Rational. Rhapsody

Getting Started Guide

Rational Rhapsody Getting Started Guide

Contents

Rational Rhapsody basics

Welcome to IBM® Rational® Rhapsody®!

In addition to this Getting Started information, you may use the following to learn Rational Rhapsody:

- Attend training at an IBM training facilities or in your office.
- Participate in a virtual training opportunity
- Read tutorials for different jobs and types of development environments

Rational Rhapsody scope

Software developers, systems engineers and systems architects use Rational Rhapsody to create either embedded or real-time systems. Rational Rhapsody provides a visual design environment to create requirements and model systems using the Unified Modeling Language™ (UML™) diagrams and SysML™ diagrams. Rational Rhapsody allows you to accomplish the following tasks:

- **Analysis -** analyze and derive system requirements, specify architecture, and define structure and behavior.
- **Design -** trace requirements to the design, taking into account architectural, mechanistic, and detailed design considerations.
- **Implemention -** automatically generate code from the analysis model, then build and run it from within Rhapsody.
- **Testing -** simulate the application on the local host or a remote target to perform design-level debugging within simulated views.

Software development in Rational Rhapsody

Software developers may use any of the four supported languages in Rational Rhapsody: C, C++, Ada, and Java. The official sample models demonstrate the software uses in the development process from requirements to creating code quickly and accurately and testing the code. These models can be accessed by any of these methods so that you can examine them in more detail:

- Navigate to the Rational Rhapsody `<version>\Samples` directory in your installation to examine the official Rational Rhapsody sample models.
- Click **Proceed** in the Project Samples area of the Welcome screen.
- Click the Samples icon on the Welcome screen.

Note

The official sample models in the Samples directory (see the **Rational Rhapsody samples** list) are different from the models created when following the instructions in the language-specific tutorials in the Rational Rhapsody documentation set. Some of the models created in the tutorials have the same names as the official product samples, but the tutorials demonstrate different techniques and features for instructional purposes.

All of the models for the four languages show different features of the Rational Rhapsody product to demonstrate the overall scope of the product.

Rational Rhapsody diagrams

Rational Rhapsody diagram types are based on standard UML and SysML, as listed below, with the ability to create domain specific types using the **Project profiles**.

Standard UML diagrams

The following Rational Rhapsody diagrams are the UML standard diagrams available in most development environments and Rational Rhapsody profiles:

- Object model diagram
- Use case diagram
- Sequence diagram
- Statechart
- Activity diagram
- Flow chart
- Structure diagram
- Collaboration diagram
- Component diagram
- Deployment diagram

Systems engineering diagrams

The SysML and Harmony profiles support some of the standard UML diagrams:

- Use case diagram
- Statechart
- Activity diagram
- Sequence diagram

These two profiles also provide additional specialized diagrams:

- Requirements diagram
- Block definition diagram
- Internal block diagram
- Parametric diagram

Starting Rational Rhapsody

To start Rational Rhapsody for developers, click the **Rhapsody** icon or from the *Windows* Start menu select **IBM Rational > IBM Rational Rhapsody** (version #) > **Rational Rhapsody Developer edition** > **Rational Rhapsody Developer for Ada**, **C**, **C++**, or **Java**. The Welcome screen provides quick access to the features listed below. To redisplay this window at any time, select the **Help > Welcome Screen** option.

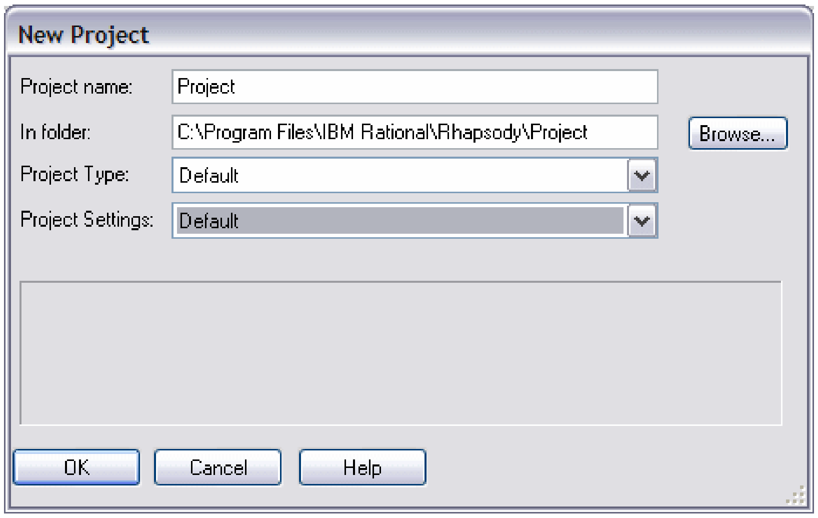
- Create a **New Project** or **Open** a **Project** created previously
- Visit our corporate Web site
- Access product **Documentation**, view the list of available technical **Training** courses, or contact technical **Support**
- View the official Rational Rhapsody **Project Samples** delivered with the system (see Rational Rhapsody samples for a list of the available samples)
- Launch **ReporterPLUS**

Creating a new project

To create a new project:

1. With Rational Rhapsody running, create the new project by either selecting **File > New**, or clicking the **New project** icon in the main toolbar.
2. Replace the default project name (Project) with `MyProject` in the **Project name** field. Enter a new directory name in the **In folder** field or Browse to find an existing directory.

Your dialog box should be similar to this example.



New Project

Project name: Project

In folder: C:\Program Files\IBM Rational\Rhapsody\Project

Project Type: Default

Project Settings: Default

3. The Default **Project Type** provides all of the basic UML structures and is useful for most Rational Rhapsody projects. To create a specialized project, you may select one of the **Project profiles** or create a profile of your own.

4. If you accept the Default **Project Settings**, you intend to use the model centric methodology. If you select a code centric setting, your project will focus on software development through the code instead of using models to generate code.

5. Click **OK**. If the directory does not currently exist, Rational Rhapsody asks whether you want to create it. Click **Yes** to create a new directory for the project.

Rational Rhapsody creates a new project in the MyProject subdirectory and opens the new project. The project name in the directory is `MyProject.rpy`.

See the **Rational Rhapsody guided tour** for descriptions of the areas in the Rational Rhapsody interface.

Project profiles

The project profiles available to you depend on the development language and add-on products licensed for Rational Rhapsody. The software may assign a starting point profile for your project depending on your development language. Generally, profiles provide predefined, domain-specific tags and stereotypes to streamline your development effort. Profiles may serve any of these purposes:

- Support add-on products
- Provide industry-specific details as starting points for the projects
- Manage backward compatibility with previous Rational Rhapsody versions
- Outline a project to meet a standard such as DoDAF or MODAF

Project import and export

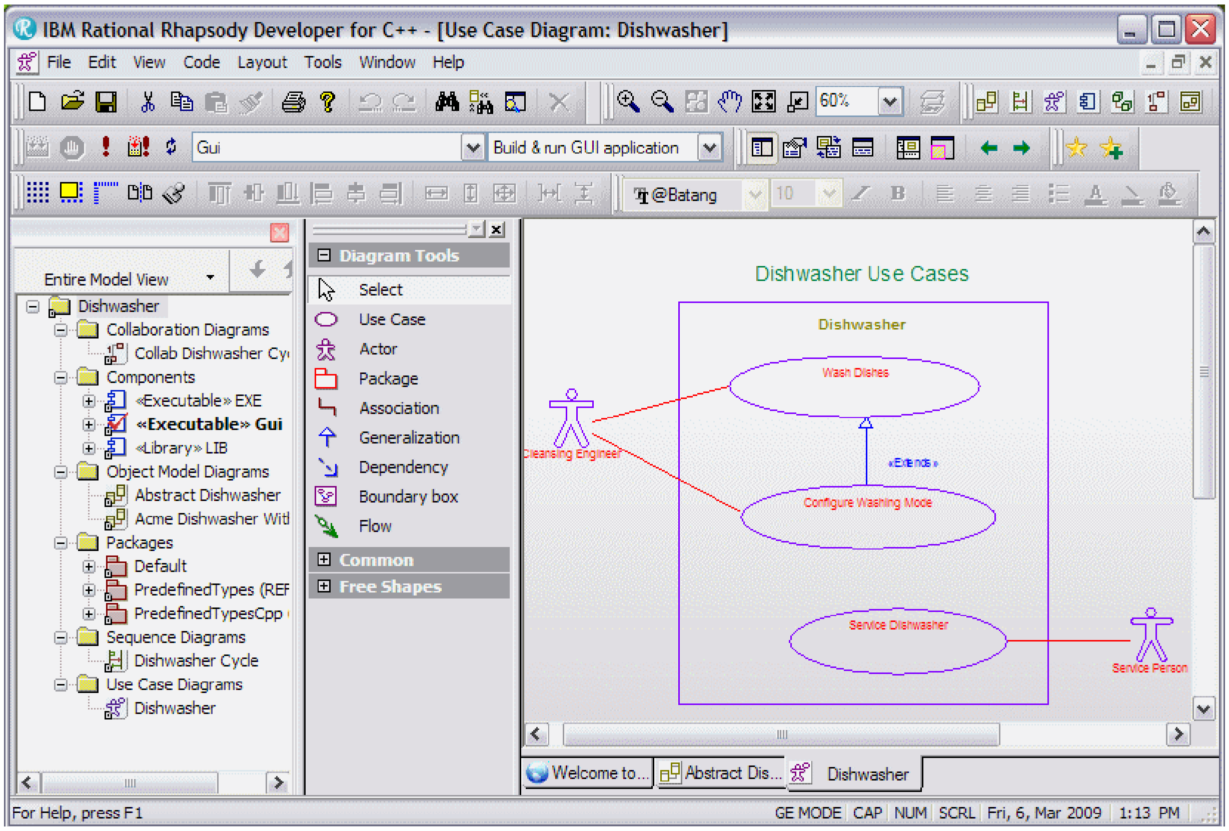
Another Rational Rhapsody technique to streamline work is the facility to import information or other project model components and export Rational Rhapsody projects and components. For example, Rational Rhapsody can import and export the following:

- Projects, models, and diagrams from other modeling tools
- Requirements
- XMI files
- Generated code

For more detailed information, see the Rational Rhapsody user procedures.

Rational Rhapsody guided tour

The Rational Rhapsody stand-alone interface has several areas used to create items in the model and show the relationships of those items. When you create a new project, Rational Rhapsody creates starting point items listed in the *browser* and creates *one drawing area*. These starting point items vary depending on the type of project you created. The **Diagram Tools,** shown in the center of the window in the example, displays the icons appropriate for the type of diagram displayed in the drawing area. Therefore, the icons change when you display a different diagram.



If the browser panel does not appear automatically on the left side of the window, select **View** > **Browser**. This shows the **Entire Model View** (browser) of your new project and operates in the same manner as standard *Microsoft Windows* trees. Click the "+" before a folder to expand the items and view the contents.

For information about Rational Rhapsody displayed in the Eclipse interface, see the **Guided tour of the Eclipse platform integration**.

Location of Rational Rhapsody icons

The icons are grouped in two areas:

- Under the menu bar across the top of the Rational Rhapsody window
- In the center of the window between the browser and the drawing area in an accordion display with the icons and their names listed.

Note

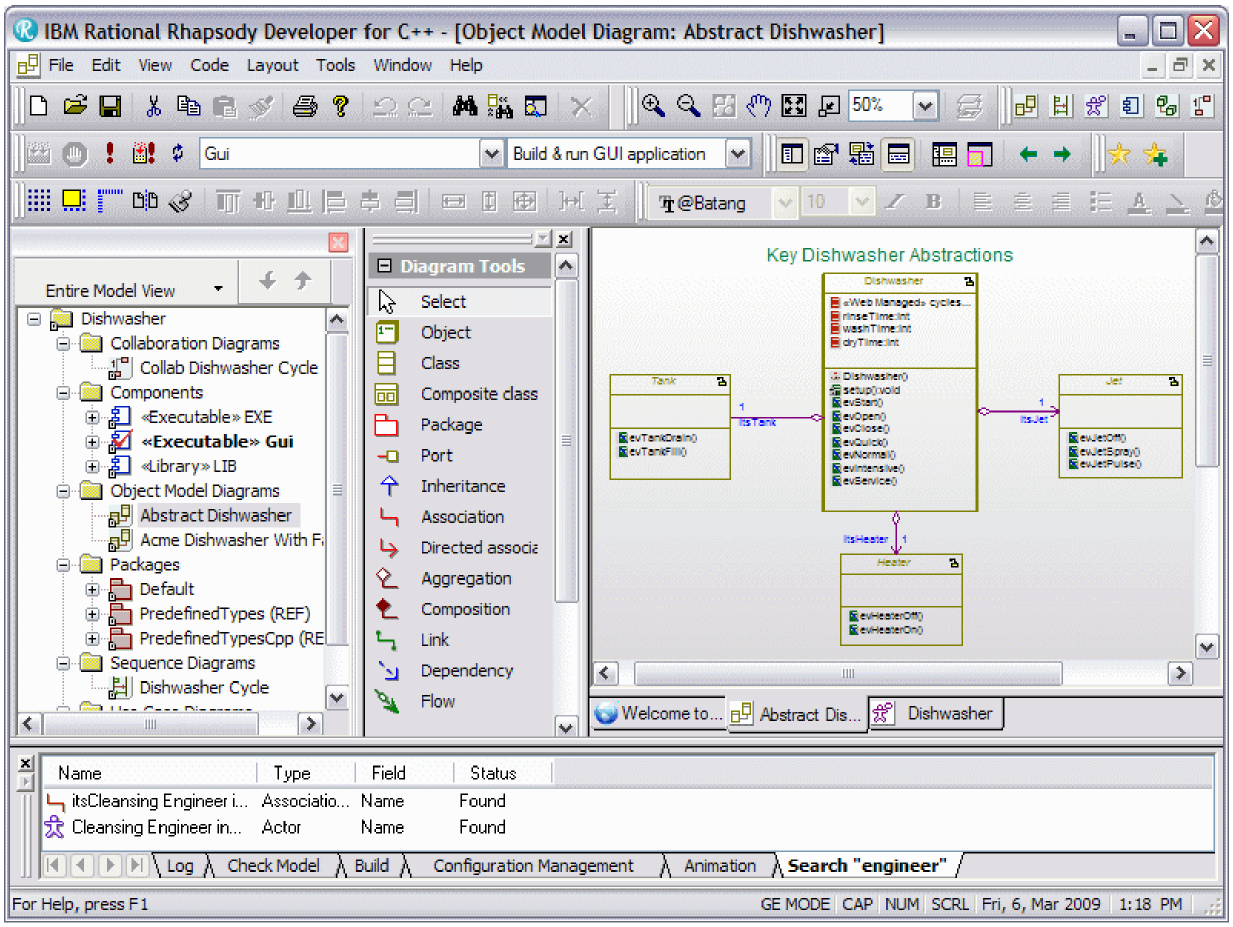
In the upper left corner of the window before the menu bar items is an icon indicating the type of diagram that is currently displayed in the drawing area.

The Drawing icons in the center of the window change depending on the type of diagram being displayed in the drawing area. As in any Windows program, you can rearrange the groups of icons at the top of the Rational Rhapsody window by dragging and dropping the double-bar, left edge of a group of icons.

The icons on the interface perform functions that are also available from menus. For example, you may select the **Tools > Sequence Diagram** menu option or click the diagram creation icon for the Sequence Diagram to create a new sequence diagram.

Output window

When testing your model with the Generate, Make, and Run icons or those **Code** menu options, Rational Rhapsody displays Log output and other messages in the **Output window**. The Output window tabs group different types of information, such as the Search results as shown in the example.



Name	Type	Field	Status
itsCleansing Engineer i...	Associatio...	Name	Found
Cleansing Engineer in...	Actor	Name	Found

When your model requires that messages or an interface be displayed, Rational Rhapsody displays the generated material in a separate window off the Rational Rhapsody interface.

Managing windows

The **Window** menu provides the usual *Microsoft Windows* Cascade and Tiling options. This menu also provides the **Back** and **Forward** options to scroll backwards and forward through the previously displayed diagrams from the current position.

Note

The Back/Forward navigation is not available in Linux.

Rational Rhapsody samples

All Rational Rhapsody installations include a large number of sample projects demonstrating different models. To examine the samples for development environment:

1. Select **File** > **Open**.
2. Navigate to your Rational Rhapsody installation and select the Samples directory to display the samples installation with your software.

If there are no sample directories, install Rational Rhapsody again and select the “Repair” installation type to select the samples for your development environment.

If the C++ sample you want to use contains code for a GUI created in a different development environment than yours, see the instructions in the `Samples\CPP\Readme.txt` file to recompile that code in your environment.

