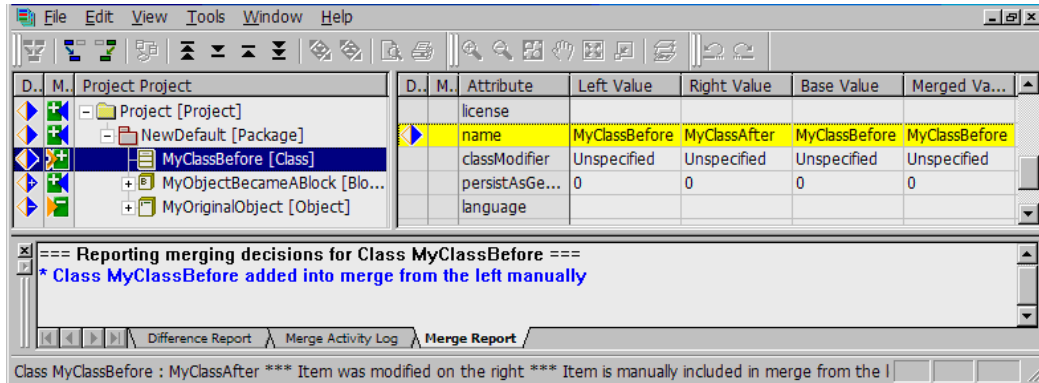
However, when switching to merging mode, DiffMerge displays actual names (meaning the names from the chosen side either by a user or automatically).

When a name difference is trivial, the DiffMerge tool will merge elements automatically. See [Trivial Versus Non-trivial Differences](#).

For example, in the case of the modification of **MyClassBefore** to **MyClassAfter**, the new class name, **MyClassAfter** is taken as the merge result. Note that in the merging mode, DiffMerge displays the actual element names. So in this example, **MyClassAfter** instead of **MyClassBefore** (that was displayed in comparison mode), as shown in the following figure:



If you decide to take another name value, the DiffMerge browser refreshes accordingly, as shown in the following figure:



Saving the merged unit

To save the merged unit, choose **File > Save Merge As**. If you started DiffMerge from within Rational Rhapsody, select Merge to Rational Rhapsody. The new, merged unit is saved as a separate file for use in the existing project or another project.

Merge units limitations

Note these limitations for merge units:

- ◆ After a graphical merge, referenced model elements that are not accessible are displayed as “unresolved” in the resulting sequence diagram. For example, model elements that are outside of Rational Rhapsody but referred to by diagram elements inside Rational Rhapsody refer are displayed as “unresolved”).
- ◆ Model elements cannot be edited in diagram graphical merge mode.
- ◆ Many of the model diagrams are displayed in read-only mode with the exception of merged statecharts, activity diagrams, and sequence diagrams. Therefore, some Rational Rhapsody functionality is not available in read-only diagram views, for example:
 - Movement
 - Changing the element properties
 - Opening referenced diagrams

Automatic merging for base-aware comparisons

If two units are being compared with a base unit, this base-aware comparison makes it possible for the DiffMerge tool to determine automatically the need for some merges using the concept of [Trivial Versus Non-trivial Differences](#).

This three-unit comparison includes detecting differences in the model elements, such as new class that was added or removed. It also locates textual differences in the attribute values of the model elements, such as change in the class description.

All the differences identified are compared to determine the trivial versus non-trivial differences. During an automatic merge operation, all of the trivial differences are automatically accepted for merging.

Trivial Versus Non-trivial Differences

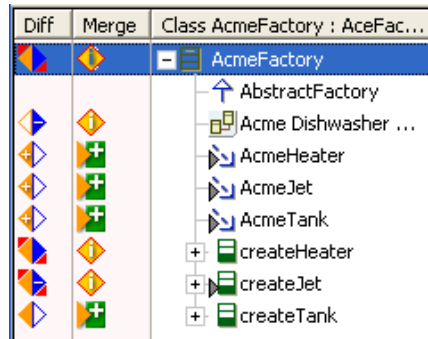
For a difference in which only one unit differs from the base unit, it is identified as a non-conflicting difference or *trivial difference*. Similarly, if both of the units are different from the base contributor but the differences are the same, then it is also a trivial difference. This applies to differences in model elements or differences between the attributes of the model elements.

However, if both units differ from the base unit, this is a *non-trivial or conflicting difference* that must be resolved by the developer manually.

Merge icons for base-aware comparisons

To understand the steps required to merge elements, see [Starting a merge operation](#).

After an element is merged in a base-aware comparison, the DiffMerge tool marks the merged item in a new Merge column, as shown in the following figure:









Note


The icon in the browser displays a gray arrow if the element has already been merged.



To see a description of each merge, position the mouse over a merge icon and read the description that displays at the bottom of the DiffMerge window, as shown in the following figure:



Dependency AcmeTank *** Item was added on the left *** Item is included in merge automatically from the left side


The Merge icons are used in both the browser and Attributes pane.


-  means an undecided element
-  means both elements were automatically merged (usually in an external textual DiffMerge tool)
-  means that based on a developer's decision, this element was manually merged from both the right and left
-  means the element was merged automatically from the right
-  means the element was merged automatically from the left
-  means the right-side element was automatically excluded from the merge

 means the left-side element was automatically excluded from the merge

 means that based on a developer's decision, this element was manually merged from the right.
Compare with .

 means that based on a developer's decision, this element was manually merged from the left.
Compare with .

 means that the left-side element was manually excluded from the merge

 means the right-side element was manually excluded from the merge

About making merge decisions

After [Starting a merge operation](#), developers might use the automatic merging capabilities to resolve trivial, non-conflicting differences. However, developers often must make some decisions about what needs to be merged and how it should be merged. The following tools can be used to help make decisions and manually merge the non-trivial differences:

- ◆ DiffMerge merging navigation and views
- ◆ Merge Report
- ◆ External text editors
- ◆ From Rational ClearCase launch DiffMerge

If a base-aware comparison is being used, the DiffMerge tool has more information about the left and right units. It is likely that the developer might need to make fewer manual merges with a three-unit (base-aware) comparison than with a two-unit comparison.

Navigation menu options for merging

The type of comparison (two units or three) being used affects the items displayed on this menu. If two units were used for the comparison, these menu options have the following uses:

- ◆ **Take from right** adds the element from the right unit to the left unit.
- ◆ **Take from left** adds the element from the left unit to the right unit
- ◆ **Browse from here** repositions the browser view to start with the select element. To switch back to the original view, choose **View > New Diff Browser**.
- ◆ **Next Diff** highlights the next difference in the browser.
- ◆ **Previous Diff** highlights the previous difference in the browser.

If three units (base-aware) were used for the comparison, these additional options are available:

- ◆ **Next Conflict** highlights the next non-trivial (conflicting) difference in the browser.
- ◆ **Previous Conflict** highlights the previous non-trivial (conflicting) difference in the browser.

Undecided view

To view differing elements that are neither in nor out of the merge for either type of comparison, the **View > View undecided** option shows only the remaining, unmerged differences. In other words, those differences that are not yet “in” or “out” of the merge. Units are not displayed that you have either explicitly marked as “in” or “out” of the merge or are always implicitly in the merge.

View in merge

To view only those elements that are currently in the merge with either type of comparison, the **View > View in merge** option displays only those elements that are currently in the merge (without recognition of the method used to include the element in the merge) are displayed.

Base-aware comparison merging

When using the base-aware comparison as the basis for a merge operation, the DiffMerge tool shows the status of all the identified changes with respect to the base version of your CM tool. This additional information for the merging task avoids running more diff operations to pinpoint the source of differences.

When using the automatic merge feature in a base-aware comparison, the developer only needs to define the merging required for the non-trivial differences. See [Automatic merging for base-aware comparisons](#) for additional information about the merging capabilities in base-aware comparisons. See [Trivial Versus Non-trivial Differences](#) for an explanation of these concepts.

Rules for merging from a two-unit comparison

When creating a merged unit from a two-unit comparison, the DiffMerge tool uses these rules:

- ◆ All identical elements are always included in the merge; they cannot be removed.
- ◆ All elements appearing in both sides must be in the merge, with attributes either from the left or from the right, or with some from the left and some from the right. The element itself must always be included.
- ◆ Elements appearing in only the left or the right side can be either in or out of the merge.


Examine the icons in the browser to identify the elements that are going to be automatically included in the merge, and those that are not.

DiffMerge inserts a small plus (+) or minus (-) sign in front of an element name to indicate the following meanings:

- ◆ Plus (+) means the element will be included in the merge, for example:

 +HomeAlarm

- ◆ Minus (-) means the element will not be included, for example:

 -Host

Merging diagrams graphically for most diagrams

You can use the DiffMerge tool to merge graphical elements in any of the Rational Rhapsody diagrams. This procedure tells you how to merge diagrams graphically for most diagrams in Rational Rhapsody, except statecharts and activity diagrams. To do this for statecharts and activity diagrams, see [Merging diagrams graphically for statecharts and activity diagrams](#).

You can copy an element in one diagram that you would like to have in another diagram. You can copy the graphic elements from the left or right window and place them in the merge window, or create new elements in the merge window and move or size them as needed.

To merge existing graphic elements in the DiffMerge interface:

Note

While this procedure shows how to merge graphic elements in an object model diagram, you can use the same method to merge elements for the other Rational Rhapsody diagrams (except statecharts and activity diagrams).

1. Optionally, right-click the object model diagram in the DiffMerge browser and select **View All Diagrams** to review the diagrams to help you decide from which side you want to take your changes.
2. Right-click the object model diagram and select **Take from Right** or **Take from Left**. See [Making merge decisions from the graphical view](#).
3. Right-click the object model diagram and select **Merge Graphically**. Rational Rhapsody opens windows to show all versions of the object model diagram, including the newly created merged object model diagram in the Merge window.
4. Review the merged object model diagram in the Merge window.
5. Edit the merged object model diagram, if necessary.
 - Right-click an element in the merged diagram and use the options on the pop-up menu; for example, **Take from Left**, **Take from Right**, **Exclude from Merge**.
 - You can move and delete elements. In addition, you can modify a textual element through its Features window.
6. When you have completed the merge operation, save the merged file as described in [Saving the graphically merged unit](#).

Merging diagrams graphically for statecharts and activity diagrams

This procedure tells you how to merge diagrams graphically for statecharts and activity diagrams. To do this for the other Rational Rhapsody diagrams (such as object model diagrams and structure diagrams), see [Merging diagrams graphically for most diagrams](#).

Note

Use this approach when you have an element in one diagram that you would like to have in another diagram.

You can copy the graphical elements from the left or right window and place them in the Merge window, or create new elements in the Merge window and move or size them as needed.

To merge existing graphic elements in a statechart or activity diagram in the DiffMerge interface:

Note

While this procedure shows how to merge graphic elements in a statechart, you can use the same method to merge elements for activity diagrams. For additional information, see [Tips for graphical merging for statecharts and activity diagrams](#).

1. Select the statechart in the DiffMerge browser, right-click and select **Merge Graphically** to open the diagrams.
2. Select one or more elements to copy, and then press **Ctrl+C** (or choose **Edit > Copy**).
3. Click in the Merge window (You can create a new statechart or merge elements into one of the original statecharts).
4. Press **Ctrl+V** (or choose **Edit > Paste**) to place the elements. Use your mouse to drag-and-drop one or more new elements to move them to the appropriate position within the merged diagram.
5. Drag an element to position it precisely. In the case of statecharts, you can copy:
 - ◆ Any state or connector.
 - ◆ Any group of elements (including states, transitions, and connectors), provided that, if it includes a transition, it also includes its source and target.
 - ◆ A label of a transition. This is the only way to copy or create a transition.
6. Note that you can copy a transition only with its source and target.
7. Repeat Steps 1–5 to merge additional elements.
8. Edit the merged statechart if necessary. For example, you can draw new elements using the toolbar icons and you can click anywhere in the window to unselect a new element.

9. When you have completed the merge operation, save the merged file as described in [Saving the graphically merged unit](#).

Tips for graphical merging for statecharts and activity diagrams

To locate the original transition and select its label, you can:

- ◆ Use **Ctrl+C** to copy the label.
- ◆ Draw the new transition in the merge window.
- ◆ Use **Ctrl+V** to paste in the new label.

To view any elements not visible within a new (active) statechart merge window, either maximize the window or choose **View > Zoom to Fit** while the window is active.

If necessary, modify textual elements of graphic items using their Features windows.

If appropriate, work on any nested statecharts recursively.

Saving the graphically merged unit

To save the graphically merged file, choose **File > Save Merge As**.

Keep in mind:

- ◆ If you started the merge from within Rational Rhapsody, select **Merge to Rhapsody**. The new, merged unit is saved as a separate file for use in the existing project or another project.
- ◆ In some cases, there might be no **Save Merge As**. There might just be a **Save** if the filename and its location was defined when DiffMerge was launched (for example, from Rational ClearCase merge manager, or Rational Team Concert merge process in Eclipse).

About merging sequence diagrams

DiffMerge allows you to merge differing versions of sequence diagrams that you have compared with DiffMerge. This is possible for both two-way comparisons and three-way (base-aware) comparisons.

The merge ability is a manual merge process. There is no automatic merging of differences when merging sequence diagrams.

Note

The operation mode of the sequence diagram (analysis or design) does not affect the merging of sequence diagrams.

When merging sequence diagrams, you can make merge decisions directly from the DiffMerge browser (structural merging), or, alternatively, you can display the various versions of the diagram and make your merge decisions from within the diagrams (graphical merging).

Note

When merging sequence diagrams, you can use structural or graphical merging, or both. However, once you begin using graphical merging, you cannot return to structural merging unless you “reset” the diagram by including one of the versions of the diagram in its entirety. In addition, structural merging of diagram elements (for example, instance lines, messages, and so forth) is only available for sequence diagrams.

When making merge decisions, whether in “structural” mode or “graphical” mode, the following principles apply:

- ◆ when an element exists in only one of the versions (“left” or “right”), you can choose to include it in the merge or exclude it from the merge.
- ◆ when an element exists in both the “left” and “right” version, but it differs in the two versions, you can choose to include either the “left” or “right” version in the merge.

Making merge decisions from the DiffMerge browser

To merge diagram elements from within the DiffMerge browser:

1. In the browser, right-click the relevant element.
2. Depending on your situation:
 - ◆ For elements that exist in both versions, select **Take from Left** or **Take from Right**.
 - ◆ For elements that exist in only one of the versions, select **Include in Merge** or **Exclude from Merge**.

Making merge decisions from the graphical view

To merge elements from the graphical view:

1. In the browser, right-click the relevant diagram, and select **View All Diagrams**.
2. Right-click the relevant element in the diagram, and depending on your situation:
 - ◆ For elements that exist in both versions, select **Take from Left** or **Take from Right**.
 - ◆ For elements that exist in only one of the versions, select **Include in Merge** or **Exclude from Merge**.

Additional changes permitted in graphical merge mode

When working in graphical mode, in addition to making merge decisions:

- ◆ You can move diagram elements on the “merged” version of the diagram that is displayed.
- ◆ You can delete diagram elements from the “merged” version of the diagram that is displayed.
- ◆ You can change element attributes such as arguments.

However, you cannot create new elements to add to the diagram.

About “referring” elements

Sequence diagrams contain certain elements that are not dependent on the presence of any other element, for example, classifier roles. Sequence diagrams also contain elements that require the presence of other elements, for example, messages, which require the presence of the sending and receiving elements. Dependent elements such as these are referred to here as “referring” elements.

When you are making merge decisions, referring elements can be included in the merge only if all of the elements they refer to are included in the merge. For example, a message can be included in the merge only if the sending and receiving classifier roles are included in the merge.

If you choose to exclude an element which other elements require, then these dependent elements will be excluded from the merge as well.

Note

If a menu command to include an element in a merge is disabled on a pop-up menu, be sure that you have not previously chosen to exclude an element upon which the selected element depends.

In general, DiffMerge does not check whether the diagram resulting from a merge is correct. It is the user's responsibility to ensure that merge decisions result in a correct sequence diagram.

Elements that realize Sequence diagram elements

Merge decisions regarding instance lines in a diagram do not affect the inclusion/exclusion of the class that realizes the instance line, if one is specified.

Merge decisions regarding messages in a diagram do not affect the inclusion/exclusion of the event that realizes the message, if one is specified.

Merge activity log

DiffMerge writes a textual log of a merge operation to the **Merge Activity Log** tab of the Output window, based on the current preference settings.

See [MergeLog preferences category](#) for the list of preferences and their default values.

Producing merge reports

After some or all of the elements are merged, you can produce a Merge Report to see what you have merged and any conflicting elements that still remain.

To produce a Merge Report:

1. If you want to have a report on a specific element, highlight it.
2. Choose **Tools > Report Merging Decisions > All** to select the whole project or **Tools > Report Merging Decisions > Selected** for only the element selected in Step 1. The Merge Report is displayed in the Output window.

Note

You can also create a Merge Report by right-clicking the element in the browser and selecting **Report Merging Decisions**.

DiffMerge writes the Merge Report based on the current preference settings. See [MergeLog preferences category](#) for information about these settings.