Note

When examining these sample projects, do not save any changes you make to the original sample. Instead, save your version with a different project name and in a different directory. This preserves the original sample with its intended demonstration features and avoids confusion of the standard Rational Rhapsody samples with any altered versions.

The following chart lists all of the available samples with their descriptions.

Rational Rhapsody samples by development environment

Sample Name	Environment	Description
API	C++	This project contains files with executables for RPYReporter, RPYExplorer, and C++ Client. The C++ Client also contains a Write API and a Read API with Debug executables.
Atg	C++	The Vending Machine model in this sample is based on the Automatic Test Generation (ATG) User Guide example. In order to generate the test cases with ATG, you must first manually extend and complete the model.
Cars	C++	The cars.rpy project describes an automated railcar system including the railcar terminal and a GUI for the system. The model contains standard operations for idle cars, passengers boarding cars, selecting destinations, and requesting railcars among other scenarios.
CD_Player	C++	Launch the `CDPlayer.rpy` project file and open the ReadMe.txt in the Files for a full explanation of this sample project. The project includes graphics and these folders: CD_player, Web, HardwarePkg, and CDTools.

Sample Name	Environment	Description
Command Line Interface	C++	This directory contains a sample script to run Rational Rhapsody using the command-line interface.
CORBA	C++	This model contains three projects: Sdm_Observers_factory, Sdm_observers, and client_Sdm_observers to demonstrate using Rational Rhapsody to create CORBA features including a CORBA interface.
DesignPatterns	C++	The project contains samples of common model patterns that can be copied into your projects and modified as needed: • WatchDogPattern • TimeSlicingPattern • StaticPriorityPattern • StaticAllocationPattern • SanityCheckPattern • SafetyExecutivePattern • ProxyPattern • PriorityCeilingPattern • PreemptiveMultitaskingPattern • PollingPattern • MonitorActuatorPattern • MicroKernelPattern • MasterSlavePattern • HomogeneousRedundancyPattern • HeterogeneousRedundancyPattern • HandshakePattern • FixedSizeBlockAllocationPattern • ExecutionControlPattern • DynamicPriorityPattern • CyclicExecutivePattern • BrokerPattern
DiffMerge	C++	This model contains four versions of the Elevator project (original, main trunk, branch, and base) to demonstrate the types of comparisons and results produced from the DiffMerge tool.
Dishwasher	C++	This project defines dishwasher parts and a GUI and contains a sequence diagram, collaboration diagram, and abstract and implementation versions of the dishwasher design.
Elevator	C++	This project contains executables for the GUI, host, and target. It also includes one main use case and several sequence and collaboration diagrams.

Sample Name	Environment	Description
Handset	C++	This mobile telephone project contains Word documents for the overview and requirements. The system requirements are shown with their dependencies and in an analysis package.
hhs	C++	The Home Heating System (hhs) project contains a GUI, object model diagrams, sequence diagrams, and collaboration diagrams.
HomeAlarm	C++	Home alarm security system demonstrating an MFC user interface with separate component builds. This project is also a good example of using multiple sequence diagrams.
HomeAlarmWithPorts	C++	Home alarm security system illustrates the usage of UML 2 ports to specify part interaction points.
Pacemaker	C++	This project contains a GUI, a simulation executable, and detailed packages and diagrams to create this medical device.
Pbx	C++	This Private Branch Exchanges (Pbx) project for telecommunications includes a Statechart, GUI and call router and connection classes with Web-enabled attributes.
PingPong	C++	This simple model demonstrates the events in a ping pong game. The project contains an object model diagram, a sequence diagram, and an animation executable.
PowerWindowWithSimulink	C++	This sample demonstrates how a continuous Matlab\Simulink Block can be used inside a Rational Rhapsody model. This model requires a Simulink license.
Radio	C++	This project contains detailed use cases, sequence diagrams, and GUI, Hardware, and Test executables to perform standard functions of a radio, such as select stations and adjust volume.
ReporterPLUS	C++	This project should be opened in ReporterPLUS to demonstrate producing reports with illustrations for the C++ Dishwasher project.
RequirementsWithTags	C++	This model demonstrates generation of documents using ReporterPLUS in conjunction with tags. Use this model with the template RequirementsTable.tpl.

Sample Name	Environment	Description
TestConductor	C++	These TestConductor samples contain prebuilt GUI components: • CppCashRegister • CppListUsage • CppPbx • CppTestConductorAPI • CppTestingExternalFiles In order to test non-GUI components, apply GMR (generate, make, and run) to the desired executable configuration.
Tetris	C++	The tetris.rpy creates an imitation of the Tetris game including a GUI and hardware package.
UML 2.0	C++	This is a mock-up solution of a generic protocol stack to handle voice and data calls only. The use case diagram shows the functional requirements for the system.
vba	C++	This model demonstrates the uses of VBA with Rational Rhapsody to perform tasks such as generating code, building a model, generating a report, and adding classes to a model. The model contains two demonstration projects: VC_6vbaDemo (vbaSample.rpy) and G3WizardDemo.rpy.
CycleComputer	C	This model uses files with dependencies and statecharts instead of classes and objects as in C++ projects. The design allows the cycle computer to be easily ported to different hardware platforms and to run the following: • With a Visual C++ GUI • With a console using keys R, L and P • With the IDF (automatically simulates key presses)
DiffMerge	C	This model contains four versions of the Elevator project (original, main trunk, branch, and base) to demonstrate the types of comparisons and results produced from the DiffMerge tool.
Dishwasher	C	This project contains a GUI with files, an abstract dishwasher in an object model diagram, and sequence diagrams.
Elevator	C	This project contains an executable GUI, an object model for the host configuration, and one main use case with several sequence diagrams.

Sample Name	Environment	Description
FlowChart	C	The project contains the following basic flowchart patterns that can be used and modified for your projects: • DoWhileLoop(int n) • DoWhileSelfLoop(int n) • IfThenElse() • OrderedIfThenElse() • Sequence() • SimpleIf(char * buffer) • SimpleNegatedIf(char * buffer) • Unstructured() • WhileLoop() • WhileNotLoop()
FunctionalC	C	This is a model of a handset protocol stack using the FunctionalC profile and use case diagrams. The project includes Analysis with comments and Architecture packages.
hhs	C	The Home Heating System (hhs) project contains a prototype executable with a GUI, object model diagrams, and sequence diagrams.
Pacemaker	C	This project contains a GUI, a simulation executable, object model diagrams, and sequence diagrams to create this medical device.
Pbx	C	This Private Branch Exchanges (Pbx) project for telecommunications includes a GUI executable and test scenarios.
Radio	C	This project contains an executable MFCGUI, files, a detailed radio package, an object model diagram, and many use cases. The model performs standard functions of a radio, such as select stations and adjust volume.
ReporterPLUS	C	This project should be opened in ReporterPLUS to demonstrate producing reports with illustrations for the C Elevator project.
S-Function	C	This model uses the SimulinkinC (automatically added profile) and requires a Simulink license.
TestConductor	C	These TestConductor samples contain prebuilt GUI components: • CStopWatch • CPbx In order to test non-GUI components, apply GMR (generate, make, and run) to the desired executable configuration.

Sample Name	Environment	Description
DiffMerge	Java	This model contains four versions of the Elevator project (original, main trunk, branch, and base) to demonstrate the types of comparisons and results produced from the DiffMerge tool.
Dishwasher	Java	This project contains a GUI with files, an abstract dishwasher in an object model diagram, and sequence diagrams.
HomeAlarm	Java	Home alarm security system demonstrating an executable test with detailed packages and shows the use of multiple sequence diagrams.
ReporterPLUS	Java	This project should be opened in ReporterPLUS to demonstrate producing reports with illustrations for the Java HomeAlarm project.
TestConductor	Java	The JavaDishwasher sample contains prebuilt GUI components. In order to test non-GUI components, apply GMR (generate, make, and run) to the desired executable configuration.
CPP	Extensibility Samples for Callback API	This simple application implements the EventListenerPlugin interface and registers with Rational Rhapsody in order to receive callbacks. For more information, see the `readme.txt` file in the Samples/ExtensibilitySamples/CallbackAPI Samples/CPP/EventListernerPlugin directory and the *API Reference Manual*.
RS232	Ada	This project demonstrates design and testing of a simple computer system with a keyboard. The project includes animated sequence diagrams.
RS232_95	Ada	This project demonstrates design and testing of a simple computer system with a keyboard in the “95” version. The project includes animated sequence diagrams.
SPARK	Ada	This model contains two projects to illustrate the SPARK standard support in Rational Rhapsody Ada. The model includes the following special features: • Stack state machine • Stack class as an abstract data type • Monitoring class showing how a data type can be extended.
TestConductor	Ada	The AdaCashRegister sample contains prebuilt GUI components. In order to test non-GUI components, apply GMR (generate, make, and run) to the desired executable configuration.

Sample Name	Environment	Description
Java	Extensibility Samples for Callback API	In the Samples/ExtensibilityChecks/CallbackAPI Samples/Java directory are two subdirectories with three applications in each: • `Plug-in` contains the ApplicationListenerPlugin, CodeGenerationListenerPlugin, and the RoundtripListenerPlugin. • `Stand-alone` contains applications that implement Listener interfaces in order to receive callbacks. For more information, see the `readme.txt` files in the subdirectories containing the application files and the *API Reference Manual*
VB	Extensibility Samples for Callback API	This simple application implements the EventListenerTest interface and registers with Rational Rhapsody in order to receive callbacks. The EventListenerTest API uses Rational Rhapsody to listen for specified events (for example, beforeProjectClose, afterProjectClose, onDiagramOpen, onFeaturesDialogOpen, onCodeGenerationCompleted, and beforeRoundtrip). Use Rational Rhapsody to perform the appropriate actions to cause these events. When one of these events occurs, the API displays a message box indicating the name of the event. For more information, see the `readme.txt` file in the Samples/ExtensibilityChecks/CallbackAPI Samples/VB/EventListeners directory and the *API Reference Manua*l.
VBA	Extensibility Samples for Callback API	This simple application implements the EventListenerTest interface and registers with Rational Rhapsody in order to receive callbacks.As in VB Callback API, the VBA Application Listener Client listens to Rational Rhapsody for specified events (for example, beforeProjectClose, onDiagramOpen, onFeaturesDialogOpen, onCodeGenerationCompleted, and beforeRoundtrip). However, in the VBA sample, the afterProjectClose event is not used since a VBA project is part of a Rational Rhapsody project. If Rational Rhapsody is closed, the VBA project would also be closed. For more information, see the `readme.txt` file in the Samples/ExtensibilityChecks/CallbackAPI Samples/VBA/ VBA_EventListeners_SampleRhpProject directory and the *API Reference Manual.*

Sample Name	Environment	Description
VB	Extensibility Samples for ExternalChecks Samples	This project runs `rpyexternalchecks.exe` to check the model default configuration via the tools model. You can examine the VB source in the project to see how this is accomplished.
Java	Extensibility Samples for ExternalChecks Samples	This Java project provides a use case to check a user defined check. It also contains a .hep file with a path to the check in the Rational Rhapsody project directory, as well as a property pointing to the .hep file.
.settings	Extensibility Samples for Simple Plug-in	This directory contains an Eclipse JDT preferences file. For more information about creating plug-ins for Rational Rhapsody, see the `How to create a plug-in.rtf` document in the Simple Plug-ins directory.
com	Extensibility Samples for Simple Plug-in	This directory contains two sample plug-ins for Rational Rhapsody: • SimplePlugin.class • SimplePlugin.java
APIExtension	Java API	This project demonstrates how to extend the Rational Rhapsody Java-API in a stand-alone application. For more information, see the `readme.txt` file in the Samples/JavaAPI Samples/APIExtension directory and the *API Reference Manual*. To extend the API in a plug-in application, see samples in the Samples/JavaAPI Samples/Plug-in directory.
ClassDumper	Java API	This sample contains a batch file to dump a Rational Rhapsody class. For more information, see the `readme.txt` file in the Samples/JavaAPI Samples/ClassDumper directory and the *API Reference Manual*.
com.telelogic.rhapsody.wfi. rhapsodyListenersExample	Java API	This plug-in demonstrates how to use the Rational Rhapsody Listener extension point to receive notifications on messages and commands from Rational Rhapsody. This plug-in implements “rhapsodyListeners” extension points. For more information, see the `readme.txt` file in the Samples/JavaAPI Samples/ com.telelogic.rhapsody.wfi. rhapsodyListenersExample directory.

Sample Name	Environment	Description
JavaPlug-in	Java API	This diagram formatter sample demonstrates how to create a Java plug-in and how to extend the Rational Rhapsody Java API. Note: You can create a Java plug-in without extending the API, as well as using the API extension in a stand-alone application. For detailed instructions to use this sample, see the `readme.rtf` file in the Samples/JavaAPI Samples/ Plug-in directory.
Adms	Systems	This sample describes the ADMS (Aircraft Defense Management Model) in a Rational Rhapsody project. See the model overview and requirements documents within the project for a complete description of this sample.
Distiller	Systems	The Distiller model uses the SysML profile to reconstruct the case study example, as described in Sandy Friedenthal's SysML Tutorial. The Distiller model is used with the permission of the Object Management Group and is based on the OMG SysML 1.0 Available Specification 07-09-01. See http://www.omgsysml.org/SysML-Tutorial-Baseline-to-INCOSE-060524-low_res.pdf
NetCentric	Systems	This directory contains a model that simulates a network of meteorological weather stations and that provide weather reports of current and forecasted weather conditions for the different locations of the stations. To create this model as a tutorial and then compare your version to the finished version in the directory, see the instructions in the `Net Centric Mini Tutorialv1.1.doc` in this directory.
SysMLHandset	Systems	This is a SysML version of the Rational Rhapsody Handset model.
VB_Post_Simplifier	CustomizeCG	The TestModel subdirectory contains a Rational Rhapsody project for post simplification.

Sample Name	Environment	Description
Statechart_Simplifier_Writer	CustomizeCG	This directory contains four sample directories: • Statechart_ALT_Simplifier - This Rational Rhapsody plug-in demonstrates a user-defined simplifier. The simplifier adds a <full state name> member to the type of the class for each state in its statechart. The attribute type is “int” and a description associates it with the state from which it originated. The simplification is enabled when the user sets the `C_CG::Statechart::Simplify` property to the “`ByUser`” value. • Statechart_Java_Simplifier - This Rational Rhapsody plug-in demonstrates a user-defined simplifier using Java. The simplifier adds a <full state name> member to the type of the class for each state in its statechart. The attribute type is “int” and a description associates it with the state from which it originated. The simplification is enabled when the user sets the `C_CG::Statechart::Simplify` property to the “`ByUser`” value. • Statechart_VB_Simplifier - This Rational Rhapsody plug-in demonstrates a user-defined simplifier using VB. The simplifier adds a <full state name> member to the type of the class for each state in its statechart. The attribute type is “int” and a description associates it with the state from which it originated. The simplification is enabled when the user sets the `C_CG::Statechart::Simplify` property to the “`ByUser`” value. • Statechart_Writer_Rules - This package Statechart_Generation provides all of the rules related to the statechart generation. You must make some modifications to enable this process. For more information, see the individual `readme.txt` files in the four directories.

Search and replace in models

Engineers and developers can use the Search and Replace facility for simple search operations and to manage large projects and expedite collaboration work. The search results display in the **Output window** with the other tabbed information.

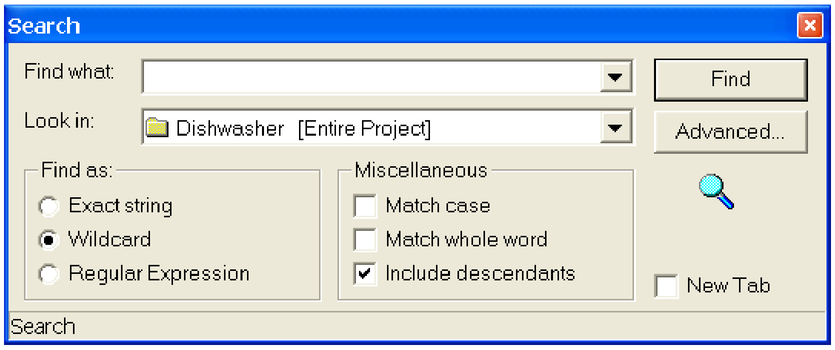
This facility provides the following capabilities:

- Perform quick searches
- Locate unresolved elements in a model
- Locate unloaded elements in a model
- Identify only the units in the model
- Search for both unresolved elements and unresolved units
- Perform simple operations on the search results
- Create a new tab in the Output window to display another set of search results
- For more detailed instructions for the Search and Replace facility.