DiffMerge tool preferences

To change the default characteristics of the DiffMerge tool, use the Preferences window (choose **View > Preferences**). The preferences settings are stored in the user's `.ini` file.

Note

While the Preferences window for the DiffMerge tool looks similar to the Properties window in Rational Rhapsody, they are different. Changing your DiffMerge preferences does not affect Rational Rhapsody units in any way.

Use the Preferences window to control the following categories of settings:

- ◆ **Colors** controls the colors used in by the DiffMerge tool. For more information, see [Colors preferences category](#).
- ◆ **DiagramViews** contains only the **BlinkWalkingThroughDiffs** preference that switches the graphical walk-through blinking feature on or off. For more information, see [Switching off element blinking](#).
- ◆ **DiffReport** controls the appearance of the log of the results of a diff operation. For more information, see [DiffReport preferences category](#).
- ◆ **General** controls the general aspects and default behavior of the DiffMerge tool.
- ◆ **MergeLog** control the appearance of the log of the results of a merge operation. For more information, see [MergeLog preferences category](#).
- ◆ **Suppressions** specifies which items should be ignored by the DiffMerge tool. For more information, see [Suppressions preferences category](#).
- ◆ **TextDiffMerge** control textual DiffMerge operations. For more information, see [TextDiffMerge preferences category](#).

Changing preferences

To change the value of a preference:

1. Open the Preferences window. Choose **View > Preferences**.
2. Click the + sign next to a category to expand its list.
3. Click the box to the right of the preference you want to change.
4. Type a new value, or select or clear a control, as applicable.
5. Click **OK**.

Restoring default settings

To revert to the default values of the preferences:

1. On the Preferences window, click **Restore Defaults**.
2. When asked to confirm your requested action, click **Yes**.

Keywords

Note that some of the default values of the preferences include the following keywords:

- ◆ `$diffs` means the number of differences found
- ◆ `$elemname` means the name of the model element
- ◆ `$elementype` means the model element type (class, component, and so on)
- ◆ `$itemname` means the name of the attribute
- ◆ `$leftonly` means the number of elements that exist only on the left side of the comparison
- ◆ `$parentname` means the name of the parent
- ◆ `$parenttype` means the parent type (class, component, and so on)
- ◆ `$rightonly` means the number of elements that exist only on the right side
- ◆ `$BaseAwareDiffInvocation` means the command to launch the external textual three-unit comparison in the DiffMerge tool (used in the [TextDiffMerge preferences category](#) preferences)
- ◆ `$DiffInvocation` means the command to launch the external textual two-unit comparison in the DiffMerge tool (used in the [TextDiffMerge preferences category](#) preferences)
- ◆ `$sourceBase` means the text file containing compared value from the base (used in the [TextDiffMerge preferences category](#) preferences)

Colors preferences category

The **Colors** preferences category on the Preferences window enables you to change the default colors used by the DiffMerge tool. To learn how to change preferences, see [Changing preferences](#).

The following table lists the possible settings and their default values.

Preference Name	Description	Default Value
DiffColor	Specifies the color used to print the items with differences, highlight graphical differences, and print the merge activity.	RGB (255, 0, 0)
LeftOnlyColor	Specifies the color used to print the items in the left-only category, highlight graphical differences, and print the merge activity.	RGB (0, 0, 255)
NestedDiffColor	Specifies the color used to print the items with nested differences.	RGB (153 153, 153)
NoDiffColor	Specifies the color used to print the "no differences" category for an item and highlight graphical differences.	RGB (0, 0, 0)
RightOnlyColor	Specifies the color used to print the items in the right-only category, highlight graphical differences, and print the merge activity.	RGB (0, 0, 0)
ReportFooterColor	Specifies the color used in the report footer.	RGB (0, 0, 0)
ReportHeaderColor	Specifies the color used in the report header.	RGB (0, 128, 64)
UseDefault	Specifies whether to use the default color or not.	Checked

DiffReport preferences category

The **DiffReport** preferences category enables you to control the appearance of the textual report on the differences found by the DiffMerge tool. To learn how to change preferences, see [Changing preferences](#).

The following table lists the possible settings and their default values.

Preference Name	Description	Default Value
BaseAwareDiffAttrChanged	Specifies that the attribute was modified on one side.	: * Attribute "\$itemname" changed \$side (\$triviality)
BaseAwareDiffAttrChngBoth	Specifies that the attribute was modified on both side.	: # Attribute "\$itemname" changed on both sides (\$triviality)
BaseAwareDiffAttrDelAndChng	Specifies that the attribute was deleted on one side and changed on the other side.	: % Attribute "\$itemname" deleted \$side and changed on the other side (\$triviality)
BaseAwareDiffElemAdded	Specifies that the model element was added	+ \$elemtype \$elemname added \$side (\$triviality)

Preference Name	Description	Default Value
BaseAwareDiffElemChanged	Specifies that the model element was changed on one side	* \$elemtype \$elemname changed \$side (\$triviality)
BaseAwareDiffElemChngBoth	Specifies that the model element was modified on both sides.	# \$elemtype \$elemname changed on both sides (\$triviality)
BaseAwareDiffElemDelAndChng	Specifies that the model element was deleted on one side and changed on the other side.	% \$elemtype \$elemname deleted \$side and changed on the other side (\$triviality)
BaseAwareDiffElemDeleted	Specifies that the model element was deleted	- \$elemtype \$elemname deleted \$side (\$triviality)
BaseAwareDiffMergeAutoNo	Specifies the string indicating that the compared element cannot be merged automatically.	cannot be merged automatically
BaseAwareDiffMergeAutoYes	Specifies the string indicating that the compared elements can be merged automatically.	can be merged automatically
BaseAwareDiffReportFooter	Specifies the footer of base-aware Difference Report.	--- Found \$diffs elements with differences, \$leftonly elements exists only on left side, \$rightonly elements exists only on right side ---
BaseAwareDiffReportHeader	Specifies the header of base-aware Difference Report.	=== Reporting differences for \$elemtype \$elemname (\$mergeauto) ===
BaseAwareDiffSideLeft	Specifies the string indicating the left side of comparison.	on the left
BaseAwareDiffSideRight	Specifies the string indicating the right side of comparison.	on the right
BaseAwareDiffTrivialNo	Specifies the string indicating the n-n-trivial (conflict) difference of the element.	non-trivial diff
BaseAwareDiffTrivialYes	Specifies the string indicating the trivial (non-conflict) difference of the element.	trivial diff
DiffPrefix	Describes the elements with differences	<> Differences found for \$elemtype \$elemname
LeftOnlyPrefix	Describes the elements that exist only on the left side of the comparison	< \$elemtype \$elemname exists only on left side
NestedDiffPrefix	Describes the elements with nested differences	>> \$elemtype \$elemname:
NestedElementPrefix	Specifies the text to be appended to each nested element	a tab character
NoDiffPrefix	Describes the elements with no differences	= No differences found for \$elemtype \$elemname

Preference Name	Description	Default Value
PrintLineNumbers	Determines whether line numbers are printed in the report	Cleared (do not include)
PrintNoDiffLines	Determines whether to report elements without differences	Cleared (do not report)
PrintSubDiffs	Determines whether to report subdifferences	Checked (report subdifferences)
ReportFooter	Specifies the text to be appended to the end of each report	Found \$diffs elements with differences, \$leftonly elements exists only on left side, \$rightonly elements exists only on right side
ReportHeader	Specifies the text to be appended to the beginning of each report	===Reporting differences for \$elemtype \$elemname===
RightOnlyPrefix	Describes the elements that exist only on the right side of the comparison	> \$elemtype \$elemname exists only on right side

General preferences category

The **General** preferences category controls the general aspects and default behavior of the DiffMerge tool. To learn how to change preferences, see [Changing preferences](#).

Note

The **RepPLUScmdline**, **ReporterPLUSPath**, and **ReporterPLUSTemplateDir** preferences are for internal use only by Rational Rhapsody staff. Any changes to these preferences should only be done by or under the direction of the Rational Rhapsody staff.

ElementMatchRule

The **ElementMatchRule** preference specifies which rule DiffMerge will apply to match the elements within the units being compared.

Possible values:

- ◆ **Default** (default)
- ◆ **Without Renaming Support**

Select **Default** if you want DiffMerge to try to match elements by ID if there are not matches by name. Note that this means that the DiffMerge tool will be able to detect and report if an element has been renamed and has a different name on each comparison side.

Select **Without Renaming Support** if you want DiffMerge to match elements only by name. Note that this means that two of the same elements in the same units (in the two Rational Rhapsody projects you are comparing) will never be matched because they have different names.

ResolveAutomaticallyWhenStartingMerge

The **ResolveAutomaticallyWhenStartingMerge** preference specifies whether DiffMerge should or should not resolve all trivial differences when performing a merge in a base-aware comparison.

The `Do you want to automatically merge trivial differences?` message appears every time until a user selects **Use my current reply as default**. After that, DiffMerge will or will not resolve differences automatically without asking a user, for example, not displaying that message any more.

Possible values:

- ◆ **Ask** (default)
- ◆ **Yes**
- ◆ **No**

ShowDMMarksInBaseAwareMode

The **ShowDMMarksInBaseAwareMode** preference specifies what DiffMerge should or should not display the legacy difference and merging state marks when performing a base-aware comparison.

Default Value: `Cleared`

MergeLog preferences category

The **MergeLog** preferences category enables you to control the appearance of the [Merge activity log](#) and [Producing merge reports](#) of the results from a merge operation. To learn how to change preferences, see [Changing preferences](#).

The following table lists the possible settings and their default values.

Preference Name	Description	Default Value
AllLeftItemMerge	Specifies all the items from the left side to add to the merge	\$elemtype "\$elemtype": All items from left added to merge
AllRightItemMerge	Specifies all the items from the right side to add to the merge	\$elemtype "\$elemtype": All items from right added to merge
ExcludeFromMerge	Specifies the model elements that will not be included in the merge	\$elemtype "\$elemname" removed from merge
GraphicalMerge	Specifies that a statechart or activity diagram was graphically merged	\$elemtype of "\$parenttype \$parentname" merged graphically
IncludeInMerge	Specifies the model element to add to the merge	\$elemtype "\$elemname" added to merge
ItemMerge	Specifies that the item has been merged	\$elemtype "\$elemname": Item "\$itemname" merged
LeftItemMerge	Specifies that the item from the left was added to the merge	\$elemtype "\$elemtype": Item "\$itemname" from left added to merge
LeftMerge	Specifies that the model elements from the left were added to the merge	\$elemtype "\$elemname" from left added to merge.
MergeToRhapsody	Specifies that the current merge has been added to Rational Rhapsody	\$elemtype "\$elemname" merged with Rational Rhapsody
RepDecidedAuto	Specifies the string indicating the automatically resolved element.	automatically
RepDecidedMan	Specifies the string indicating the manually resolved element.	manually
RepElemExcluded	Specifies that the model element is excluded from merge.	- \$elemtype \$elemname excluded from merge \$decided
RepElemIncluded	Specifies that the model element is included into merge.	+ \$elemtype \$elemname included into merge \$side \$decided
RepElemMerged	Specifies that the model element existing on both sides is merged (a user took some part from the left, some part from the right, or merged attribute values by external textual DiffMerge tool).	* \$elemtype \$elemname merged \$decided
RepElemTakenFrom	Specifies that the model element existing on both sides is included into merge from particular side.	* \$elemtype \$elemname added into merge \$side \$decided
RepElemUndecided	Specifies that the model element existing on both sides is still undecided.	# \$elemtype \$elemname is undecided

Preference Name	Description	Default Value
RepFooter	Specifies the footer of the Merge Report.	
RepHeader	Specifies the header of the Merge Report.	=== Reporting merging decisions for \$elemtype \$elemname ===
ReplItemDecided	Specifies that the item (the attribute) is manually included into merge from particular side whereas its model element is taken into merge from the other side.	: Attribute \"\$itemname\" added into merge \$side manually
ReplItemMerged	Specifies that merged value of the item (the attribute) is manually edited by user.	: Attribute \"\$itemname\" merged manually
RepSideLeft	Specifies the string indicating the left side of comparison.	from the left
RepSideRight	Specifies the string indicating the right side of comparison.	from the right
RightItemMerge	Specifies that the item from the right was added to the merge	\$elemtype \"\$elemtype\": Item \"\$itemname\" from right added to merge
RightMerge	Specifies that the model elements from the left were added to the merge	\$elemtype \"\$elemname\" from right added to merge
SaveMerge	Specifies that the merge has been saved	Saved merged \$elemtype \"\$elemname\" to \$filename
StartMerge	Specifies that the merge has started	Started merge for \$elemtype \"\$elemname\"

Suppressions preferences category

The **Suppressions** preferences category enables you to specify which items should be ignored by the DiffMerge tool. To learn how to change preferences, see [Changing preferences](#).

DiffAttributesFilter

Specifies a comma-separated list of attributes that should be ignored by the DiffMerge tool. For example, if this value is `id,name` the DiffMerge tool ignores the differences in the ID and name of objects.

Default value:

```
id, lastID, ImportData, cmheader,
state, RequirementTracabilityHandle,
isSaveUnit, isUR, isNameGenerated,
isReadOnly, errorStatus, version, baseVersion,
defNumber, directoryName, CPUtype, icon,
isTemplate, typeID, stereotypeID, DependsOnID,
DependsOnImportData
```


ExcludeGraphTypesVLess6

Specifies a comma-separated list of Rational Rhapsody classes that should be ignored by DiffMerge tool when comparing diagrams from Rational Rhapsody version older than 6.0 (to provide DiffMerge backward compatibility).

Default value:

`CGIMessageLabel,CGIFreeText`

IgnoreGraphDiffs

Specifies the initial value of the **Ignore Graphical Differences** option.

Default value: `Checked`

ShowMetaInfoInBrowser

Determines whether meta information is displayed in the DiffMerge browser. This preference turns on/off displaying element types (metaclasses) in the right part of the DiffMerge browser.

Default value: `Cleared`

ShowStereotypeInBrowser

Determines whether element stereotypes are displayed in the DiffMerge browser.

Default value: `Cleared`

SuppressRenamePropagatedDiffs

In cases when the DiffMerge tool detects that there are Rational Rhapsody element name changes, the **SuppressRenamePropagatedDiffs** preference specifies whether propagated differences related to the detected name changes should be filtered out.

Default value: `Cleared` (meaning this preference is turned off)

When turned on (meaning the check box is selected), the DiffMerge tool suppresses all attribute differences that correspond to detected name changes.

Because of the purpose of this preference, this preference value is *disregarded* when DiffMerge is launched:

- ◆ From the command line when you use the `merge`, `xmerge`, `compare`, or `xcompare` commands
- ◆ From the Configuration Management window in Rational Rhapsody when you try to compare and/or merge unit versions
- ◆ From an integrated CM tool (Rational Synergy, Rational ClearCase, and so forth)

Note

This use of the **SuppressRenamePropogatedDiffs** preference is for comparison and reporting purposes only. There is no merging allowed when this preference is turned on to avoid unintentional merging and corrupting a model.

TextDiffMerge preferences category

The **TextDiffMerge** preferences category controls the appearance of textual DiffMerge operations, including the Rational ClearCase ClearDiff and ClearDiffMrg tools. Rational Rhapsody searches for the preferred external, textual diff/merge tool in this sequence:

- ◆ Choose **View > Preferences** for any changes in the `DiffMerge.ini` file
- ◆ Current source control management tool metaclass
- ◆ Under `DiffMerge::TextDiffMerge` in the preferences file

The following table lists the **TextDiffMerge** preference settings and their default values.

Preference Name	Description	Default Value
BaseAwareAutoMergeInvocation	Specifies how to launch the external textual DiffMerge tool supporting base-aware (three-unit) detection of triviality of textual difference and three-unit automatic merging.	
BaseAwareAutoMergeableAttributes	Specifies a list of attribute names, which contain text values and are allowed to merge automatically. The value of attribute can be also <code>All</code> (meaning all attribute can be textually merged) and <code>None</code> (meaning no automatic textual merge should be done). The default value, <code>ItsBody</code> , determines that only the implementation for the methods or/and operations will be merged automatically when possible.	<code>ItsBody</code>
BaseAwareDiffInvocation	Specifies how to launch the external textual DiffMerge tool supporting a base-aware comparison and merging in Base Aware Diff mode.	<code>\$OMROOT\etc\tkdiff.exe \$source1 \$source2 -a \$sourceBase</code>
BaseAwareDiffMergeInvocation	Specifies how to launch the external textual DiffMerge tool supporting base-aware comparison and merging in Base Aware Merge mode.	<code>\$BaseAwareDiffInvocation -o \$output</code>
BaseAwareTextDiffMergeEnabled	Determines whether a base-aware (three-unit) textual DiffMerge tool is available to be launched.	<code>Cleared</code> (check box cleared)

Preference Name	Description	Default Value
DiffInvocation	Specifies how to launch the external textual DiffMerge tool in Diff mode. For example, if the value is <code>tkdiff \$source1 \$source2</code> , DiffMerge calls the TkDiff tool.	<code>\$OMROOT\etc\tkdiff.exe \$source1 \$source2</code> The keywords are as follows: <ul style="list-style-type: none"> • <code>\$OMROOT</code> means the location of the Share subdirectory under the Rational Rhapsody installation • <code>\$source1</code> means the text file containing left value of compared attribute • <code>\$source2</code> means the text file containing right value of compared attribute
DiffMergeInvocation	Specifies how to launch the external textual DiffMerge tool in the Merge mode. For example, if the value is <code>tkdiff \$source1 \$source2 -o \$output</code> , DiffMerge calls the TkDiff tool.	<code>\$DiffInvocation -o \$output</code> The keyword is <code>\$output</code> means the text file that the merged result will be written to (is replaced by value of the MergeOutput preference).
MergeOutput	Specifies the text file path and name that the merging result will be written to. For example, if the value is <code>c:\temp\out.txt</code> , DiffMerge looks for the merging result in the <code>out.txt</code> file in the <code>c:\temp</code> folder.	<code>\$temp\out.txt</code> The keyword is <code>\$temp</code> means the folder specified in the operational system to store temporary files.

Note

The metaclass `General::DiffMerge` and its preferences (**MergeOutput**, **DiffInvocation**, and **DiffMergeInvocation**) were removed in Version 4.1 of Rational Rhapsody. Therefore, if you previously overrode those properties in your `site.prp` file, Rational Rhapsody ignores them (they will have no effect) unless you move them under `DiffMerge::TextDiffMerge`.

Command-line options for the DiffMerge tool

The DiffMerge CLI (command line interface) supports all of the DiffMerge tool features including automatically accepting all trivial, non-conflicting differences for a merge operation (except where noted). The CLI can also be launched from [IBM Rational ClearCase](#) to perform the same tasks that can be performed using the DiffMerge interface, as described previously.

Developers use DiffMerge CLI to create batch files that launch the DiffMerge command-line interface and automate some of the tasks associated with software development, particularly scheduled nightly builds.

Launching the DiffMerge tool interface using the command line

To launch the DiffMerge tool interface using the command line:

1. Open a command-line prompt window.
2. Change the command-line path to the Rational Rhapsody installation folder.
3. Type `Diffmerge.exe` and press the **Enter** key to open the DiffMerge window.

This interface provides all of the features available for two-way and base-aware comparisons.

Launching the DiffMerge tool from the command line

If you do not want to work through the interface, you can run the DiffMerge tool completely from the command line:

1. Open a command-line prompt window.
2. Change to the Rational Rhapsody installation folder.
3. Type `Diffmerge.exe <options>` as listed in [DiffMerge command-line syntax options](#). The interface does not appear, unless you use the `-xcompare` or `-xmerge` option.

DiffMerge command-line syntax options

The Rational Rhapsody DiffMerge tool basic syntax is the `diffmerge.exe` command with the options described in the following table.

```
Diffmerge.exe <options>
```

For example, you might enter the following code to create a base-aware comparison/merge that includes the subunits (`-recursive`) and specifies the output file in `C:\Radio_Merge\Radio.rpy`:

```
Diffmerge.exe -merge -recursive
C:\Radio_Main\Radio.rpy
C:\Radio_Branch\Radio.rpy
-base C:\Radio_Base\Radio.rpy
-out C:\Radio_Merge\Radio.rpy
```

The following table lists the command-line options.

Option	Description	Syntax
<code>-base <filename></code>	Specifies the name of the file that is the common ancestor of the two compared files (<i>file1</i> and <i>file2</i> in the syntax) for a base-aware comparison, as described in For three units .	<code>Diffmerge.exe -base <base file name> <file1> <file2> -compare</code>
<code>-compare</code>	Starts the DiffMerge tool in Compare mode for two units, but does not display the DiffMerge interface. (To start a compare and display the interface, use the <code>-xcompare</code> command.) Graphical mode performs the comparison and then exits to the system prompt. The results of the comparison are as follows: 0 = identical Rational Rhapsody units 1 = differences between the two units were identified This result can be retrieved by the <code>ERRORLEVEL</code> MSDOS variable. For example, <code>echo Exit code = %ERRORLEVEL%</code>	<code>Diffmerge.exe <file1> <file2> -compare</code>
<code>-diffReport <Difference Report file></code>	Writes all the text in the Difference Report tab to the specified file. For more information, see Difference Report generation . Use only when using <code>-compare</code> or <code>-merge</code> , otherwise <code>-DiffReport</code> will not be executed.	<code>Diffmerge.exe -compare <file1> <file2> -diffReport <filename></code> or <code>Diffmerge.exe -merge <file1> <file2> -base <file0> -diffReport <filename></code>

Option	Description	Syntax
-merge	<p>Starts the DiffMerge tool in merge mode, but does not display the DiffMerge interface. (To start a merge and display the interface, use the -xmerge command.)</p> <p>If the tool detects a merge conflict, the merge action is stopped, and the tool returns a "1" exit code.</p> <p>This result can be retrieved by the ERRORLEVEL MSDOS variable. For example, echo Exit code = %ERRORLEVEL%</p> <p>If the merge can be completed automatically (without conflicts), the merged unit is saved using the file named in the -out command. If no -out command is specified, the Save window appears to allow the developer to enter a name for the new file.</p> <p>Note: The -merge command is applicable for base-aware mode only, therefore you must specify a base unit (see -base).</p>	<pre>Diffmerge.exe <file1> <file2> -base <file0> -merge</pre>
-mergeLog <Merge Activity Log file>	<p>Writes all the text in the Merge Activity Log tab to the specified file. For more information, see Merge activity log.</p> <p>If the file does not exist, DiffMerge creates the file. If it already exists, the new information is appended to the existing file.</p> <p>Use only when using -merge, otherwise -mergeLog will not be executed.</p>	<pre>Diffmerge.exe -merge <file1> <file2> -base <file0> -mergeLog <filename></pre>
-mergeReport <Merge Report file>	<p>Writes all the text in the Merge Report tab to the specified file. For more information, see Producing merge reports.</p> <p>If the file does not exist, DiffMerge creates the file. If it already exists, the new information is appended to the existing file.</p> <p>Use only when using -merge, otherwise -mergeReport will not be executed.</p>	<pre>Diffmerge.exe -merge <file1> <file2> -base <file0> -mergeReport <filename></pre>
-out <fileName>	<p>Replaces the Save merge as option with an option to save the merge results to the specified file.</p> <p>Use only when using <file1> and <file2> to merge two files.</p>	<pre>Diffmerge.exe <file1> <file2> -out <filename></pre>
-recursive	<p>Compares with subunits. DiffMerge loads the subunit files automatically. By default, DiffMerge compares without the subunits.</p>	<pre>Diffmerge.exe <file1> <file2> -recursive</pre>
-xcompare	<p>Starts the DiffMerge tool in Compare mode and displays the interface.</p>	<pre>Diffmerge.exe <file1> <file2> -xcompare</pre>
-xmerge	<p>Starts the DiffMerge tool in Merge mode and displays the interface.</p>	<pre>Diffmerge.exe <file1> <file2> -xmerge</pre>

IBM Rational Synergy

Rational Rhapsody supports CM tools, including IBM® Rational® Synergy®. This subject provides information and procedures on Rational Synergy and Rational Rhapsody.

Setting up Rational Rhapsody for use with Rational Synergy

Before you can use Rational Synergy with Rational Rhapsody, you need to set up Rational Rhapsody so that the two systems can communicate with each other using SCC mode. Be certain that the following operations are performed to establish that communication:

1. Install the SCC add-on for Rational Synergy. Download it from the IBM Web site at <http://www-01.ibm.com/support/docview.wss?uid=swg21380569>:
 - a. Select your Rational Synergy product version.
 - b. Sign in with your IBM ID and password.
 - c. On the Downloads page, in the Integrations section, select the check box for an applicable PC Integration and download the file.
2. To enable the integration and display the Rational Synergy toolbar within Rational Rhapsody, add the following flag to the General section of `rhapsody.ini` file:
`ShowSynergyTaskBar=TRUE`
3. Be certain that the location of the `ccm.exe` file (in `<Synergy Installation Directory>\bin`) is in the PATH environment variable.
4. When using [Rational Synergy and the Rational Rhapsody DiffMerge tool](#), you need to include the location of the `DiffMerge.exe` file that is in the Rational Rhapsody installation folder (for example, `<Rhapsody installation path>\Rhapsody73`) in the PATH environment variable.

5. Import the Rational Rhapsody type definition file into Rational Synergy/CM. This adds Rational Rhapsody types to the Rational Synergy Type Manager. It also links the Rational Rhapsody DiffMerge tool to perform comparison and merging on Rational Rhapsody files within Rational Synergy. Download this file from the IBM Web site at <http://www-01.ibm.com/support/docview.wss?uid=swg21380564>.
 - a. Select your Rational Rhapsody product version.
 - b. Sign in with your IBM ID and password.
 - c. On the Downloads page, in the Integrations section, select the check box for the Rational Rhapsody Type Definition File and Application Note File and download the file.
 - d. After it has completed downloading, see the instructions provided with the file.
6. Add the following code to your `ccm.ini` file:

```
rhapsodytypes_merge_cmd = DiffMerge.exe -xmerge %file1  
                           %file2 -base %ancestor -out %outfile
```

Rational Synergy and Rational Rhapsody

With Rational Synergy and Rational Rhapsody, you can perform all of the standard configuration management operations and these additional operations:

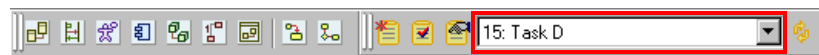
- ◆ [Creating new Rational Synergy tasks](#)
- ◆ [Checking in Rational Rhapsody work](#)
- ◆ [Viewing the properties for a Rational Synergy task](#)
- ◆ Viewing assigned tasks and setting up or changing the current task
- ◆ Refreshing the Rational Synergy task list in Rational Rhapsody to display newly created tasks

Using Rational Synergy with Rational Rhapsody

To begin using Rational Synergy with Rational Rhapsody:


1. To determine where to save your Rational Rhapsody project within Rational Synergy, right-click the Rational Synergy project (in which you want to save a Rational Rhapsody project) in the Rational Synergy work area and select **Explore**, and then note the path to the project on your local disk.
2. Within Rational Rhapsody, create a new Rational Rhapsody project and save it in the directory for the Rational Synergy project in the Rational Synergy work area on your local drive.
3. Set the Rational Rhapsody `Configuration Management::General::UseSCCTool` property to `Yes` to indicate that you are using Rational Synergy and not one of the other configuration management tools. Other CM tools are selected with the `CMTTool` property.

If you do not see the Rational Synergy toolbar, as shown in the following figure, you must change the `ShowSynergyTaskBar` flag in the General section of `rhapsody.ini` file to `TRUE`. Or if the flag is set correctly, make sure that viewing the toolbar has not been switched off. Choose **View > Toolbars Synergy tasks** to reset it.




Connecting to the Rational Synergy archive

To connect to the Rational Synergy archive with Rational Rhapsody:

1. In Rational Rhapsody, open the Configuration Items window. Choose **File > Configuration Items**.
2. Click the Connect to Archive button . The Startup Info window opens.
3. Make any changes needed on the Startup Info window, and click **Continue**. The Open Synergy/CM Project window opens.
4. On the Open Synergy/CM Project window, select the **Scope** to list the groups of projects managed in Rational Synergy. Then select the **Project Name** and **Project Version** for the specific project you are going to use.
5. Click **OK** to close the Open Synergy/CM Project window.
6. Click **OK** to dismiss the confirmation message.

Creating new Rational Synergy tasks

In Rational Rhapsody, to create each Rational Synergy task that you want to use:

1. Click the Create New Task button  on the Rational Synergy toolbar in Rational Rhapsody.
2. In the window that opens, type the **Task Synopsis** name that you want to appear in the drop-down list on the Rational Synergy toolbar in Rational Rhapsody.
3. Enter any necessary information on this window. The entries in these boxes become the properties of the task. That information is accessible for each task, as described in [Viewing the properties for a Rational Synergy task](#).
4. Click **Assign** to save the task in Rational Synergy and make it available for use in Rational Rhapsody.

Note


When you create a task, Rational Synergy names it, by default, as `Task <task_number>`. This is also the default for the `ConfigurationManagement::Synergy::AssignedTasksItsTaskId` property in Rational Rhapsody. However, when you configure your DCM server, you can set it to insert a prefix before `<task_number>`. In this case, you might want to update the `ConfigurationManagement::Synergy::AssignedTasksItsTaskId` property to use the regular expression you want. For example, if you set the prefix `ukan#` on your DCM server and you want this to appear in Rational Rhapsody too, then you should set the value for this property to `Task ukan#([0-9\.]+)` otherwise the default is `Task ([0-9\.]+)`. The value in this property must match the value set on your DCM server for created tasks to appear as you want in the drop-down list on the Rational Synergy toolbar in Rational Rhapsody.

You might also want to set the `ConfigurationManagement::Synergy::AssignedTasksItsTitle` property and the `ConfigurationManagement::Synergy::GetCurrentTaskItsTaskId` property. Use `AssignedTasksItsTitle` to specify a regular expression for the title of a task. Its default value is `Task (.*)`. You would use `GetCurrentTaskItsTaskId` while getting the current task from Rational Synergy to check if the ID for the current task is matching the given regular expression. Its default is `((^[#]*#)?[0-9\.]+)`.

Viewing the properties for a Rational Synergy task

Each Rational Synergy task has information supplied when it is created. These are the properties of each task.

To examine the properties for a task:

1. With the task appearing in the drop-down list on the Rational Synergy toolbar, click the Task Properties button .
2. You can view the task properties, but not change them.


Working with a Rational Synergy task in Rational Rhapsody

To create elements in Rational Rhapsody and use Rational Synergy for configuration management:

1. Be certain that the Rational Rhapsody project is open in the Rational Synergy work area.
2. Select the Rational Synergy task from the drop-down list on the Rational Synergy toolbar, Create Use Cases, as shown in the following figure:




Note

If the tasks you created in Rational Synergy do not appear in the drop-down list, click the Refresh button . In addition, see the note in [Creating new Rational Synergy tasks](#).

Checking in Rational Rhapsody work

After working on the selected task in Rational Rhapsody, check it into Rational Synergy.

1. Click the Checkin Current Task button  on the Rational Synergy toolbar.
2. All units in the task are checked into Rational Synergy, and the processing messages display on the Configuration Management tab of the Rational Rhapsody Output window.

Rational Synergy and the Rational Rhapsody DiffMerge tool

To use Rational Synergy with the Rational Rhapsody DiffMerge tool, use these methods:

- ◆ Include the location of the DiffMerge.exe file that is in the Rational Rhapsody installation folder (for example, <Rhapsody installation path>\Rhapsody73) in the PATH environment variable.
- ◆ Import the Rational Rhapsody type definition file into Rational Synergy/CM. This adds Rational Rhapsody types to the Rational Synergy Type Manager. It also links the Rational Rhapsody DiffMerge tool to perform comparison and merging on Rational Rhapsody files within Rational Synergy. For instructions on how to import this file, see step 5 in [Setting up Rational Rhapsody for use with Rational Synergy](#).
- ◆ To use the Rational Synergy Text Diff tool from the Rational Rhapsody DiffMerge tool, specify these settings from the Rational Rhapsody DiffMerge through **View > Preferences** and set the following values. After you set and save any changes to these preferences, they are set for every time you use the DiffMerge tool.
 - BaseAwareTextDiffMergeEnabled. Select the check box (TRUE)
 - BaseAwareAutoMergeInvocation. Enter value of `ccm_merge.exe -3edu -l $source1 -r $source2 -a $sourceBase -z $output`

Customize Rational Rhapsody and Rational Synergy

To customize Rational Rhapsody and Rational Synergy operations on the task toolbar you can use the properties available in the `ConfigurationManagement::SYNERGY` metaclass from the **Properties** tab in the Features window.

IBM Rational ClearCase

Rational Rhapsody supports CM tools, including IBM® Rational® ClearCase®, in either of two main modes: Batch and SCC. This subject discusses the Rational Rhapsody and Rational ClearCase integration, mostly in Batch mode.

To determine which mode might be best for your situation, review [Batch mode Versus SCC mode](#).

Batch mode Versus SCC mode

Rational Rhapsody supports CM tools, including Rational ClearCase, in either of two main modes: Batch and SCC.

Batch mode is the traditional method of interacting with CM tools that do not conform to the SCC standard. In this mode, Rational Rhapsody has a custom set of properties for each tool that launch tool-specific commands for the CM operations.

SCC mode is an alternate method of interacting with CM tools that conform to the SCC standard. In SCC mode, you need set only one property to interface with any of dozens of SCC-compliant CM tools, without further customization. You interact directly with the GUI elements for the CM tool to perform SCC-supported operations. Return status information, or error information in the case of failure, comes directly from the CM tool. In this way, you have more direct CM tool interaction, and receive more complete feedback on CM operations, in SCC mode.

Each mode has pros and cons in the Rational Rhapsody and Rational ClearCase integration. To help you decide which mode might be best for your situation, this topic provides you with an in-depth understanding of these differences. It also provides you with a decision-making process to help Rational Rhapsody and Rational ClearCase users determine which mode might better address their needs.

The differences between the Batch and SCC modes

The following table lists all the differences between the Batch and SCC modes of Rational Rhapsody/Rational ClearCase integration.