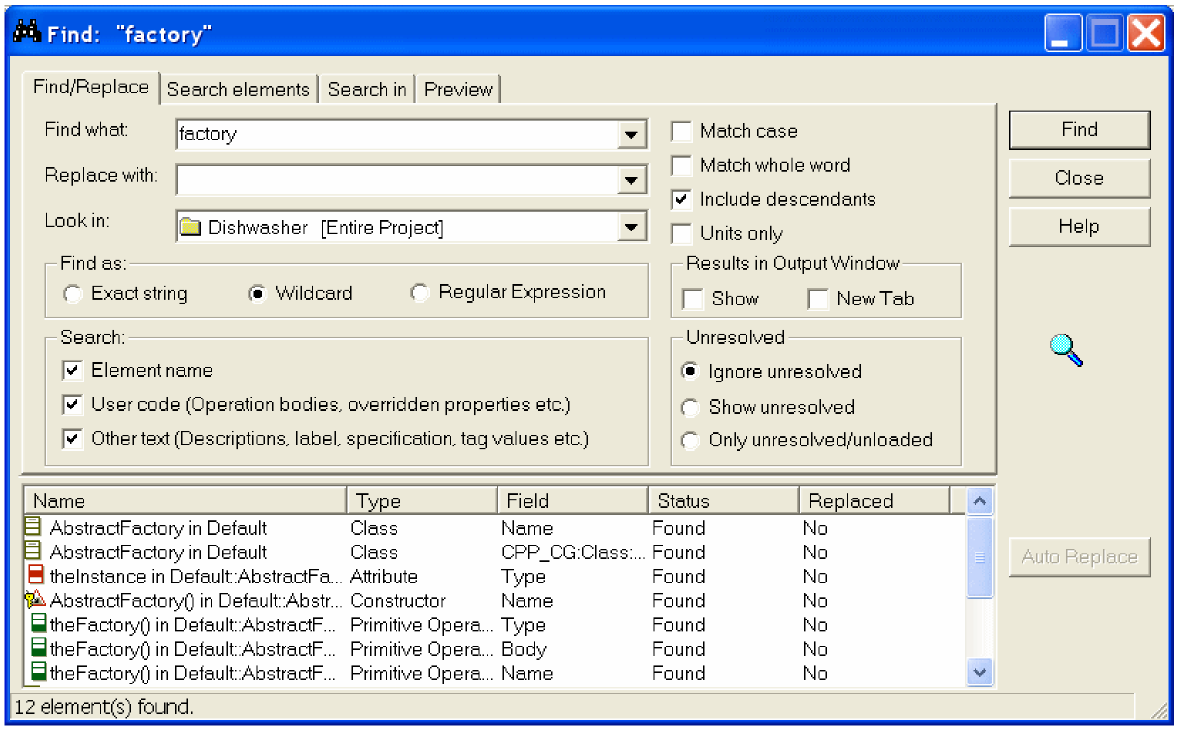
Searching models

To search models:

1. With the model displayed in Rational Rhapsody, there are three methods to launch the Search facility: select the **Edit** menu and then select the **Search** option, click the binoculars icon, or press **Ctrl** + **F**.
2. This displays the Search dialog box to perform a quick search. Type the search criteria into the **Find what** field and click **Find**. The results display in the Output window. The search criteria displays on the Output window Search tab.



3. To display the more detailed search dialog box, select **Edit > Advanced Search and Replace** or click the **Advanced** button in the Search dialog box. Both methods display the Advanced Search dialog box. This dialog box provides the Unresolved and Units only search features.


Find: "factory"

Find/Replace | Search elements | Search in | Preview

Find what: factory

Replace with:

Look in: Dishwasher [Entire Project]

Find as: Exact string [✓ Wildcard] Regular Expression

Search:
- [✓] Element name
- [✓] User code (Operation bodies, overridden properties etc.)
- [✓] Other text (Descriptions, label, specification, tag values etc.)

- [ ] Match case
- [ ] Match whole word
- [✓] Include descendants
- [ ] Units only

Results in Output Window: [ ] Show [ ] New Tab

Unresolved: [✓ Ignore unresolved] Show unresolved Only unresolved/unloaded

Name	Type	Field	Status	Replaced
AbstractFactory in Default	Class	Name	Found	No
AbstractFactory in Default	Class	CPP_CG:Class:...	Found	No
theInstance in Default::AbstractFa...	Attribute	Type	Found	No
AbstractFactory() in Default::Abstr...	Constructor	Name	Found	No
theFactory() in Default::AbstractF...	Primitive Opera...	Type	Found	No
theFactory() in Default::AbstractF...	Primitive Opera...	Body	Found	No
theFactory() in Default::AbstractF...	Primitive Opera...	Name	Found	No

12 element(s) found.

4. You may customize the search criteria using the following:

- **Exact string** permits a non-regular expression search. When selected the search looks for the string entered into the search field (such as char*)
- **Wildcard** permits wildcard characters in the search field such as "*" and produces results during the search operation that include additional characters. For example, the search `*dishwasher` matches class `dishwasher` and attribute `itsdishwasher`.
- **Regular Expression** allows the use of Unix style regular expressions. For example, itsdishwasher can be located using the search term [s]dishwasher.

5. Advanced Search and Replace results do not display in the Output Window by default. To display results in the Output Window, check the **Show** box in the **Results in Output Window** area.

6. If after performing one search you want another Search tab with additional search results displayed in the Output window, check the **New Tab** box in the **Results in Output Window** area. Perform the next search.

Working with search results

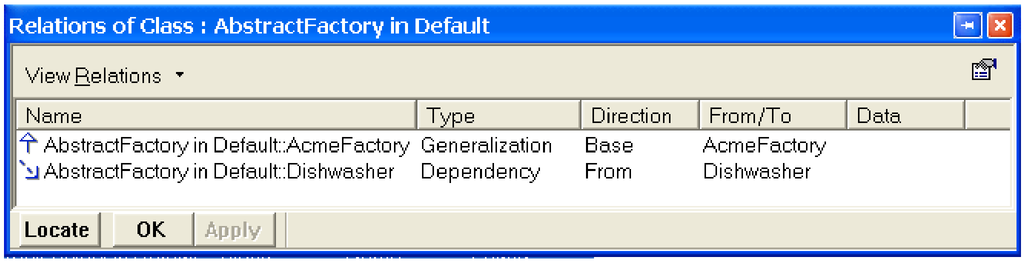
After locating elements using the Search facility, you may perform these operations in the Search dialog box or in the Output window:

- Sort items
- Check the references for each item
- Delete
- Load

To sort items in the list, click the heading above the column to sort according to information in that column.

To examine the *references* for an item in the search results:

1. Highlight an item in the search results list.
2. Right-click to display the pop-up menu.
3. Select **References...** and examine the information displayed in the dialog box, as shown in this example.



Name	Type	Direction	From/To	Data
AbstractFactory in Default::AcmeFactory	Generalization	Base	AcmeFactory	
AbstractFactory in Default::Dishwasher	Dependency	From	Dishwasher	

To delete an item located in search process:

1. Highlight an item in the search results list.
2. Right-click to display the pop-up menu.
3. Select **Delete from Model**. The system displays a message for you to confirm the deletion.
4. Click **Yes** to complete the deletion process.

If you have located an unloaded item in the search results and want to load it into the model, right-click the item. Load the item in the same manner as it is loaded from within the browser.