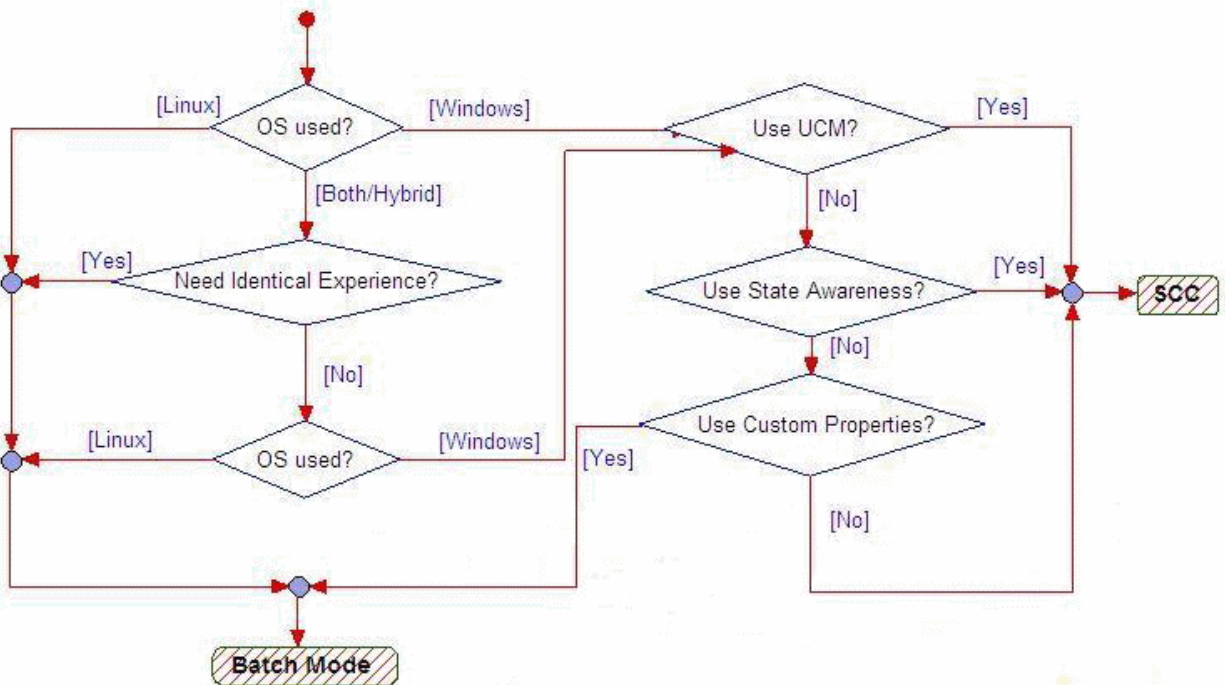
CM Operation	Batch Mode	SCC Mode	Comments
Operating System			
OS	Windows and Linux	Windows	SCC mode is available only on Windows, while Batch mode is available on both Windows and Linux. Note that this is the operating system that Rational Rhapsody and the Rational ClearCase client are installed on. It is not the operating system that the Rational ClearCase server is installed on.
Basic CM Operations			
Checkout	You can specify which version to check out	You cannot specify the version to check out. It depends on the configuration specification.	In Batch mode, you never know what version should be typed in the window. Best practice in both cases is the same, check out directly from the version tree and reload the unit using "synchronize."
Fetch	N/A	N/A	N/A (Not Applicable) in Rational ClearCase. The view concept means the file is always "fetched."
Uncheckout	Supports uncheckout of multiple items with descendant.	A single item can be unchecked out and descendants are not supported.	Although Uncheckout is available in both modes, SCC does not have Uncheckout With Descendants.
Exploring History and Past Views			
History	No	Yes	History and version tree provides the same information but in different form. While version tree provides a graphical view, history just lists the versions. It is possible to switch from one view to the other in both cases.
Version Tree	Yes	No	
Diff with Rhapsody	Yes	Yes	This operation can be performed successfully in SCC only if the hybrid mode is enabled. Therefore, the <code>ConfigurationManagement::General::UseHybridModeWhenPossible</code> property should be set to <code>Checked</code> in this case.
Properties	No	Yes	Available only in SCC mode. In Batch mode it is possible through user-defined buttons (needs a simple command: <code>describe -graphical</code>).
Advanced CM Interaction and UCM			
UCM support and other Advanced Options	No	Yes	If using SCC mode, during the CM operations you can get to advanced options.

CM Operation	Batch Mode	SCC Mode	Comments
Customization	Yes	No	While advanced options are not available in Batch mode in the same way, it is possible to achieve many of those functionalities through properties customization in Batch mode.
Repository Restructuring			
Rename Member	Yes	Yes	Available in both modes depending on the value of <code>ConfigurationManagement::ClearCase::RenameActivation</code> (if you are using Batch mode) or <code>ConfigurationManagement::SCC::RenameActivation</code> (if you are using SCC mode). When this operation is performed on a package-as-directory in SCC, the repository can be restructured successfully only if hybrid mode is enabled. Therefore, the <code>ConfigurationManagement::General::UseHybridModeWhenPossible</code> property should be set in this case.
Move Member	Yes	Yes	Available in both modes depending on the value of <code>ConfigurationManagement::ClearCase::MoveActivation</code> (if you are using Batch mode) or <code>ConfigurationManagement::SCC::MoveActivation</code> (if you are using SCC mode). When this operation is performed on a package-as-directory in SCC, the repository can be restructured successfully only if hybrid mode is enabled. Therefore, the <code>ConfigurationManagement::General::UseHybridModeWhenPossible</code> property should be set in this case.
Delete Member	Yes	Yes	Available in both modes depending on the value of <code>ConfigurationManagement::ClearCase::DeleteActivation</code> (if you are using Batch mode) or <code>ConfigurationManagement::SCC::DeleteActivation</code> (if you are using SCC mode). When this operation is performed on a package-as-directory in SCC, the repository can be restructured successfully only if hybrid mode is enabled. Therefore, the <code>ConfigurationManagement::General::UseHybridModeWhenPossible</code> property should be set in this case.

CM Operation	Batch Mode	SCC Mode	Comments
Set a package to be stored in its own directory	Yes	Yes	Available in both modes depending on the value of <code>ConfigurationManagement::ClearCase::StoreInSeparateDirectoryActivation</code> (if you are using Batch mode) or <code>ConfigurationManagement::SCC::StoreInSeparateDirectoryActivation</code> (if you are using SCC mode). The corresponding directory will be created in CM archive as well, and the relevant .sbs file with its entire descendants will be moved to this directory. In SCC, this operation is executed in hybrid mode. Therefore, in addition to the above-mentioned property, the <code>ConfigurationManagement::General::UseHybridModeWhenPossible</code> property should be set as well.
Set a package not to be stored in its own directory	Yes	Yes	Available in both modes depending on the value of <code>ConfigurationManagement::ClearCase::StoreInSeparateDirectoryActivation</code> (if you are using Batch mode) or <code>ConfigurationManagement::SCC::StoreInSeparateDirectoryActivation</code> (if you are using SCC mode). The relevant .sbs file with its entire descendants will be moved to the parent directory, and the directory created for this package will be removed from the archive. In SCC, this operation is executed in hybrid mode. Therefore, in addition to above-mentioned property, the <code>ConfigurationManagement::General::UseHybridModeWhenPossible</code> property should be set as well.
CM Commands Execution Mode			
User interaction and feedback	Commands are executed in a shell as batch commands. Errors or other messages are shown to the user in the Rational Rhapsody Output window. Rational Rhapsody is not able to react to CM errors.	Interacts directly with CM tool using their UI and API. Rational Rhapsody is aware of CM errors and reacts accordingly.	The user interacts directly with the CM tool's GUI elements to perform SCC-supported operations. Return status information, or error information in the case of failure, comes directly from the CM tool. In this way, Rational Rhapsody has direct CM tool interaction, and receives more complete feedback on CM operations, in SCC mode.
CM State Awareness			
CM State Awareness	No	Yes	CM State Awareness is available in SCC if <code>ConfigurationManagement::SCC::ShowCMStatus</code> is set to Checked (check box is selected).

SCC mode or Batch mode?

In order to decide whether Batch mode or SCC mode better addresses your needs, you should answer the following questions in order. Based on your response to each question, you will progress through the set of questions as applicable until you are finally advised as to what particular mode might be best for your situation. The following figure illustrates the questions graphically.



Question 1: Which operating system are you using?

Rational Rhapsody runs both on Windows and Linux.

- ◆ If all of your team members run Rational Rhapsody on Linux, you have to use Batch mode because SCC is not available on Linux.
- ◆ If all of your team members run Rational Rhapsody on Windows, continue with [Question 4: Are you going to use Rational ClearCase UCM from the Rational Rhapsody interface?](#)
- ◆ If some of your team members run Rational Rhapsody on Linux while others run it on Windows, or if some members run Rational Rhapsody on both operating systems, continue with the next question.

Question 2: Is an identical Rational Rhapsody/Rational ClearCase integration experience required?

- ◆ If it is significant that all users have the same Rational Rhapsody/Rational ClearCase integration experience, or if some users run Rational Rhapsody on both operating systems, then you should prefer Batch mode.
- ◆ If an identical experience is not mandatory, continue with the next question.

Question 3: Is Rational Rhapsody run on Windows or Linux?

- ◆ If Rational Rhapsody is going to run on Linux, you have to use Batch mode because SCC is not available on Linux.
- ◆ If not, continue with the next question.

Question 4: Are you going to use Rational ClearCase UCM from the Rational Rhapsody interface?

- ◆ If you are going to use Rational ClearCase UCM (Unified Change Management) from the Rational Rhapsody interface, then SCC mode is the only option because UCM from the Rational Rhapsody interface is not supported in Batch mode. When you use Rational ClearCase in SCC mode, Rational Rhapsody supports UCM so that you can make use of “activities” to enforce defect and change tracking with the code development. Note, however, that you can use UCM in Batch mode and set “activities” externally although Rational Rhapsody does not provide you this capability through its interface.
- ◆ If you do not have to use UCM from the Rational Rhapsody interface, continue with the next question.

Question 5: Are you going to use CM State Awareness feature of Rational Rhapsody?

- ◆ Rational Rhapsody has the CM State Awareness feature only in SCC mode. Therefore, if you want to make use of state information of units (for example, checked in, checked out), you should prefer SCC mode.
- ◆ If CM State Awareness is not required, continue with the next question.

Question 6: Are you going to use custom properties for CM operations?

- ◆ If you have already made an investment in customizing the properties in Rational Rhapsody for CM operations (for example, you have a custom script that handles check in and set the `ConfigurationManagement::ClearCase::CheckIn` property to launch this script) and if you want to continue using these custom properties, you have to use Batch mode because SCC does not let you do such a customization.
- ◆ If you have not made an investment in customizing properties or you do not have to use these custom properties, you can use SCC mode.

SCC Mode or Batch Mode Summary

The Rational Rhapsody/Rational ClearCase integration is available in Batch or SCC mode with the following considerations:

- ◆ Windows users might use either mode.
- ◆ Linux users must use Batch mode because SCC is not available on Linux.

See [The differences between the Batch and SCC modes](#) and [SCC mode or Batch mode?](#)

For those who can use it, the SCC mode has the following advantages:

- ◆ Improved stability and error detection
- ◆ Ability to use Rational ClearCase UCM from the Rational Rhapsody interface
- ◆ Can benefit from the CM State Awareness feature of Rational Rhapsody

For those who can use SCC mode but who already have custom Rational Rhapsody properties, you should decide which one is more significant for you: UCM and CM State Awareness or customized CM behavior through these properties.

Setting up Rational ClearCase

To set up a Rational ClearCase environment, consult the documentation that accompanies the Rational ClearCase application. Conceptually, the following steps should accomplish the task. However, the process might differ depending on your Rational ClearCase environment and which version of Rational ClearCase you are using.

To set up Rational ClearCase to use as your Rational Rhapsody source control management tool:

1. Using Rational ClearCase, create a directory to serve as the VOB mounting point.
2. Create the VOB (`mkvob`) and mount it (`mount`). Alternatively, you can use the VOB Creation Wizard.
3. Create a view (`mkview`) and activate it (`startview`). Alternatively, you can use the View Creation Wizard.
4. Add to version control (`mkelem`) any directory that is a parent to the Rational Rhapsody workspace repository (the `_rpy` directory). This includes the directory that you can optionally create as part of a new Rational Rhapsody project.

Note

Check out (`reserve`) this directory before connecting to the archive. Connect to Archive basically makes the `<project>_rpy` directory a VOB element, and therefore should be done only once, by one user. In Rational ClearCase, adding a new VOB element performs a Check In operation as well.

You should place a Rational Rhapsody project within a directory with the same name as the project. For example, create a directory called `MyProject` and save your Rational Rhapsody project to this directory so it contains the `MyProject.rpy` file and `MyProject_rpy` directory.

Controlling case sensitivity in Rational ClearCase

When you save a model in a Rational ClearCase directory (under source control), it is assigned a name in lowercase letters. When you connect it to an archive, Rational Rhapsody creates the directory with an uppercase name. This leaves you with the file `<project>.rpy` in lowercase and the directory `<PROJECT>_RPY` in uppercase, which does not work in Rational ClearCase.

To fix the problem:

1. Change the MVFS settings in the **Control Panel > ClearCase** applet to be “case preserving.”
2. Reboot the machine to make the changes take effect.

If this setting is not changed and a project name has uppercase characters, then, in some cases, Rational ClearCase will not be able to find the correct `_rpy` files.

About checking out Rational Rhapsody files

In Rational ClearCase, you must pay careful attention to which version of a file needs to be checked out. The default file is usually the most recently archived version. To check out an earlier version, you must specify the earlier version even if that version is already selected by a Rational ClearCase configuration specification.

Note

If you check out the file without specifying the earlier version you wanted, the most recent version of the Rational Rhapsody file is checked out.

About setting up Rational Rhapsody projects for team members

Rational Rhapsody users can create views on their machines, and the Rational ClearCase MVFS automatically makes the Rational Rhapsody projects that exist in the VOB visible in those views. No additional steps are required; any user can launch Rational Rhapsody and open the `.rpy` file located in the newly created view.

About adding new files to the archive

When users add new elements to their models, team members might see them in the Show Items window (Archive Members window) as eligible for checkout, because adding a member with Rational ClearCase also checks it in. To avoid conflicts, the user creating the new element should immediately check out the element after adding it to the archive.

Rational ClearCase limitations with Rational Rhapsody

Note the following Rational ClearCase limitation with Rational Rhapsody:

- ◆ Rational Rhapsody does not support the Rational ClearCase “snapshot” view.
- ◆ Upgrading Rational Rhapsody on UNIX does not automatically update the soft links (as opposed to Windows installation that will update the registry).

Rational ClearCase semantics

The following list shows the semantics of common CM operations as specifically implemented for Rational ClearCase.

- ◆ The Connect to Archive operation makes the <project>_rpy directory a VOB element and creates a new version-controlled directory for it.
- ◆ The List Archive operation lists the files that appear in the current view that are under version control.
- ◆ The Add Member operation adds the configuration item file to version control.
- ◆ The Check In operation checks in configuration items, even if the current items are identical to those already in the archive. Rational ClearCase does not have Check In with Lock capabilities, so selecting the Lock option has no effect.
- ◆ The Check Out operation checks out the configuration item. The Lock option activates the Rational ClearCase Reserved option.

Evil twins issue

In Rational ClearCase, you should avoid using the same name for different elements. This type of situation creates what is called “evil twins.” For an explanation and guidance about the evil twins issue in the Rational ClearCase environment, see the technote provided on the IBM Web site at <http://www-1.ibm.com/support/docview.wss?rs=984&uid=swg21125072>.

Integration issues

You can set up your Rational ClearCase environment so `diff` and `merge` commands from within Rational ClearCase (either from the command line, version tree, or Rational ClearCase Merge Manager) automatically launch the Rational Rhapsody DiffMerge tool. See [Setting up the Rational ClearCase Type Manager](#).

To perform difference identification and merging operations, you might want to integrate these Rational ClearCase textual difference tools with the Rational Rhapsody DiffMerge tool:

- ◆ `cleardiffmrg`
- ◆ `cleardiff`

To integrate these tools, you need to set the following Rational ClearCase values as shown:

- ◆ `BaseAwareAutoMergeInvocation`. Set to `cleardiff.exe -out $output -base $sourceBase -abo -qui $source1 $source2`
- ◆ `BaseAwareDiffInvocation`. Set to `cleardiffmrg.exe -base $sourceBase $source1 $source2`
- ◆ `BaseAwareDiffMergeInvocation`. Set to `$BaseAwareDiffInvocation -out $output`
- ◆ `BaseAwareTextDiffMergeEnabled`. Set to `TRUE`
- ◆ `DiffInvocation`. Set to `cleardiffmrg.exe $source1 $source2`
- ◆ `DiffMergeInvocation`. Set to `$DiffInvocation -out $output`

These value setting changes launch the Rational ClearCase textual merge tool `cleardiffmrg` from the Rational Rhapsody DiffMerge tool when a user performs diff/merge of operation bodies. Similarly, DiffMerge silently launches `cleardiff` to determine if the given differences are trivial or non-trivial and to merge them automatically in base-ware mode.