Replacing

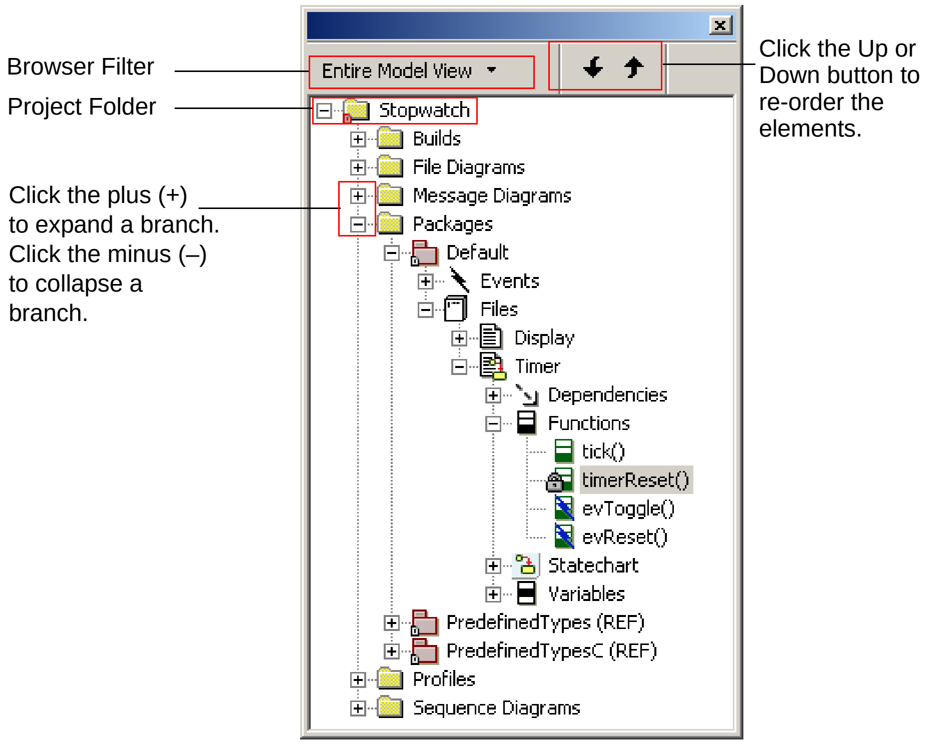
If you want to replace item names or other terminology throughout the model:

1. Display the **Advanced Search**.
2. Enter the current terminology in the **Find what** field.
3. Enter the new terminology into the **Replace with** field.
4. Make any additional selections to limit the search and replace process.
5. Click **Find**.
6. Highlight the results to be replaced and click **Auto Replace**.
7. Click **Yes** to complete the replacement process.

The Rational Rhapsody browser

The Rational Rhapsody browser shows the contents of the project in an expandable tree structure. By default, it is the upper, left section of the Rational Rhapsody interface. The top-level folder, which contains the name of the project, is the *project folder* or *project node*. Although this folder contains no elements, the folders that reside under it contain elements that have similar characteristics. These folders are referred to as *categories*.

The browser displays the automatically generated elements for the project type, as well as the elements that users defined. A project consists of at least one package in the **Packages** category. A package contains UML elements, such as classes, files, and diagrams. Rational Rhapsody automatically creates a default package called **Default**, which it uses to save model parts unless you specify a different package.



Repositioning the browser

To make more room to work on diagrams, you can move the browser outside of the Rational Rhapsody GUI to reposition it as a separate window on the desktop. To reposition the Rational Rhapsody browser, click the bar at the top of the browser and drag it to another desktop location.

Filtering the browser display

The browser filter allows you to display only the elements relevant to your current task. To display the filter menu, click the down arrow button at the top of the browser. Select one of these menu options:

- **Entire Model View** displays all model elements in the browser and is the default view.
- **Use Case View** displays use cases, actors, sequence diagrams, use case diagrams, and relations among use cases and actors.
- **Component View** displays components, nodes, and packages that contain components or nodes.
- **Diagram View** filters out all elements except diagrams.
- **Unit View** displays all the elements that are also units.
- **Loaded Units View** displays only the units that have been loaded into your workspace.
- **Requirement View** displays only those elements with requirements.
- **Overridden Properties View** displays only those elements with overridden properties.

Note

If the browser is filtered, you can add only elements that appear in the current view.

Re-ordering the browser elements

You can re-order the elements in the Rational Rhapsody browser. **Choose View** > **Browser Display Options** > **Enable Ordering** to activate the Up and Down buttons for the browser. Once activated, select an element in the browser and then click the appropriate Up or Down button

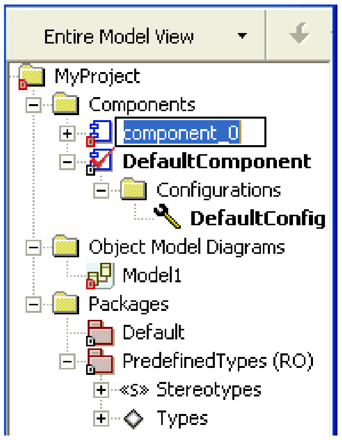
⬇ ⬆.

Adding browser items

To add new items to a category in the browser:

1. Highlight the item in the browser.
2. Select the **Edit** > **Add New <item type>** from the main menu or right-click the browser item and select the **Add New <item type>** from the menu.
3. Depending on the type of item being added, the system may require additional information or it may immediately add a new item to the browser category, as shown with the addition of a new component.

Note: The system allows you type in the name of the new item in the browser list.



Moving and copying browser items

You can move all elements within the browser by dragging-and-dropping them from one location to another, even across packages. You may move a group of highlighted items to a new location in the project by dragging and dropping them into that location in the browser.

You can copy all elements within the browser. To copy an element onto a new owner:

1. Highlight the item in the browser.
2. Press the Ctrl key while dragging the element.
3. Drop it onto the new owner.

Creating a list of favorites

If you want to select a few items from a project for some concentrated work, you may select these items as your Favorites and display them in a separate browser window. To select Favorites:

1. Display the project in Rational Rhapsody.
2. Click the Favorites star icon to display the **Favorites** browser.
3. Highlight the item of interest in the project browser list, and click the star icon with a plus symbol to add the highlighted item to the Favorites.

Deleting browser items

To delete an item from your project:

1. Highlight the item in the browser.
2. Right-click and select the **Delete from Model** from the menu or select **Edit** > **Delete** from the main menu
3. The system asks for confirmation of the deletion operation. Click **OK** to approve it.

Saving the project

Save the project using one of the following methods:

1. Select **Save** or **Save As** from the **File** main menu.
2. Click the **Save** icon 💾 in the main toolbar.

Note

Rhapsody performs an autosave every ten minutes, but does not actually create any files in the project directory until you manually save the project.

Closing the project and exiting Rational Rhapsody

To close the project:

1. Select **File** > **Close**. Rhapsody asks whether you want to save changes to the project.
2. Click **Yes**. Rhapsody saves the project, creates a backup of the previous version (if you set the `General::Model::BackUps` property), and closes the project without exiting.
3. To exit from Rhapsody, select **File** > **Exit**.

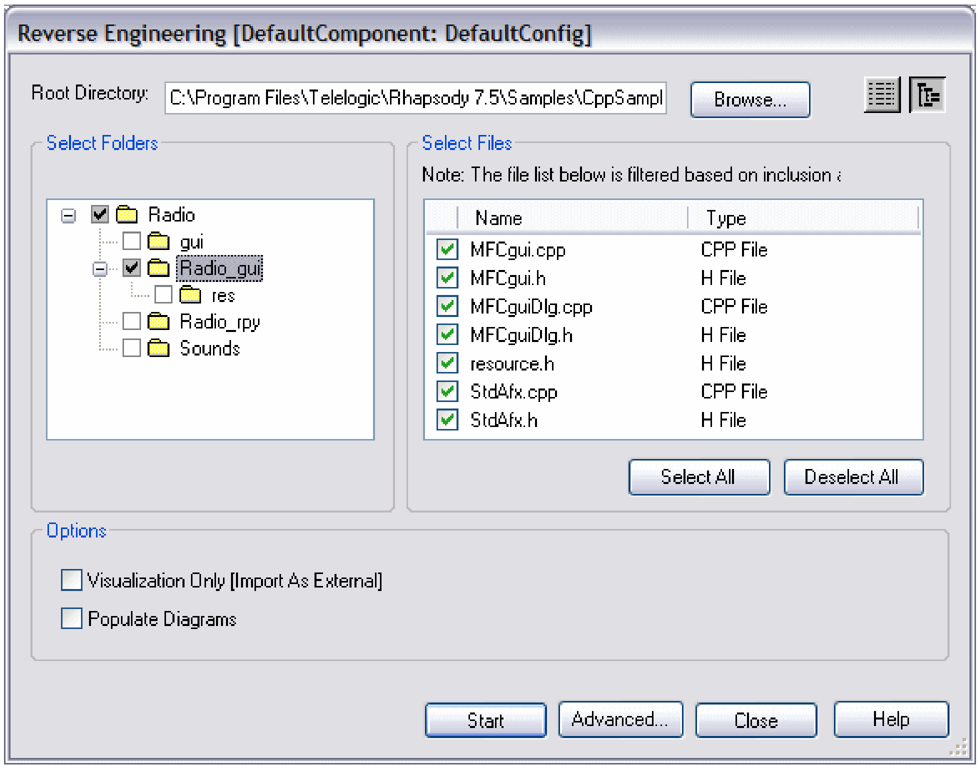
Reverse engineering legacy source code

Companies are always seeking ways to reuse their software assets. A software developer may use Rational Rhapsody to reverse engineer legacy C or C++ source code to provide these capabilities:

- Use the legacy code as a starting point to develop a model-driven design
- Mix model-driven design with external source code
- Redesign the system architecture
- Integrate code generated by another tool
- Integrate with third-party libraries

To import legacy code into Rational Rhapsody, follow these basic steps:

1. Launch Rational Rhapsody and open an existing project or create a new project.
2. Select **Tools** > **Reverse Engineering** and navigate to the source code directory and select the desired files.