This approach is made possible because all elements stored in a Rational ClearCase VOB have a type. *Rational ClearCase administrators* can define a new type and associate it with a Diff tool and a Merge tool.

Hierarchical repository and Rational ClearCase

To perform CM operations on Rational Rhapsody files, the directory containing them must be a VOB element. Directories created by Rational Rhapsody could be either the project directory (for example, `MyProject_rpy`) or directories created to contain packages.

The `*_rpy` directory becomes a VOB element once you connect to the archive.

In addition, package directories can be made into VOB elements when you initially create them (for more information, see the property definitions of

`ConfigurationManagement::ClearCase::MakeCMShadowDirActivation` and `ConfigurationManagement::ClearCase::MakeCMShadowDir` on the **Properties** tab of the Features window in Rational Rhapsody).

However, in the situation where you try to “add to source control” for a Rational Rhapsody element and the operation fails because the parent directory is not a VOB element, you can make the directory a VOB element using Rational ClearCase, then continue working in Rational Rhapsody.

Changes to an existing directory structure

If you want to change the directory structure of the repository files (for example, from flat to hierarchical), you must make changes to both Rational Rhapsody and Rational ClearCase. Although branch creation, maintenance of configuration specification files, and the overall policy of branch and merge is done outside of the scope of Rational Rhapsody, the merge process requires the usage of the Rational Rhapsody DiffMerge tool. The DiffMerge tool allows you to examine the differences between Rational Rhapsody units in a clear, visual interface and to merge two versions of the same unit into a third, new unit.

For detailed information about the DiffMerge tool, see [Parallel development](#).

However, using the Rational Rhapsody DiffMerge tool to make these changes requires some adjustments:

- ◆ You must manually type in the unique identifier of each of the versions to be compared.
- ◆ You must manually create a Rational ClearCase link (merge arrow) from the merge source to the merge target.
- ◆ By working from within Rational Rhapsody, you are not using the Rational ClearCase visual version tree representation efficiently.

Limitations for changing an existing directory structure

Note the following limitation for changing an existing directory structure:

- ◆ When the Rational Rhapsody DiffMerge tool is activated from the Rational ClearCase version tree (see [Rational ClearCase Type Manager](#)), the DiffMerge tool does not expose the option **With Descendant**.
- ◆ The Rational Rhapsody DiffMerge tool can compare **two versions of a file** (as opposed to the Rational ClearCase default text diffmerge tool that can handle up to 32 files).
- ◆ If a client machine needs to work with **two different versions of Rhapsody simultaneously**, the map file pointer on Windows, which is based on a registry key, needs to be manually set to the correct version of the Rational Rhapsody DiffMerge tool. The registry key is set by the Rational Rhapsody installation wizard. Similarly, a UNIX client will need to modify the `compare`, `xcompare`, `merge`, `xmerge` soft links to ensure invocation of the correct version of the DiffMerge tool.

Rational ClearCase Type Manager

Using a Rational ClearCase extension mechanism called `type_manager`, you can launch the Rational Rhapsody DiffMerge tool directly from the Rational ClearCase version tree tool, merge a unit from one branch to another, and draw the hyperlink arrow (all as a single atomic operation). For more information about this feature, see [Launching DiffMerge inside Rational Rhapsody](#).

All elements stored in a Rational ClearCase VOB have a type. Rational ClearCase administrators can define a new type and associate it with a Diff tool and a Merge tool. For more information about this approach, see the Rational ClearCase documentation (search keywords: `type_manager`, `map file`, `magic file`, `magic_path`). Your Rational ClearCase administrator should be familiar with these concepts prior to the implementation phase.

Your Rational ClearCase administrator needs to implement the necessary changes for this enhancement, coordinating all changes with project management, for the following reasons:

- ◆ Not all users will need this enhancement.
- ◆ Not all projects will need this enhancement.
- ◆ The setup involves modifying the Rational ClearCase setup and configuration files, which is usually not handled by end users.

Setting up the Rational ClearCase Type Manager

To set up the Rational ClearCase Type Manager:

Note

Step 1 – 4 are described using the Windows GUI. To implement the same functionality on UNIX servers, use the following sample command:

```
jupiter 21> cleartool mkeltype -super text_file -manager _rhp
-mergetype auto rhp_file
Comments for "rhp_file":
Rhapsody units files
.
Created element type "rhp_file".
jupiter 22>
```

1. Run the Rational ClearCase **Home Base** tool. The Home Base window opens.
2. On the **VOBs** tab, click **Type Explorer**.
3. In the Type Explorer window, select the required VOB and open the element type.
4. Open the Create window. Choose **Type > Create**.
5. Define a new type, `rhp_file` and click **OK**.
6. Edit the properties for the new type manager using the Properties window. Set the following values:
 - ◆ Make sure **Supertype** is `text_file`.
 - ◆ Select the **Override type manager** check box and `type _rhp` in the text box.
 - ◆ For the **Merge Type**, select the **Use type manager's merge method** radio button.

7. On Windows NT servers, open the map file (usually located in the directory <ClearCase home dir>\lib\mgrs) and add the following section to the map file (in, for example, C:\Program Files\Rational\ClearCase\lib\mgrs):

```
_rhpconstruct_version          <same entry used for text_file_delta>
_rhpcreate_branch              <same entry used for text_file_delta>
_rhpcreate_element            <same entry used for text_file_delta>
_rhpcreate_version            <same entry used for text_file_delta>
_rhpdelete_branches_versions <same entry used for text_file_delta>
_rhpcompare                   <path for Rhapsody files>\DiffMerge.exe
_rhpxcompare                  <path for Rhapsody files>\DiffMerge.exe
_rhpmerge                     <path for Rhapsody files>\DiffMerge.exe
_rhpxmerge                    <path for Rhapsody files>\DiffMerge.exe
_rhpannotate                  <same entry used for text_file_delta>
_rhpget_cont_info            <same entry used for text_file_delta>
```

In this syntax:

- ◆ <same entry used for text_file_delta> means you should copy the entry that already exists for the text file type_manager into the Rational Rhapsody file type_manager.
- ◆ <path for Rhapsody files> is the full path to the Rational Rhapsody DiffMerge tool.

Example:

If the DiffMerge.exe is installed on all users' machines on

D:\Rhapsody\DiffMerge.exe and all the text_file_delta entries are
 ..\..\bin\tfdmgr.exe, add the following section to the map file:

```
_rhpconstruct_version          ..\..\bin\tfdmgr.exe
_rhpcreate_branch              ..\..\bin\tfdmgr.exe
_rhpcreate_element            ..\..\bin\tfdmgr.exe
_rhpcreate_version            ..\..\bin\tfdmgr.exe
_rhpdelete_branches_versions  ..\..\bin\tfdmgr.exe
_rhpcompare                   D:\Rhapsody\DiffMerge.exe
_rhpxcompare                  D:\Rhapsody\DiffMerge.exe
_rhpmerge                     D:\Rhapsody\DiffMerge.exe
_rhpxmerge                    D:\Rhapsody\DiffMerge.exe
_rhpannotate                  ..\..\bin\tfdmgr.exe
_rhpget_cont_info            ..\..\bin\tfdmgr.exe
```

Instead of D:\Rhapsody\DiffMerge.exe, you can use:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Telelogic\Rhapsody\DiffMerge\
Install Path
```

This creates a new key.

Note: The old key, created in Rhapsody versions before 7.1, was under
 HKEY_LOCAL_MACHINE\SOFTWARE\I-Logix\Rhapsody\DiffMerge\
 Install Path. This key is no longer used and should be changed to the new
 key listed above.

8. On UNIX/Solaris servers, do the following steps:

- ◆ Under the `<clearcase install dir>/mgrs` directory, create a `_rhp` directory.
- ◆ Under the `_rhp` directory, create the following links:
 - `compare`, `xcompare`, `merge`, and `xmerge` should point respectively to the `DiffMerge_Compare.exe`, `Diffmerge_XCompare.exe`, `DiffMerge_Merge.exe`, and `DiffMerge_XMerge.exe`. (The Rational Rhapsody installation program creates these files in the installation directory.)
 - All other links should point to the `text_delta_file` entry.

Setting up the Rational ClearCase .magic file

To be able to use the Rational ClearCase Type Manager with Rational Rhapsody or the Rational Rhapsody Eclipse plug-in for the Eclipse platform, you must set up the Rational ClearCase .magic file. By default, the .magic file is named `default.magic`. However, yours might have a different name.

Note

The supported Rational ClearCase Eclipse plug-in is version 7.0.0.20080131A.

To set up the Rational ClearCase .magic file:

1. Open the .magic file that is visible to all users at your site and add the following lines to the beginning of the

```
# Match by name without examining data core file : -name "core" ;
```

and *before* the

```
# (seems printable, but has binary at the end of it)
lisp_object object_module file : -name "*.lbin" ;
```

sections of the .magic file.

```
#### Rhapsody file types (begin) ####
rhp_file: -name "*.rpy";
rhp_file: -name "*.sbs";
rhp_file: -name "*.cls";
rhp_file: -name "*.omd";
rhp_file: -name "*.cmp";
rhp_file: -name "*.ctd";
rhp_file: -name "*.clb";
rhp_file: -name "*.ucd";
rhp_file: -name "*.msc";
rhp_file: -name "*.std";
rhp_file: -name "*.dpd";
rhp_file: -name "*.fil";
rhp_file: -name "*.fol";
#### Rhapsody file types (end) ####
```


2. Save your changes to the .magic file.

For more information about setting up the Rational ClearCase .magic file, see the technote provided on the IBM Web site at (<http://www-1.ibm.com/support/docview.wss?rs=0&uid=swg21118099>).

Rational Rhapsody models and changing the default properties

For the enhancement to work, Rational Rhapsody files should be of type `rhp_file`, not `text_file`. This issue has two aspects:

- ◆ Ensuring that the development team is adding new Rational Rhapsody units as type `rhp_file`, not `text_file`
- ◆ Converting Rational Rhapsody units that are already stored under a Rational ClearCase VOB from type `text_file` to `rhp_file`

To ensure new Rational Rhapsody units are being added as type `rhp_file`, change the following property in the Rational Rhapsody properties file (usually `site.prp`):

```
Subject: ConfigurationManagement
MetaClass: ClearCase
Property: AddMember
```

Note

The only difference is the `-eltype` flag for the Rational ClearCase `mkelem` command (shown in **bold**).

To convert Rational Rhapsody units that are already stored under a Rational ClearCase VOB from `text_file` to `rhp_file`, use the Rational ClearCase `chtype` command.

Code generation performance improvements

If you notice that performance is reduced, you might need to change the location of the temporary files.

To change the location for temporary files, change the following entry in the `Rhapsody.ini` file, as follows:

```
[CodeGen]
TemporaryFilesDirectory=<the temporary code generation
files directory (for example, C:\TEMP)>
```

When the entry is not present, temporary files are created in the project directory where the `.rpy` file is stored. By changing the location to a directory outside of a Rational ClearCase VOB domain, code generation performance improves.

Note

When you initially generate code inside a Rational ClearCase VOB, there is still performance loss due to VOB overhead (creating the code directories and files). You will see the full performance gain beginning with the second code generation.

Forced check in of a package with unchanged subunits

If you check out a package with descendants and change only a few subunits, checking in the package can be problematic because Rational ClearCase will sometimes refuse to check in an unchanged element. To overcome this problem, use the `SensitiveCheckin.bat` file provided in the Rational Rhapsody distribution. The file is located in `<root>/Share/etc`.

The batch file compares each file to the version that was checked out. If the two versions are the same, an Undo Checkout operation is performed. Otherwise, the changed version is checked in.

To use this file, it must be in the `$OMROOT/etc` directory. In addition, you must set the `ConfigurationManagement::ClearCase::CheckIn` property to the following value:

```
$OMROOT/etc/SensitiveCheckin.bat $UnitPath $log
```

When is a Rational ClearCase license consumed?

In general, any Rational ClearCase operation that involves the database (MVFS access) or the `albd_server` will most likely cause a license to be activated for that user. This means that the user has “accessed” Rational ClearCase and that a license has been consumed for that session.

Most anything you do that involves the product consumes a license, for example:

- ◆ When you are in a VOB or looking at data through a view. This is whether you are on a Rational ClearCase host or a non-supported host via an exported view.
- ◆ Use any `cleartool` commands that modify data (for example, checkout and checkin).
- ◆ Call any metadata operations on labels, branches, attributes, triggers, and so forth.
- ◆ Use any listing or reporting commands like `lsvob`, `lsview`, `describe`, `find`, and so forth.
- ◆ Use ClearMake and merge.

For the CM interface license on the Rational Rhapsody side, it is consumed for the entire Rational Rhapsody session.

Regarding the Rational ClearCase license itself, any Rational ClearCase client utility you run (such as `cleartool`) tries to get a license. If it is successful, you keep it for 60 minutes by default. When you enter a Rational ClearCase command during this period of time, the license is renewed. Otherwise, after the period of time, another user can take the license. All this applies when you use the `cleartool` commands from the batch interface.

Regarding the SCC interface, it would depend whether you initialize/uninitialize the interface with each command, or at the first usage/after closing Rational Rhapsody.

Customize Rational Rhapsody and Rational ClearCase

To customize Rational Rhapsody and Rational ClearCase you can use the properties available in the `ConfigurationManagement::ClearCase` metaclass from the **Properties** tab in the Features window.

Checking out/Checking in a directory once

This feature is applicable with Rational ClearCase in batch mode.

By default, when you add multiple files that are contained in the same directory to the archive, Rational Rhapsody checks out/checks in the directory for these files multiple times. (You can see this on the Rational Rhapsody Output window.)

To set it so that Rational Rhapsody check outs/checks to the same directory for multiple files only once, you can set the `ConfigurationManagement::ClearCase::CheckOutCheckInDirectoryOnceDuringAddToArchive` property to `Checked`.

Storing an existing package in a separate directory

This feature is applicable with Rational ClearCase in batch mode.

With the use of the `ConfigurationManagement::ClearCase::StoreInSeparateDirectoryActivation` property, you can set it so that when an existing flat package is converted to a package as directory (you *selected* the **Store in separate Directory** check box on the Unit Information window) the directory is created on the configuration management side and the children of this package are moved to this directory. The following values are available for the `StoreInSeparateDirectoryActivation` property:

- ◆ `Automatic` if you want Rational Rhapsody to automatically do so without asking you to confirm your request.
- ◆ `UserConfirmation` if you want Rational Rhapsody to ask you to confirm your request.
- ◆ `Disable` (the default) means the unit is not added.

Removing an existing directory for a package and reconciling its contents

This feature is applicable with Rational ClearCase in batch mode.

With the use of the `ConfigurationManagement::ClearCase::StoreInSeparateDirectoryActivation` property, you can set it so that when an existing package as directory is converted to a flat package (you *cleared* the **Store in separate Directory** check box on the Unit Information window) the directory is removed on the configuration management side and the children of this package are removed as well. The following values are available for the `StoreInSeparateDirectoryActivation` property:

- ◆ `Automatic` if you want Rational Rhapsody to automatically do so without asking you to confirm your request.
- ◆ `UserConfirmation` if you want Rational Rhapsody to ask you to confirm your request.
- ◆ `Disable` (the default) means the unit is not added.

Note

Files that are not added to the archive in this directory (which will be removed) will not be moved by the configuration management tool. Therefore, if there are any such files, they might be lost after this directory is removed from the archive.

Adding a unit to the CM archive automatically

This feature is applicable with Rational ClearCase in batch mode.

To make it so that Rational Rhapsody adds the relevant file to the configuration management archive after creating a unit, you can, you can use the `ConfigurationManagement::ClearCase::AddToArchiveAfterCreateUnitActivation` property. The following values are available for this property:

- ◆ `Automatic` if you want Rational Rhapsody to automatically do so without asking you to confirm your request.
- ◆ `UserConfirmation` if you want Rational Rhapsody to ask you to confirm your request.
- ◆ `Disable` (the default) means the unit is not added.

Serena PVCS Dimensions

Rational Rhapsody supports CM tools, including Serena® PVCS® Dimensions®. This subject describes how to use Rational Rhapsody with PVCS Dimensions in SCC mode.

Enabling a SCC-compliant CM tool

To enable any SCC-compliant CM tool with Rational Rhapsody, you must set the `ConfigurationManagement::General::UseSCCTool` property to `Yes`. When the `UseSCCTool` property is set to `Yes`, all other tool-specific CM properties are ignored. Rational Rhapsody uses the alternative (batch mode) for traditional CM tools (such as ClearCase) when the `UseSCCTool` property is set to `No`.

Access to Dimensions from Rational Rhapsody

The CM tool administrator must set up access to Dimensions from Rational Rhapsody. The administrator can do this by using the IDE Setup utility in PVCS Dimensions.

Create the initial connection to the SCC tool

To create the initial connection to the SCC tool, use the Connect to Archive tool. Once connected, you can use SCC API commands (such as Check In and Check Out), which will prompt you to log in to the archive, if required.

Once connected, only use the Connect to Archive tool again if you want to disconnect and reconnect to a different archive.

Creating the initial connection to the SCC tool in Dimensions

To connect to an archive with PVCS Dimensions:

1. Open the Configuration Items window. Choose **File > Configuration Items**.
2. Click the Connect to Archive button. Rational Rhapsody searches for the PVCS Dimensions DLL (specified by the `SourceCodeControlProvider` registry key).
 - ◆ If it finds the DLL, Rational Rhapsody launches PVCS Dimensions to perform the Connect to Archive operation.
 - ◆ If it does not find the DLL, the CM tool might not be registered properly in the Windows registry. See [Unable to connect to SCC-Compliant CM tool \(SCC mode\)](#) for troubleshooting information.
 - ◆ You should see whether the specific version of the tool you are using is SCC-compliant. Some tools that are SCC-compliant in one version are not compliant in earlier versions (for example, version 5.0 versus 6.0).
3. If the PVCS Dimensions DLL is found, the PVCS Dimensions Remote Login window opens.
4. Type your user password in the **Password** box, then click the **Connect** button to open the Select Workset Directory for Project window.
5. Click **OK**. PVCS Dimensions connects the project to the archive and confirms completion of the Connect operation.
6. Click **OK**.

An effect of the Connect to Archive operation is that Rational Rhapsody stores project information in both the `ConfigurationManagement::SCC::ProjName` and `ConfigurationManagement::SCC::AuxProjPath` properties, so the project is automatically connected to the same archive the next time you open it. You should never need to modify these properties manually, unless you want to connect the project to a different archive.

Add to SCC archive operation

When you set the `ConfigurationManagement:General:UseSCCTool` property to `Yes`, you can use the Add to Archive Options window to add a unit to an SCC archive. This property sets for use the standard SCC interface between Rational Rhapsody and your CM tool.

Adding a unit to an SCC archive

To add a unit to an SCC archive:

1. In the Configuration Items window, select a unit.
2. Click the Add to Archive button to open the SCC Options window.
3. Select the **Include descendants** check box on the SCC Options window if you want to include nested units in the Add operation.
4. Type a comment describing the units you are adding.
5. If your CM tool has advanced options, you can click the **Advanced** button on the SCC Options window to open its Advanced Options window; otherwise the button is disabled. The Advanced Options window that opens is provided by your CM tool.
6. If you opened an Advanced Options window, click **OK** to close it after you make your selections.
7. Click **OK** on the SCC Options window.
 - ◆ PVCS Dimensions might display a message box to confirm that the Add operation has completed. Click **Close**.
 - ◆ The PVCS Dimensions output might be displayed in the Rational Rhapsody Output window on the **Configuration Management** tab. This behavior is controlled by the `ConfigurationManagement::SCC::RedirectOutputToRhapsody` property. By default, this property is set to `Checked`, so messages are seen in Rational Rhapsody.

PVCS Dimensions adds the unit to the archive and changes its mode to read-only (RO) or unlocked.

Check out operation in SCC archive

In SCC mode, units are always checked out with a lock (RW). You must add a unit to the archive before you can check it out.

Checking out a unit in SCC archive

To check out a unit from an SCC archive:

1. In the Configuration Items window, select a unit that has been added to the archive.
2. Click the Check Out button to open the SCC Options window.
3. Select the **Include descendants** check box on the SCC Options window if you want to include nested units in the Check Out operation.
4. Type a comment describing the units you are checking out.
5. If your CM tool has advanced options, you can click the **Advanced** button on the SCC Options window to open its Advanced Options window; otherwise the button is disabled. The Advanced Options window that opens is provided by your CM tool.
6. If you opened an Advanced Options window, click **OK** to close it after you make your selections.
7. Click **OK** on the SCC Options window.
 - ◆ PVCS Dimensions might display a message box to confirm that the Check Out operation has completed. Click **Close**.
 - ◆ You might see the PVCS Dimensions output in the Rational Rhapsody Output window on the **Configuration Management** tab. This is controlled by the `ConfigurationManagement::SCC::RedirectOutputToRhapsody` property. By default, this property is set to `Checked`, so messages are seen in Rational Rhapsody.

PVCS Dimensions copies the unit from the archive to your workspace and changes the mode for the unit to read/write (RW) or locked.

Check in operation in SCC archive

In SCC mode, you must add a unit to the archive and check it out before you can check it in.

Checking in a unit in SCC archive

To check a unit into an SCC archive:

1. In the Configuration Items window, select a unit that has already been added to the archive and checked out.
2. Click the Check In button to open the SCC Options window.
3. Select the **Include descendants** check box on the SCC Options window if you want to include nested units in the operation.
4. Type a comment describing the units you are checking in.
5. If your CM tool has advanced options, you can click the **Advanced** button on the SCC Options window to open its Advanced Options window; otherwise the button is disabled. The Advanced Options window that opens is provided by your CM tool.
6. If you opened an Advanced Options window, click **OK** to close it after you make your selections.
7. Click **OK** on the SCC Options window.
8. PVCS Dimensions displays a message box to confirm the Check In operation. Click **OK** to dismiss the message box.

PVCS Dimensions copies the unit from your workspace into the archive and changes its mode to read-only or unlocked.

Listing the archive in PVCS Dimensions

To list an archive in SCC mode, click the List Archive button on the Configuration Items window. Rational Rhapsody displays the Archive window, which lists the units that have been added to the archive.

Note

In SCC mode, it is possible to see the version number of the file currently loaded in Rational Rhapsody in both the List Archive and CM Items windows. For more information, see the definitions for the `ConfigurationManagement::PVCS::HeaderFile` and `ConfigurationManagement::PVCS::CMHeaderItsVersion` properties on the **Properties** tab of the Features window.

Fetching in Dimensions

The Fetch operation fetches the latest version of a unit from the archive without a lock (RO). The alternative is the SCC Check Out operation, which always fetches a unit with a lock (RW).

To fetch:

1. In the Configuration Items window, select the units that you want to check out as unlocked.
2. Click the Fetch button to open the SCC Options window.
3. Select the **Include descendants** check box on the SCC Options window if you also want to fetch nested units.
4. If your CM tool has advanced options, you can click the **Advanced** button on the SCC Options window to open its Advanced Options window; otherwise the button is disabled. The Advanced Options window that opens is provided by your CM tool.
5. If you opened an Advanced Options window, click **OK** to close it after you make your selections.
6. Click **OK** on the SCC Options window.

PVCS Dimensions confirms the Fetch operation has completed.

7. Click **Close**.

PVCS Dimensions copies the unit from the archive into your workspace without a lock.

Unchecking Out in Dimensions

The Uncheck Out operation reverses the effect of a Check Out, releasing the lock on a unit (making it RO) and reverting to the file version before the last Check Out operation.

To uncheck out:

1. In the Configuration Items window, select a unit that has been checked out (whose mode is RW).
2. Click the Un-Check Out button to open the SCC Options window.
3. If your CM tool has advanced options, you can click the **Advanced** button on the SCC Options window to open its Advanced Options window; otherwise the button is disabled. The Advanced Options window that opens is provided by your CM tool.
4. If you opened an Advanced Options window, click **OK** to close it after you make your selections.
5. Click **OK** on the SCC Options window.

PVCS Dimensions changes the mode for the unit mode to read-only (RO) and makes the version that was archived prior to the last Check Out the latest working version.

Viewing the history of a unit

The History operation is available in SCC mode only. It is an SCC command that opens the archive for a unit so you can review the history of the unit and access previously archived revisions.

To view the history of a unit, click the History button on the Configuration Items window.

Viewing the file details for a unit

The Properties command retrieves the file details for a unit, such as the file name, the date it was created, and so on.

To display the SCC file details of a unit that is a member of an archive:

1. In the Configuration Items window, select a unit that has been added to the archive.
2. Click the Properties button. PVCS Dimensions displays the file details for the unit.

Customize Rational Rhapsody and PVCS Dimensions

To customize Rational Rhapsody and PVCS Dimensions you can use the properties available in the `ConfigurationManagement::PVCS` metaclass from the **Properties** tab in the Features window.

Concurrent Versions System (CVS)


This subject provides some information that helps you get started with using the Rational Rhapsody/CVS integration in Eclipse. Concurrent Versions System (CVS) is an open source configuration management (CM) tool. The Rational Rhapsody Platform Integration lets software developers work on a Rational Rhapsody project within the Eclipse platform.

For details about how to use Eclipse and CVS, see the documentation provided for those products.

Sharing a Rational Rhapsody project in CVS

This topic assumes you know how to create a CVS repository and connect to an existing CVS repository.

To share a Rational Rhapsody project in CVS:

1. Switch to the Rational Rhapsody Unit View.
 - a. In Eclipse, open the Show View window. Choose **Window > Show View > Other**.
 - b. Select **Rhapsody > Unit View** and click **OK**.
2. In the Unit View window, right-click your project and then select **Team > Share Project** to open the Share Project window, which has various views.
3. On the Enter Repository Location Information view of the Share Project window, enter the repository location information and then click **Next**.
4. On the Enter Module Name view of the Share Project window, make sure the **Use project name as module name** radio button is selected and then click **Next**.
5. On the Share Project Resources view, notice that there is an arrow overlay added to the icons for all the files/folders in your project . This means that those files/folders will be added to the repository. Click **Finish**.
6. Because Eclipse/CVS is not familiar with Rational Rhapsody file extensions (for example, `.sbs`, `.cmp`), it asks you if the content of those files are binary or ASCII text. On the Add Resources view of the Commit Files window, select **ASCII Text** as the content for each extension and then click **Next**.

7. Enter a commit comment on the next view of the Commit Files window and then click **Finish** to commit the operation. This adds those marked units into the repository.

You can switch to the Eclipse perspective to Rational Rhapsody Modeling to keep working on your Rational Rhapsody model. Notice in the Unit View window there is a database overlay added

to the icons for the files/folders in your project . This means that the files/folders are checked in units in a CVS repository.

Checking out a Rational Rhapsody project from a CVS repository

To check out a Rational Rhapsody project from a CVS repository:

1. Start Eclipse.
2. Open the New Project window. Choose **File > New > Project**.
3. Expand the **CVS** folder, select **Projects from CVS**, and then click **Next** to open the Checkout from CVS window, which has various views.
4. On the Checkout from CVS window, select the **Use existing repository location** radio button, select the repository that you want, and then click **Next**.
5. On Select Module view of the Checkout from CVS window, select the **Use an existing module (this will allow you to browse the modules in the repository)** radio button, select the Rational Rhapsody project that you want to check out, and then click **Finish**.

You can now switch to the Rational Rhapsody Modeling perspective in Eclipse to work on the Rational Rhapsody project you have checked out.

Collaboration with other users in CVS

You can use other related CM operations while working on a project that you already checked out so that you can successfully collaborate with other users working on the same Rational Rhapsody project.

Repository synchronization in CVS

In order to see if a Rational Rhapsody unit is still in sync with the repository after you checked it out, right-click the unit in the Unit View window and select **Team > Synchronize with repository**.


- ◆ If there are no differences between the workspace unit and the remote, a message box tells you that there are no changes between the workspace resource and the remote.


You can do an [Updating a Rhapsody unit in Eclipse to the CVS repository](#) to get the incoming changes.


- ◆ If this message does not appear when you synchronize with the repository, it means that there are some differences between the workspace unit and the corresponding repository unit. This means either you modified this unit and/or somebody else has committed some changes on this unit to the repository.


In this case, you can click the **Invoke Rhapsody DiffMerge to compare and merge manually** button to launch the Rational Rhapsody DiffMerge tool. After you merge, you should right-click the unit and select **Mark as Merged**.

If a Rational Rhapsody unit is out-of-sync with the repository, one of the following overlays will appear on the icon for the unit:

 means you have created this unit and it is not added to the repository yet (outgoing new file)

 means the repository contains this new unit that does not exist in your workspace (incoming new unit)

 means this unit is changed in your workspace (outgoing modification)

 means this unit is changed in the repository (incoming modification)

 means this unit is modified both in the repository and in your workspace so there is a conflict

Updating a Rhapsody unit in Eclipse to the CVS repository

While you are working on a Rational Rhapsody unit in Eclipse, other members of your team might have committed changes to the copy of the unit in the repository. To get these changes, you can update your Rational Rhapsody unit to match the repository. You can do this in the following ways:

- ◆ Right-click the unit in the Unit View window and select **Team > Update**.
- ◆ Switch to the Synchronization perspective. Right-click the unit, and select **Update**.

Eclipse lets you configure the **Update** operation to do one of the following choices:

- ◆ **Update all non-conflicting changes and then preview the remaining changes:** All non-conflicting incoming changes will be merged in automatically and any remaining conflicts will be displayed either in the Sync View window (default) or in a window. You can specify where to display conflicts through the Update/Merge Preference window.
- ◆ **Preview all incoming changes before updating:** All changes will be displayed in either the Sync View window or a window (depending on your settings).
- ◆ **Never preview and use CVS text markup to indicate conflicts:** This option will automatically merge all changes without any user interaction. Conflicting changes will be merged in using the CVS text markup:

```
<<<<<< original file revision
[original code]
= = = = = = =
[incoming code]
>>>>>> incoming file revision
```

Note: You should not configure Update to use the **Never preview and use CVS text markup to indicate conflicts** option because this will corrupt the repository with CVS text markups.

Limitation: After doing **Team > Update**, new descendants become unresolved in the model although they exist on the hard disk. To remedy this situation, after doing the update, right-click the unresolved unit in the Rational Rhapsody browser and select **Load with descendants**.

Configuring how Update behaves


To configure how the **Update** operation behaves:

1. Open the Preferences window. Choose **Window > Preferences**.
2. Expand **Team > CVS > Update/Merge** and make your selection on the Update/Merge pane.
3. Click **OK**.

Adding a unit created in Rational Rhapsody to the CVS repository

When you create a new unit in Rational Rhapsody, you can add it to the repository.

To add to version control:

1. Make sure that you save your Rational Rhapsody model so that Unit View is refreshed to contain this new unit as well. The new unit should appear in Unit View with an icon with a question mark overlay  to signify that it is new file that is not added to version control yet.
2. Right-click the unit to be added and select **Team > Add to Version Control**.

Showing the history of a unit in CVS

To view the history of a Rational Rhapsody unit in version control:

1. Open the History window for the unit. Right-click the unit in the Unit View window and select **Team > Show History**.
2. If you want to get content for a particular revision, right-click the revision and select **Get Contents**.
3. From the History window, you can also compare different revisions. For example, you can click the **Invoke Rhapsody DiffMerge to compare and merge manually** button on the Compare Editor in Eclipse. The Rational Rhapsody DiffMerge tool will open for you to do the compare/merge of Rational Rhapsody elements.

Subversion (SVN)

This subject provides some information that helps you get started with using the Rational Rhapsody/Subversion integration in Eclipse. Subversion (SVN) is an open source configuration management (CM) tool. The Rational Rhapsody Platform Integration lets software developers work on a Rational Rhapsody project within the Eclipse platform.



For details about how to use Eclipse and Subversion, see the documentation provided for those products.

Sharing a Rational Rhapsody project in Subversion

This topic assumes you know how to create a Subversion repository and connect to an existing Subversion repository.

To share a Rational Rhapsody project in Subversion:

1. Switch to the Rational Rhapsody Unit View.
 - a. In Eclipse, open the Show View window. Choose **Window > Show View > Other**.
 - b. Select **Rhapsody > Unit View** and click **OK**.
2. In the Unit View window, right-click the project and then select **Team > Share Project** to open the Share Project window, which has various views.
3. On the Enter Repository Location Information view of the Share Project window, enter the repository location information and then click **Next**.
4. On the Enter Module Name view of the Share Project window, make sure the **Use project name as module name** radio button is selected and then click **Next**.
5. Enter a commit comment on the Ready to Share Project view of the Share Project window and click **Finish**.
6. Switch to the Team Synchronizing perspective of Eclipse. Choose **Window > Show View > Other** to open the Show View window, and then select **Team > Synchronize**.

7. Notice the new units marked with arrow overlay  Rh . To add those units into the repository, right-click the project and select **Commit**.

You can now switch to the Eclipse perspective to Rational Rhapsody Modeling to keep working on your Rational Rhapsody model.

Checking out a Rational Rhapsody project from a Subversion repository

To check out a Rational Rhapsody project from a Subversion repository:

1. Start Eclipse.
2. Open the New Project window. Choose **File > New > Project**.
3. Expand the **SVN** folder, select **Checkout Projects from SVN** and then click **Next**.
4. If you have not already connected to your Subversion repository in this workspace, select **Create a new repository location** and then click **Next**.
5. Select the Rational Rhapsody project that you want to check out and then click **Finish**.

You can now switch to the Rational Rhapsody Modeling perspective in Eclipse to work on the Rational Rhapsody project you have checked out.

Collaboration with other users in Subversion

You can use other related CM operations while working on a project that you already checked out so that you can successfully collaborate with other users working on the same Rational Rhapsody project.

Repository synchronization in Subversion

In order to check if a Rational Rhapsody unit is still in sync with the repository after you checked it out, right-click the unit in Unit View window and select **Team > Synchronize with repository**.


- ◆ If there are no differences between the workspace unit and the remote, a message box tells you that there are no changes between the workspace resource and the remote.


You can do an [Updating a Rational Rhapsody unit in Eclipse to the Subversion repository](#) to get the incoming changes.


- ◆ If this message does not appear when you synchronize with the repository, it means that there are some differences between the workspace unit and the corresponding repository unit. This means either you modified this unit and/or somebody else has committed some changes on this unit to the repository.


In this case, you can click the **Invoke Rhapsody DiffMerge to compare and merge manually** button to launch Rational Rhapsody DiffMerge. After you merge, you should right-click on the unit and select **Mark as Merged**.


If a Rational Rhapsody unit is out-of-sync with the repository, one of the following overlays will appear on the icon for the unit:

 means you have created this unit and it is not added to the repository yet (outgoing new file)

 means the repository contains this new unit that does not exist in your workspace (incoming new unit)

 means this unit is changed in your workspace (outgoing modification)

 means this unit is changed in the repository (incoming modification)

 means this unit is modified both in the repository and in your workspace so there is a conflict

Updating a Rational Rhapsody unit in Eclipse to the Subversion repository


While you are working on a Rational Rhapsody unit in Eclipse, other members of your team might have committed changes to the copy of the unit in the repository. To get these changes, you can update your Rational Rhapsody unit to match the repository. You can do this in the following ways:

- ◆ Right-click the unit in the Unit View window and select **Team > Update**.
- ◆ Switch to the Synchronization perspective. Right-click the unit, and select **Update**.

Adding a unit created in Rational Rhapsody to Subversion repository

When you create a new unit in Rational Rhapsody, you can add it to the repository.

To add to version control:

1. Make sure that you save your Rational Rhapsody model so that Unit View is refreshed to contain this new unit as well. The new unit should appear in Unit View with an icon with a question mark overlay  to signify that it is new file that is not added to version control yet.
2. Right-click the unit to be added and select **Team > Add to Version Control**.

Showing the history of a unit in Subversion

To view the history of a Rational Rhapsody unit in version control:

1. Open the History window for the unit. Right-click the unit in the Unit View window and select **Team > Show History**.
2. If you want to get content for a particular revision, right-click the revision and select **Get Contents**.
3. From the History window, you can also compare different revisions. For example, you can click the **Invoke Rhapsody DiffMerge to compare and merge manually** button on the compare editor in Eclipse. DiffMerge will open for you to do the compare/merge of Rational Rhapsody elements.

IBM Rational Team Concert

The IBM® Rational Team Concert™ collaborative software delivery environment uses the IBM® Rational® Jazz™ technology platform, enabling collaborative teams to work together. You can use the Rational Rhapsody Platform Integration if you have Eclipse, Rational Rhapsody with plug-ins, and Rational Jazz/Rational Team Concert.

Rational Jazz is an initiative to transform software delivery to make it more collaborative, productive, and transparent. Rational Team Concert is a collaborative work environment for developers, architects, and project managers with workitem, source control, build management, and iteration planning support. The Rational Rhapsody Platform Integration lets software developers work on a Rational Rhapsody project within the Eclipse platform.

For details on how to use Rational Jazz, Rational Team Concert, and Eclipse, see the documentation provided for those products.

How changes are accepted and conflicts resolved

Each team member has an Eclipse Rational Team Concert workspace connected to the Rational Jazz server and works with Rational Rhapsody projects that are acquired from the Rational Jazz repository. When a team member initiates acceptance of changes made by other team members, Rational Team Concert replaces the local Rational Rhapsody units with ones fetched from the Rational Jazz repository. However, if you have already modified some of these Rational Rhapsody units, Rational Team Concert merges local and incoming units. The Rational Rhapsody DiffMerge tool is used to perform the merge.

When Rational Team Concert detects that some of the local units cannot just be replaced with new ones, it reports a conflict and it gives you the opportunity to try to merge automatically or to merge later. The following options are possible.

- ◆ **Auto-Merge.** In this case, Rational Team Concert launches the Rational Rhapsody DiffMerge tool and, if it is possible, automatically merges differences in silent mode. However, if there are any conflicts from the Rational Rhapsody DiffMerge perspective, Rational Team Concert displays a message box stating that auto-merge is not possible for some of the files and suggests you do the merge manually for them. All units that could not be merged automatically are put into the Rational Team Concert **Merge required** list, for which you must do a manual merge. See the next bullet for how to do this.
- ◆ **Merge Later.** When you click this button, whatever that can be replaced is replaced (without having to open the DiffMerge tool). All conflicting units are put into the Rational Team Concert **Merge required** list. You will need to manually merge them later. Note that among those units there might be differences that can be auto-merged.

To merge units that were postponed (either due to the inability of the Rational Rhapsody DiffMerge tool to merge them automatically or because you decided to merge conflicts later), you can right-click **Merge required** in Rational Team Concert and select **Open in compare editor** to open the Compare Editor window.

- You can then try (or retry) to merge the conflicts automatically by clicking the **Auto-Merge** button (to call the DiffMerge tool in silent mode).
- You can click the **Invoke Rhapsody DiffMerge to compare or merge manually** button so that you can resolve conflicts by merging manually. You must save your changes in the DiffMerge tool and click the **Resolve as Merged** button to commit the results when you return to the Compare Editor window.

For more information about the DiffMerge tool, see [Parallel development](#).

Index

Symbols

.magic file (Rational ClearCase) 156

A

Activity diagrams 18
 as units 20
 to define a use case 15
AddToArchiveAfterCreateUnitActivation property 161
AllLeftItemMerge preference 123
AllRightItemMerge 123
Application 16
 CM 44
 rapid prototyping 62
Architect 5
Architecture 16
Archive 42, 49, 165
 adding to 49
 batch mode 43, 44
 cannot update 36
 CM tool 44
 connect to 136
 connecting to 41, 44
 connecting to different 43
 connecting to PVCS Dimensions 164
 creating files 29
 directory structure changes 152
 list 167
 multiple 21
 operations 56
 PVCS Dimensions 165
 Rational ClearCase adding to 149
 Rational Synergy 136
 showing items 41, 44
 tools 11, 63, 64
 units for comparison 64
 updating 27
AssignedTasksItsTaskId 137
AssignedTasksItsTitle property 137
Attributes 80
 differences 81
 pane 71
 text 102
Autosave file 13
Autosynchronize 46

B

Backup file 13
Base-aware comparisons 68, 70, 73, 96
 automatic merging 106
 command-line 129
 Diff icons 78
 example of command-line 130
 merge icons 107
 merging CM branches 109
 reporting preferences 119
 resolving conflicts in 109
 show DiffMerge marks preference 122
 trivial versus non-trivial differences 106
BaseAwareAutoMergeableAttributes preference 127
BaseAwareAutoMergeInvocation preference 127
BaseAwareDiffAttrChanged preference 119
BaseAwareDiffAttrChngBoth preference 119
BaseAwareDiffAttrDelAndChng preference 119
BaseAwareDiffElemAdded preference 119
BaseAwareDiffElemChanged preference 120
BaseAwareDiffElemChngBoth preference 120
BaseAwareDiffElemDelAndChng preference 120
BaseAwareDiffElemDeleted preference 120
BaseAwareDiffInvocation preference 127
BaseAwareDiffMergeAutoNo preference 120
BaseAwareDiffMergeAutoYes preference 120
BaseAwareDiffMergeInvocation preference 127
BaseAwareDiffReportFooter preference 120
BaseAwareDiffReportHeader preference 120
BaseAwareDiffSideLeft preference 120
BaseAwareDiffSideRight preference 120
BaseAwareDiffTrivialNo preference 120
BaseAwareDiffTrivialYes preference 120
BaseAwareTextDiffMergeEnabled preference 127
Batch mode 39, 163
 actions 56
 Connect to Archive 43, 44
 creating directories in 29
 deleting files 33
 DiffMerge command line 129
 keyword expansion 30
 Linux 11, 64
 moving a file or directory 35
 renaming a file 34
 renaming a package 35
 troubleshooting 54

Index

- versus SCC mode 39
- BlinkWalkingThroughDiffs preference 85, 117
- Browse From Here browser 76
- Browsers 37, 65
 - Browse From Here 76
 - DiffMerge 71, 76, 125
 - DiffMerge differences 76
 - DiffMerge symbols 77
 - DiffMerge tool navigation 79
 - icons 65
- C**
- Case sensitivity 149
- Check in 42, 48
 - PVCS Dimensions 167
 - Rational ClearCase 150
 - Rational Synergy task 138
 - SCC mode 167
- Check out 42, 47
 - PVCS Dimensions 166
 - Rational ClearCase 150
- Check Out Branch 41
- CheckOutCheckInDirectoryOnceDuringAddToArchive property 160
- CI (configuration items) 18
- Classes 16
 - as units 20
 - automatic unit creation 18
 - divide model based on 16
 - reference from other project 6
- ClassIsSaveUnit property 20
- CLS file 12
- CM operations 49
 - check in 48
 - check out 47
 - Connect to Archive 44
 - DiffMerge tool 45
 - fetch 50
 - history 51
 - List Archive 44
 - list of supported 41
 - Lock 50
 - Properties 45
 - Synchronize 46
 - undo check out 51
 - unlock 50
 - version tree 51
- CM status 52, 53
- CM tool 41
 - comparing units stored in 67
 - configuring 43
 - extend interface 58
 - messages 55
 - pre- and post-actions 56
 - PVCS Dimensions in SCC mode 163
 - Rational ClearCase 148
 - Rational Synergy 133
 - troubleshooting common problems 53
 - unable to update 36
- CMConflictResolution property 36
- CMHeaderItsVersion property 167
- CMOperationEndSeparator 55
- CMOperationStartSeparator 55
- CMTool property 135
- Code generation 17
 - improving performance 158
 - incremental 12
- Collaboration 1
 - designing for 15
 - methodologies 6
 - multiple site 6
 - remote 6
 - sharing by copying 7
 - using CM tools 39
 - using Web-enabled devices 61
- Colors 19
 - codes for DiffMerge categories 75
 - DiffMerge settings 119
 - DiffMerge showing differences 72
- Command line 32
 - DiffMerge options 129
 - launch the external tool 80
- Comments 19
- Comparing 63
 - three units 73
 - two units 72
 - units 45
 - units graphically 82
- Complexity of projects 1
- Components 2
 - organizing by 17
 - scope 17
- Concurrent Versions System (CVS) 171
 - adding unit created in Rational Rhapsody to repository 175
 - checking out a Rational Rhapsody project 172
 - collaborating with other users 173
 - repository synchronization 173
 - sharing a Rational Rhapsody project 171
 - updating a Rational Rhapsody unit 175
 - viewing unit history 175
- Configuration Items 136
- Configuration items (CIs) 18
- Configuration management (CM) 11, 63, 64, 163
 - merging from a base-aware comparison 109
 - Rational Synergy 133
 - SCC compliant 163
 - with Rational ClearCase 148
- Configurations
 - items accessing 40
 - items dividing projects 18
 - of the CM tool 43
 - system manager 4

-
- Connect to Archive 41
 - in batch mode 44
 - PVCS Dimensions 164
 - Content management 3
 - CVS 171

 - D**
 - DAT file 12
 - DefaultDirectoryScheme property 26
 - Delete 33
 - DeleteActivation property 33
 - Descendant 66
 - adding to an SCC archive 165
 - checking in 167
 - fetching 168
 - Design 3
 - Designing
 - for collaboration 15
 - for readability 19
 - Developer 5
 - DiagramIsSaveUnit property 20
 - Diagrams 19
 - as unit 20
 - comparing 76
 - file 18
 - graphically merging 111, 112
 - storing as units 19
 - DiffAttributesFilter preference 124
 - DiffColor preference 119
 - Differences
 - filtering 81
 - graphical 88
 - high level view 98, 100
 - logical 88
 - major 98, 100
 - propagated view 98
 - reporting 74
 - trivial versus non-trivial 106
 - DiffInvocation preference 128
 - DiffMerge tool 7, 21, 23, 41, 63
 - advantages of launching inside Rational Rhapsody 67
 - advantages of launching outside Rational Rhapsody 69
 - Attributes pane 71
 - attributes pane 79
 - base-aware icons 78
 - base-aware mode 68, 70, 73
 - browser 71, 76, 77, 125
 - browser navigation 79
 - changing preferences 117
 - color coded categories 75
 - color preferences 119
 - command-line options 129, 130
 - compare archived unit with current 66
 - compare archived units 67
 - comparing diagrams 76
 - comparing units 64
 - configuration management tools 64
 - descendant 66
 - Diff text option 80
 - Difference Report 85
 - Difference Report preferences 130
 - difference symbols 77
 - Diffmerge.ini 101
 - display difference in red 72
 - examining selected file paths 70
 - exporting reports 90
 - external textual tool 79, 118
 - filtering comparisons 81
 - for collaboration 45
 - graphical comparisons 82
 - graphically merging diagrams 111, 112
 - highlight graphical differences 83
 - how performs a model comparison 96
 - interface 71
 - keywords for command line 32
 - keywords for preference values 118
 - launch from Rational ClearCase 153
 - launching inside Rational Rhapsody 64
 - launching outside Rational Rhapsody 68
 - left value 68, 70, 72
 - limitations 87, 97, 105, 153
 - making merge decisions 108
 - merge icons 107
 - merge output setting 128
 - merging menu options 109
 - merging sequence diagrams 114
 - merging units 101
 - nested difference icon 78
 - nested differences in subelements 77
 - nested units 68
 - Output window 74
 - parallel development 63
 - preference keywords 118
 - preferences 117
 - print report 86
 - Printing 64
 - process 91
 - properties 128
 - Rational ClearCase 156
 - Rational Team Concert 182
 - renamed elements 92
 - renaming support 121
 - report preferences 119
 - reporting differences 74
 - resolving conflicts in base-aware comparisons 109
 - right value 68, 70, 72
 - running from command line 129
 - saving merged unit 105
 - selecting units to compare 68
 - setting Rational ClearCase tools to launch 127
 - starting a merge 101
 - suppress graphical differences 86
-

- suppression preferences 124
- switch off graphics blinking 85
- text merge 127
- trivial/non-trivial differences 106
- using with Rational ClearCase 151
- walk-through graphical differences 83
- with CM tools 45
- with Rational Synergy 139
- DiffMergeInvocation preference 128
- DiffPrefix preference 120
- Directories 29
 - creating 29
 - creating in SCC mode 29
 - moving 35
 - of package 34
 - structure 152
- Distributed team 4, 22, 63
- Dividing
 - model based on classes 16
 - project in two projects 22
 - project into units 18
 - use cases 15
- Domains 16
 - organizing by 16
 - Rational ClearCase VOB 158

E

- Eclipse 156, 171, 177, 181
- Editing
 - files 9
 - units 33
- EHL file 12
- ElementMatchRule preference 121
- Elements 21
 - code generation for 17
 - how DiffMerge makes a match 91
 - merging of renamed 104
 - merging referring 115
 - referenced model 105
 - renamed 92
 - reuse of 17
 - sequence diagram merging 115
 - sequence diagrams 116
 - testing 17
- Errors 12
 - failed to open document 12
 - failed to save document 12
 - unable to create process 54
 - unable to rename package 35
 - unable to store package in new directory 34
- Events 19
- Events history list 12
- Evil twins issue (Rational ClearCase) 150
- ExcludeFromMerge preference 123
- ExcludeGraphTypesVLess6 preference 125
- External textual tool 79, 80, 118

F

- Features window
 - configuration management properties 139, 159, 170
 - property definitions 152
 - PVCS Dimensions properties 170
 - Rational ClearCase properties 159
 - Rational Synergy properties 139
- Fetch 42, 50
 - PVCS Dimensions 168
 - supported mode 39
- Files 6
 - autosave 13
 - backup 13
 - base-aware comparisons 68, 70, 73
 - checking in and out 10
 - creating 29
 - DAT 12
 - Diffmerge.ini 101
 - editing 9
 - EHL 12
 - locking 10
 - LOG 12
 - management by reference 8
 - management in CM tool 10
 - moving 35
 - permissions 50
 - project 21
 - renaming 34
 - RPW (workspace) 12
 - RPY 12, 21
 - RPY dividing projects 18
 - sharing 7
 - sharing by copy 7
 - sharing by reference 9
 - table 12
 - types 12
 - units 12
 - VBA 12
- Filtering 9, 81
- Flat repository 23, 25
- Framework 16

G

- Geographical distribution 4
- GetCurrentTaskItsTaskId property 137
- Graphical differences 76, 82, 88
 - highlight 83
 - ignored in merge 86
 - suppressing 86
 - walk-through 83
- Graphically merging diagrams 111, 112, 115
 - tips 113
- GraphicalMerge preference 123
- Groups 17

H

Hierarchical repository 23, 26, 152
High level differences 98, 100
History 42, 51, 169

I

Icons 65
 base-aware comparison 78
 Diffmerge browser 77
 DiffMerge trivial vs. non-trivial 73
 merge 107
IgnoreGraphDiffs preference 125
IncludeInMerge preference 123
Integrator 5
Interface, custom 10
ItemMerge preference 123

K

Keywords 30
 administrative 30
 arguments to CM commands 32
 DiffMerge preferences 118
 DiffMerge tool 32
 expansion in CM batch mode 30

L

Layout 19
Lead developer 5
LeftItemMerge preference 123
LeftMerge preference 123
LeftOnlyColor preference 119
LeftOnlyPrefix preference 120
Licensing
 Rational ClearCase 159
Limitations
 CVS 174
 DiffMerge tool 87, 97, 105, 153
 graphical merging 105
 merged units 105
 no CM version number display 11
 Rational Rhapsody with Rational ClearCase 150
 read-only diagrams 105
 UNIX 153
Linux 11, 64
 DiffMerge with Rational ClearCase 156
List Archive 44
 PVCS Dimensions 167
Lock 42, 50
LOG file 12
Logical differences 88

M

Macros (VBA) 12
Major structure differences 98, 100
MakeCMSShadowDirActivation 29
Makefile 22
mergeLog option 131
MergeOutput preference 128
MergeToRhapsody preference 123
Merging 101
 activity log preferences 123
 automatic 106
 automatic resolve preference 122
 base-aware 106
 CM branches 109
 diagrams graphically 111, 112, 115
 graphical limitations 105
 icons 107
 include graphical differences 86
 log of 116
 manually 108
 navigation for manual 109
 referenced model elements 105
 renamed elements 104
 report on 116
 saving merged unit 105
 sequence diagram elements 114, 115
 sequence diagrams 114
 starting 101
 text 102
 trivial versus non-trivial differences 106
 two units 110
 units 45
 using command line options 131
Message 55
Methodology 6
 collaboration 6
 combining 6
 sharing by copy 7
 sharing by reference 6, 8
 using a CM tool 6
mkelem 148
mkview 148
mkvob 148
Mode 29
 batch creating directories in 29
 batch keyword expansion 30
Model comparisons 96
Models 16
 framework organization 16
 organizing and partitioning 15
 organizing by domains 16
 testing 17
 upgrading 157
mount 148
MoveActivation property 35
MoveDirectory property 35

Multiple projects
 issues 21
 workflow 23

N

Nested 66, 67
 differences 78
 units 9, 67, 68
NestedDiffColor preference 119
NestedDiffPrefix preference 120
NestedElementPrefix preference 120
NoDiffColor preference 119
NoDiffPrefix preference 120

O

OMD file 12
Operating system 22
Operation semantics, Rational ClearCase 150
Operations 41
 add to Archive 49
 check in 48
 check out 47
 Connect to Archive 44
 fetch 50
 history 51
 List Archive 44
 list of supported 41
 Lock 50
 Synchronize 46
 undo check out 51
 unlock 50
 version tree 51
Organizing 15
 by components 17
 by domain 16
 by logical and physical architectures 16
 by team members 17
 by use case 15
 model 15
Output window 55

P

PackageIsSaveUnit property 20
Packages 15
 adding elements 21
 as unit 18
 as unit property setting 20
 dividing use cases 15
 in directory 34
 rename 35
Parallel development 63
Partial load 22
Partition model 15
Permission 50

Preferences 117
 AllLeftItemMerge 123
 AllRightItemMerge 123
 base-aware reporting 68
 BaseAwareAutoMergeableAttributes 127
 BaseAwareAutoMergeInvocation 127
 BaseAwareDiffAttrChanged 119
 BaseAwareDiffAttrChngBoth 119
 BaseAwareDiffAttrDelAndChng 119
 BaseAwareDiffElemAdded 119
 BaseAwareDiffElemChanged 120
 BaseAwareDiffElemChngBoth 120
 BaseAwareDiffElemDelAndChng 120
 BaseAwareDiffElemDeleted 120
 BaseAwareDiffInvocation 127
 BaseAwareDiffMergeAutoNo 120
 BaseAwareDiffMergeAutoYes 120
 BaseAwareDiffMergeInvocation 127
 BaseAwareDiffReportFooter 120
 BaseAwareDiffReportHeader 120
 BaseAwareDiffSideLeft 120
 BaseAwareDiffSideRight 120
 BaseAwareDiffTrivialNo 120
 BaseAwareDiffTrivialYes 120
 BaseAwareTextDiffMergeEnabled 127
 BlinkWalkingThroughDiffs 85, 117
 DiffAttributesFilter 124
 DiffColor 119
 DiffInvocation 128
 DiffMerge tool 81
 DiffMergeInvocation 128
 DiffPrefix 120
 DiffReport 119
 ElementMatchRule 121
 ExcludeFromMerge 123
 ExcludeGraphTypesVLess6 125
 filtering difference attributes 81
 GraphicalMerge 123
 IgnoreGraphDiffs 125
 IncludeInMerge 123
 ItemMerge 123
 LeftItemMerge 123
 LeftMerge 123
 LeftOnlyColor 119
 LeftOnlyPrefix 120
 MergeOutput 128
 MergeToRhapsody 123
 NestedDiffColor 119
 NestedDiffPrefix 120
 NestedElementPrefix 120
 NoDiffColor 119
 NoDiffPrefix 120
 PrintLineNumbers 121
 PrintNoDiffLines 121
 PrintSubDiffs 121
 RepDecidedAuto 123
 RepDecidedMan 123

-
- RepElemExcluded 123
 - RepElemIncluded 123
 - RepElemMerged 123
 - RepElemTakenFrom 123
 - RepElemUndecided 123
 - RepFooter 124
 - RepHeader 124
 - RepItemDecided 124
 - RepItemMerged 124
 - ReportFooter 121
 - ReportFooterColor 119
 - ReportHeader 121
 - ReportHeaderColor 119
 - RepSideRight 124
 - ResolveAutomaticallyWhenStartingMerge 122
 - RightItemMerge 124
 - RightMerge 124
 - RightOnlyColor 119
 - RightOnlyPrefix 121
 - RightSideLeft 124
 - SaveMerge 124
 - ShowDMMarksInBaseAwareMode 122
 - ShowMetaInfoInBrowser 125
 - ShowStereotypeInBrowser 125
 - StartMerge 124
 - Suppressions 124
 - SuppressRenamePropagatedDiffs 126
 - SuppressRenamePropogatedDiffs 126
 - TextDiffMerge 127
 - UseDefault 119
 - Printing 64
 - diagrams 64
 - DiffMerge reports 64, 86
 - settings for DiffMerge 119
 - PrintLineNumbers preference 121
 - PrintNoDiffLines preference 121
 - PrintSubDiffs preference 121
 - Process message 54
 - Projects 1
 - adding elements 21
 - by type of CM 3
 - CM status 51
 - combining two 23
 - design complexity 3
 - dividing files 18
 - dividing into units 18
 - dividing one in two 22
 - file 12, 21
 - geographical distribution 4
 - guidelines 1
 - manager 4
 - managing multiple 21, 23
 - properties 21
 - restructuring 23
 - restructuring under CM 27
 - setting up for teams 149
 - size by component 2
 - size by team members 2
 - splitting 22
 - Propagated differences 98
 - Properties 19, 157, 159
 - AddToArchiveAfterCreateUnitActivation 161
 - archived file 45
 - AssignedTasksItsTaskId 137
 - AssignedTasksItsTitle 137
 - AutoSynchronize 46
 - changing default 157
 - CheckOutCheckInDirectoryOnceDuringAddToArchi
ve 160
 - ClassIsSaveUnit 20
 - CMConflictResolution 36
 - CMHeaderItsVersion 167
 - CMOperationEndSeparator 55
 - CMOperationStartSeparator 55
 - CMTool 41
 - command 169
 - configuration management 139, 159, 170
 - DefaultDirectoryScheme 26
 - DeleteActivation 33
 - DiagramIsSaveUnit 20
 - DiffMerge tool 128
 - GetCurrentTaskItsTaskId 137
 - MakeCMSShadowDirActivation 29
 - MoveActivation 35
 - MoveDirectory 35
 - operation 42
 - PackageIsSaveUnit 20
 - project-level 21
 - PVCS Dimensions 170
 - Rational ClearCase 152, 159
 - Rational Synergy 139
 - RedirectOutputToRhapsody 166
 - Rename 34
 - RenameActivation 34
 - RenameDirectory 35
 - RunCMToolCommand 44
 - ShowCMStatus 53
 - StoreInSeparateDirectoryActivation 160, 161
 - supported mode 39
 - UserDefCommand_x 58
 - UseSCCtool 135, 163, 165
 - viewing details 45
 - Prototyping 62
 - PVCS Dimensions
 - archive 165
 - Check in 167
 - Check out 166
 - Connect to Archive 164
 - Fetch 168
 - History 169
 - Properties command 169
 - SCC tool 39
 - setting up access from Rational Rhapsody 163
-

Q

Quality manager 5

R

Rapid prototyping 62

Rational ClearCase 11, 63

 .magic file 156

 adding members 148

 adding new members 149

 adding relevant file to the CM archive after creating a unit 161

 administrators 151

 Batch mode 141, 142

 case sensitivity 149

 checking out files 149

 checking out/checking in a directory once 160

 ClearDiff 127

 ClearDiffMrg 127

 configuration for DiffMerge 151

 consumed license 159

 creating a view 148

 deciding between Batch and SCC modes 145

 DiffMerge for Linux 156

 DiffMerge tool limitations 153

 evil twins issue 150

 in batch mode only 43, 44

 launch the Rational Rhapsody DiffMerge tool 153

 license 159

 limitations with Rational Rhapsody 150

 Linux 11, 64

 moving a file or directory 35

 operation semantics 150

 Rational Rhapsody DiffMerge tool with 151

 removing an existing directory for a package and reconciling its contents 161

 renaming a directory or file 34

 SCC mode 141, 142

 SCC tool 39

 set textual tool for DiffMerge 80

 setting up 148

 storing an existing package in a separate directory 160

 synchronizing workspace 46

 team environments 149

 Type Manager 153, 154, 156

 Unified Change Management (UCM) 146

 VOB mounting point 148

Rational Jazz technology platform 181

Rational Rhapsody 1, 12

 checking out files in Rational ClearCase 149

 Concurrent Versions System 171

 CVS 171

 DiffMerge tool 41, 45

 DiffMerge with Rational ClearCase 151

 Eclipse plug-in 156

 files 12

 launching DiffMerge inside 64

 launching DiffMerge outside 68

 project components 2

 properties 157

 PVCS Dimensions 163

 Rational ClearCase 141

 Rational Synergy 133

 Serena PVCS Dimensions 163

 setting up access to Dimensions 163

 Subversion 177

 SVN 177

 units 63

 upgrading models 157

 using with Rational Synergy 135

 Web pages for customization 62

Rational Rhapsody Platform Integration (Eclipse and Rational Rhapsody) 171, 177, 181

Rational Synergy 11, 133

 checking in Rational Rhapsody work 138

 communication with Rational Rhapsody 133

 comparing an archived unit to the current version 66

 connect to archive 136

 create new task 137

 how names tasks 137

 SCC tool 39

 selection in properties 135

 set up 133

 task properties 138

 tasks in Rational Rhapsody 138

 text differences tool 139

 toolbar in Rational Rhapsody 135

 Type Manager 134, 139

 with Rational Rhapsody 135

Rational Team Concert 181

Recursive option 131

RedirectOutputToRhapsody property 166

Reference 8

 adding by 23

 sharing by 8

 units 59

 unresolved 59

Remote collaboration 6

Removing

 units 19

Rename 34

 files 34

 in CM operation 34

 package 35

Rename property 34

RenameActivation property 34

Renamed elements 92

RenameDirectory property 35

Renaming support in DiffMerge 121

RepDecidedAuto preference 123

RepDecidedMan preference 123

RepElemExcluded preference 123

-
- RepElemIncluded preference 123
 - RepElemMerged preference 123
 - RepElemTakenFrom preference 123
 - RepElemUndecided preference 123
 - RepFooter preference 124
 - RepHeader preference 124
 - RepItemDecided preference 124
 - RepItemMerged preference 124
 - ReportFooter preference 121
 - ReportFooterColor preference 119
 - ReportHeader preference 121
 - ReportHeaderColor preference 119
 - Reports 86
 - differences 74, 86
 - DiffMerge differences 85
 - DiffReport 119
 - exporting DiffMerge 90
 - merge information 116
 - print DiffMerge differences 86
 - RTF 90
 - Repository 23
 - flat 23
 - flat using 25
 - hierarchical 23, 152
 - hierarchical using 26
 - structuring 23
 - RepSideRight preference 124
 - reserve 148
 - ResolveAutomaticallyWhenStartingMerge preference 122
 - RightItemMerge preference 124
 - RightMerge preference 124
 - RightOnlyColor preference 119
 - RightOnlyPrefix preference 121
 - RightSideLeft preference 124
 - Roles 4
 - architect 5
 - configuration system manager 4
 - developer 5
 - integrator 5
 - lead developer 5
 - list of team members 4
 - project manager 4
 - quality manager 5
 - RPW file 12
 - RPY file 12, 21
 - dividing projects 18
 - Run CM Tool 41
 - RunCMToolCommand property 44
- S**
- SaveMerge preference 124
 - SBS file 12
 - SCC mode 11
 - actions 56
 - check in 167
 - Connect to Archive 43
 - creating directories 29
 - deleting files 33
 - moving a directory or file 35
 - property 163
 - renaming a directory or file 34
 - renaming a package 35
 - supported tools 39
 - synchronizing workspace 46
 - troubleshooting 53
 - versus batch mode 39
 - Scope 17
 - Sequence diagrams 116
 - elements 116
 - limitations of graphical merge 105
 - merging 114
 - merging "referring" elements 115
 - merging elements 114, 115
 - Serena PVCS Dimensions 163
 - Setting up
 - comparisons 72
 - local workspace 8
 - Rational ClearCase 148
 - Rational Rhapsody for use with Rational Synergy 133
 - Rational Rhapsody projects 149
 - Type Manager for Rational ClearCase 154
 - Share 8
 - by copy 7
 - by reference 8
 - editing files 9
 - setting up workspace 8
 - Show items in archive 41, 44
 - ShowCMStatus property 53
 - ShowDMMarksInBaseAwareMode preference 122
 - ShowMetaInfoInBrowser preference 125
 - ShowStereotypeInBrowser preference 125
 - Source artifacts 47, 48
 - Source control management (CM) 3
 - selecting files types for 12
 - supported tools 10
 - using conventional tools 10
 - Splitting projects 22
 - StartMerge preference 124
 - startview 148
 - Statecharts 18
 - as units 20
 - nested 113
 - to define a use case 15
 - Status of CM 52, 53
 - StoreInSeparateDirectoryActivation property 160, 161
 - Subelements, nested differences 77
 - Subversion (SVN) 177
 - adding unit created in Rational Rhapsody to repository 180
 - checking out a Rational Rhapsody project 178
 - collaborating with other users 179
 - repository synchronization 179
-

Index

- sharing a Rational Rhapsody project 177
 - updating a Rational Rhapsody unit 180
 - viewing unit history 180
 - Suppressions 124
 - SuppressRenamePropagatedDiffs preference 126
 - SuppressRenamePropagatedDiffs preference 126
 - SVN 177
 - Synchronize 41, 46
 - files modified outside of Rational Rhapsody 46
 - with view 46
- ## T
- Take from left 101
 - Take from right 101
 - Tasks 138
 - checking into Rational Synergy 138
 - create new Rational Synergy 137
 - view Rational Synergy properties 138
 - working with Rational Synergy 138
 - Team 6
 - collaboration 6
 - environment 149
 - member roles 4
 - members 17
 - number of members 2
 - organizing by member 17
 - remote member 6
 - Testing 17
 - Text 102
 - attributes 102
 - differences 106
 - external editor 79, 80
 - merging 102
 - modifying in graphics 113
 - TextDiffMerge 127
 - Trivial versus non-trivial differences 106
 - Troubleshooting 53
 - .magic file (Rational ClearCase) 156
 - batch mode 54
 - Rational ClearCase Type Manager 156
 - SCC mode 53
 - unresolved references 59
 - Type Manager
 - Rational ClearCase 154, 156
 - Rational Synergy 134, 139
- ## U
- UCM 146
 - Uncheck Out 42, 169
 - Undo 33
 - Undo check out 51
 - Unified Change Management 146
 - Units 12, 18, 63
 - added by reference 59
 - attributes 71
 - class 20
 - CM status 51
 - compare archived 67
 - compare graphically 82
 - compare nested 68
 - comparing 45
 - creating 19, 20, 65
 - diagram 20
 - dividing project into 18
 - editing 20, 33
 - examining selections 70
 - merging 45, 101
 - moving 33
 - package 20
 - removing 19
 - selecting to compare 68
 - testing 17
 - UNIX 150, 153, 154, 156
 - Unlock 42, 50
 - Unresolved reference 59
 - Update 27
 - CM system error message 36
 - configuration management (CM) archive 27
 - Use cases 15
 - UseDefault preference 119
 - UserDefCommand_x 58
 - User-defined button 58
 - UseSCCTool & CMTool properties 135
 - UseSCCtool property 135, 163, 165
- ## V
- VBA file 12
 - Version Tree 42, 51
 - View 148
 - VOB mounting point 148
- ## W
- Web collaboration 61
 - Webify Toolkit 61
 - Workflow 16, 17, 23
 - Workspace 8
 - file 12
 - synchronizing 46