Reverse Engineering [DefaultComponent: DefaultConfig]

Root Directory: C:\Program Files\Telelogic\Rhapsody 7.5\Samples\CppSampl

Select Folders
- [✓] Radio
  - [ ] gui
  - [✓] Radio_gui
    - [ ] res
  - [ ] Radio_rpy
  - [ ] Sounds

Select Files

Note: The file list below is filtered based on inclusion a

	Name	Type
✓	MFCgui.cpp	CPP File
✓	MFCgui.h	H File
✓	MFCguiDlg.cpp	CPP File
✓	MFCguiDlg.h	H File
✓	resource.h	H File
✓	StdAfx.cpp	CPP File
✓	StdAfx.h	H File

Options
- [ ] Visualization Only [Import As External]
- [ ] Populate Diagrams

3. Check the **Visualization Only (Import as External)** checkbox in the lower left corner and click **Start**. The dialog boxes used to complete this operation contain multiple options for saving and reusing the selected code.

Parallel development

When many developers are working in distributed teams, they often need to work in parallel. These teams use a source control tool or configuration management (CM) software, such as Clearcase, to archive project units. However, not all files may 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 may 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.

Rational Rhapsody units

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

DiffMerge tool features

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.

The DiffMerge tool can be operated inside and/or outside your CM software to access the units in an archive. It can be launched from inside or outside Rational Rhapsody. It 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

Rational Rhapsody and Eclipse basics

If you are developing software and want to use Eclipse, you may use either of these plug-in integrations:

- **Workflow Integration**: allows you to import a Rhapsody C or C++ project into *Eclipse* using the Rhapsody plug-in for Windows, but not for Linux
- **Platform Integration**: integrates Rhapsody within the Eclipse environment for a single modelling and coding environment for C, C++ or Java. The license required for this feature should be a Rhapsody Developer Multi Language license.

Standard Rational Rhapsody and the platform integration

The standard Rhapsody and the Eclipse Platform Integration both use the same repository so that you may switch between the two interfaces if desired.

The standard Rhapsody interface elements displayed in Eclipse have the same features as in the stand-alone version except that the icons associated with a specific window are displayed at the top of the window.

Creating a Rational Rhapsody project in Eclipse

To create a new Rational Rhapsody project in Eclipse:

1. Open Eclipse.
2. Check the Eclipse **Help > About Eclipse SDK** option to be certain that the Rhapsody icon is displayed, as in the Java example below. If it is not, see the Eclipse setup instructions for Rhapsody.
3. To create a new Rhapsody project within Eclipse, select **File > New > Project**.
4. In the New Project dialog box, select the **Rhapsody Project** wizard from the options and click **Next**.

5. Define the Rhapsody project name, name of the first object model diagram, the development language to be used (C, C++, or Java), and the Rhapsody project type and Rhapsody project settings.

6. Click **Finish**.

Rhapsody creates a new project in the Eclipse workarea and opens the new project.

See the **Guided tour of the Eclipse platform integration** for descriptions of the areas in the Rhapsody interface.

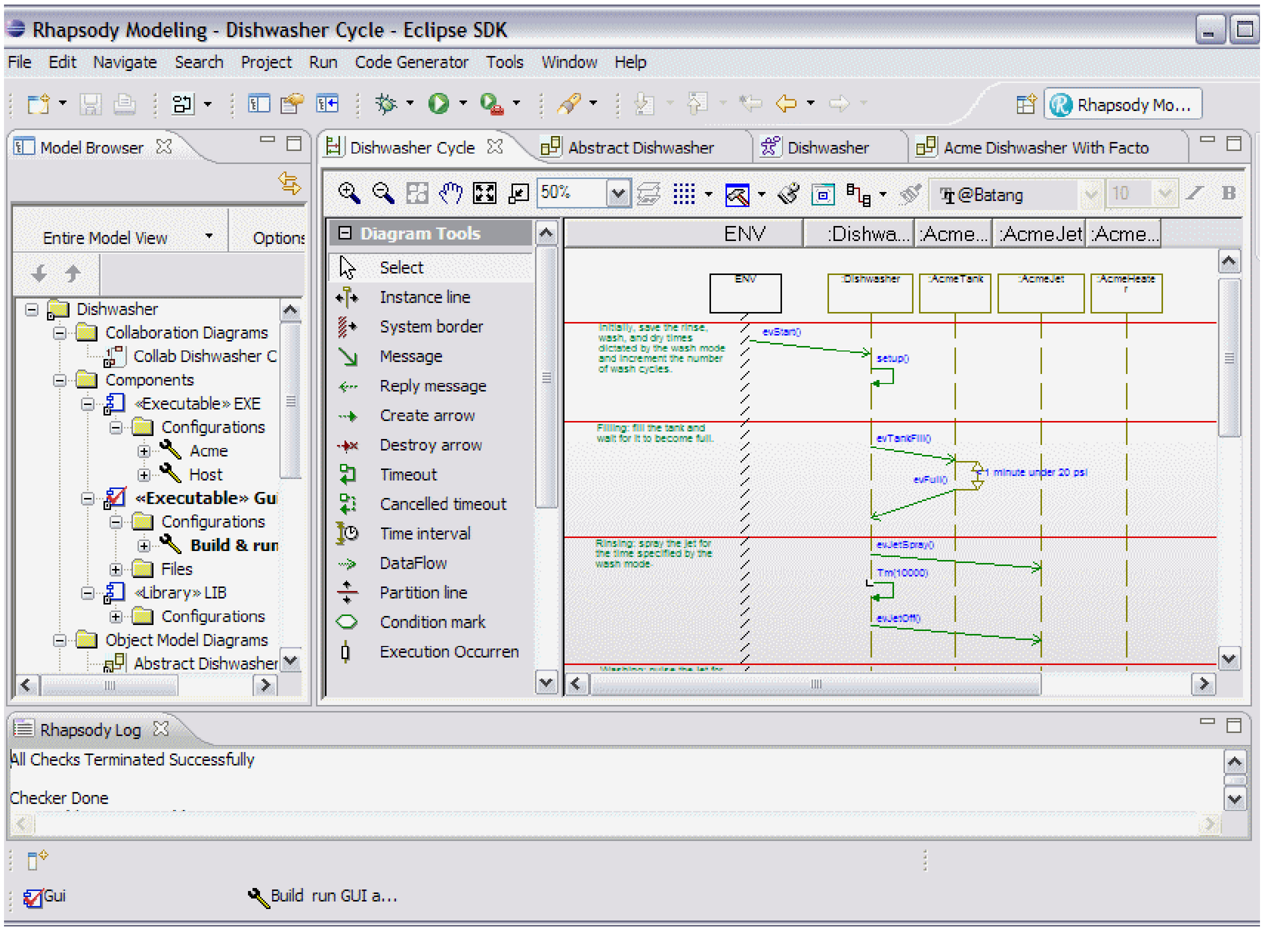
Guided tour of the Eclipse platform integration

The Rhapsody Platform Integration with Eclipse adds two Rational Rhapsody perspectives on tabs in the upper right corner of the Eclipse IDE:

- Rhapsody Modeling (shown in the example below)
- Rhapsody Debug

The Eclipse interface displays the following main Rhapsody interface components:

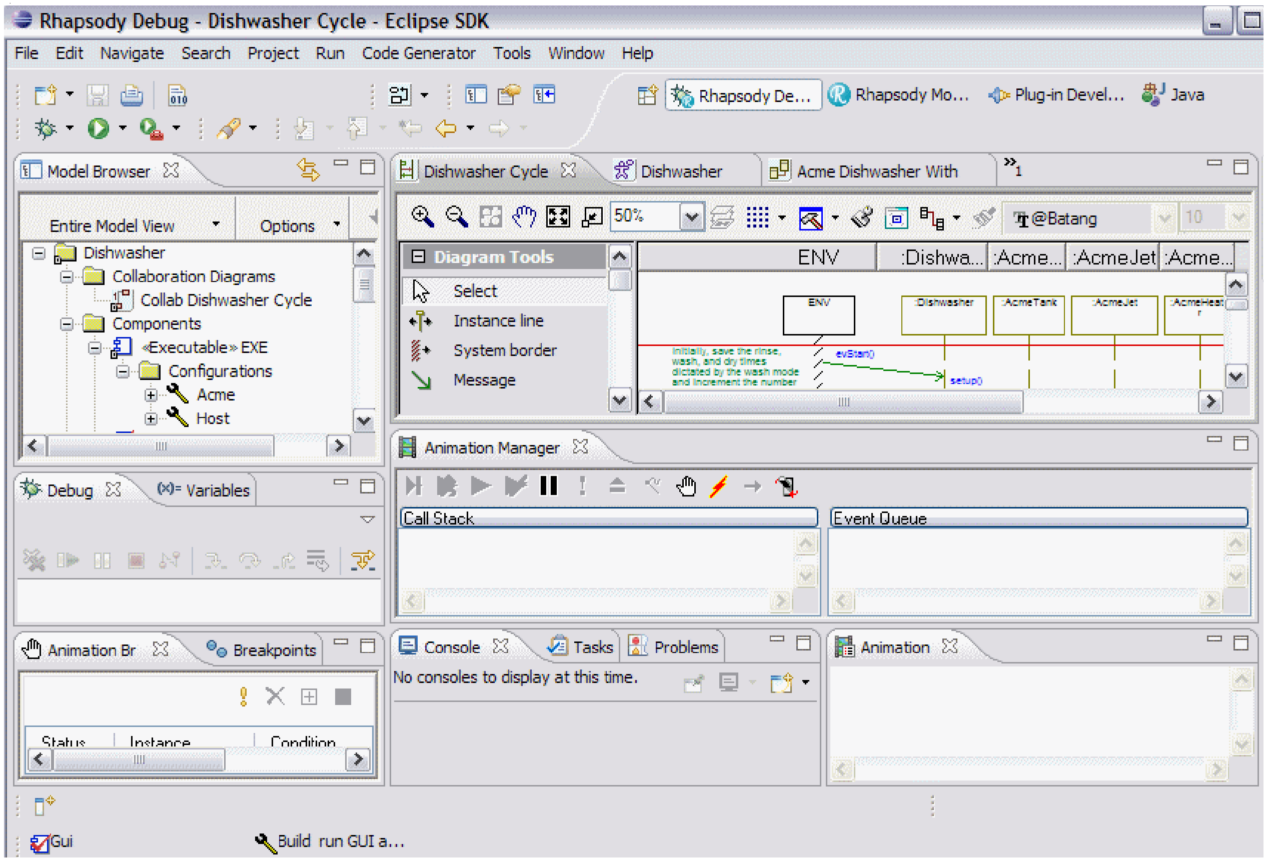
- Browser (Model Browser tab in Eclipse)
- Diagram Tools
- Drawing area
- Output window (Rhapsody Log tab in the example)



All Checks Terminated Successfully

Checker Done

The Rational Rhapsody menu options and drawing capabilities have been added to the Eclipse code editing, interface customization, and other capabilities. The Rhapsody Debug perspective displays the windows shown in this example



No consoles to display at this time.

Developers can then use the Eclipse code or design level debugger and animation with breakpoints for a thorough and efficient debugging strategy.

Creating Java plug-ins

You may create your own Rational Rhapsody plug-in for a Java application. Basically, there are three stages required to create a Java plug-in:

1. Code the plug-in.
2. Write a .hep file containing the plug-in definitions.
3. Attach it to the profile.

To speed your plug-in development process, you may use the `SimplePluginProfile.sbs` and its `SimplePluginProfile.hep` files located in the Samples/ExtensibilitySamples/Simple Plug-in directory along with instructions for using these samples. See the list of **Rational Rhapsody samples** for more information.

Rational Rhapsody for systems engineers

Rational Rhapsody allows systems engineers to capture and analyze *requirements* quickly and then design and validate system behaviors. In addition, the software produces high-quality systems engineering specification documents using report templates.

The Rational Rhapsody Systems Engineering *Add-on* installation provides the features to support the UML and SysML standards.

- Examine the UML Forum **UML specification**
- Examine the OMG **SysML specification**

Rational Rhapsody for systems engineers

Systems engineers use the Rational Rhapsody Harmony/SE process to manage:

- Requirements Analysis
- System Functional Analysis
- Design Synthesis
- Design validation through simulation
- Architectural Design

Note

For more information about the Harmony/SE process, see the *Harmony Deskbook* by Hans-Peter Hoffmann, Ph.D., 2009 (available on the corporate Web site).

Diagrams for systems engineering

A systems engineering Rational Rhapsody project includes both UML and SysML diagrams to define the model. You may use any of these diagrams for systems engineering:

- Use case diagrams
- Requirements diagrams
- Sequence diagrams

- Activity diagrams
- Statecharts
- Internal Block diagrams
- Block Definition diagrams
- Parametric diagrams

Special options and wizards

The Rational Rhapsody systems engineering features includes the following automated tools:

- Right-click menu options to perform common tasks quickly
- Wizards that perform repetitive tasks automatically or reduce the number of steps required to perform a systems engineering operation

DoDAF and MODAF development

Rational Rhapsody supplies the DoDAF, MODAF and UPDM Add On allowing engineers to create a compliant architecture model for their national standard. The Add On products, a template driven solution, can be customized and extended to meet specific customer requirements. To use the combined MODAF and DoDAF specification standard for your project, select the UPDM (**U**nified **P**rofile for **D**oDAF and **M**ODAF).

MODAF add on and viewpoints

To provide an effective Model Driven Development Solution for creating MODAF-compliant architectural models, use the Rhapsody MODAF Add On product together with Rhapsody in conjunction with a sound Systems Engineering Process and Methodology.

In addition to the MODAF variations of the standard DoDAF views, the Rational Rhapsody MODAF supports these viewpoints:

- **Strategic viewpoint** represents, in an abstract manner, what you want to do over time. It documents the strategic picture of how a capability (for example, a military capability) is evolving in order to support capability deployment and equipment planning. The Strategic, Operational, and Systems viewpoints have a layered relationship.
- **Acquisition viewpoint** is partly derived from elements of the Strategic viewpoint and provides information for the Operational and Systems viewpoints. The Acquisition viewpoint represents acquisition program dependencies, timelines, and the status of MOD Defence Lines of Development (DLOD, equivalent to U.S. Department of Defense DOTMLFPs) status so that the various MOD programs are managed and synchronized correctly. Note that the AcV-1 and AcV-2 views are not supported in the Rhapsody MODAF profile.

DoDAF add on and viewpoints

The Rhapsody DoDAF Add On includes the DoDAF profile, a number of DoDAF helper utilities, a DoDAF Reporter Template, a Microsoft Word Document Template file, a Rhapsody ReporterPLUS License, and an image library with a set of public domain graphics for military applications.

Use the DoDAF profile to define the *architecture* model in these viewpoints:

- **Operational viewpoint**. In both DoDAF and MODAF, this view documents the operational processes, relationships, and context to support operational analyses and requirements development (meaning it identifies what needs to be accomplished and who does it).
- **Systems viewpoint**. In both DoDAF and MODAF, this view documents system functionality and interconnectivity to support system analysis and through-life management. (In other words, it relates systems and characteristics to operational needs).
- **Technical viewpoint**. In both DoDAF and MODAF, this view documents policy, standards, guidance, and constraints to specify and assure quality expectations.
- **All Views viewpoint**. In both DoDAF and MODAF, this view provides summary information for the architecture that enables it to be indexed, searched, and queried. All Views encompasses all of the other views as there are overarching aspects of architecture that relate to the Strategic, Acquisition, Operation, Systems, and Technical views.

The DoDAF architecture model includes the following architecture products:

Architecture Product	Viewpoint or View	Product Name	Product Description
All Views	Package	AllViews	This optional stereotyped package allows you to add in AV products and other views and packages, if desired.
AV-1	All	Overview and Summary Information	This product is typically a text (Word, FrameMaker, HTML) document. You can add AV-1documents and launch them by clicking on them.
AV-2	All	Integrated Dictionary	This is also a text product.
Operational View	Package		This optional stereotyped package is similar to the All View product. It supports all the operational products.
OV-1	Operational	High-Level Operational Concept Graphic	This High-level graphical/ textual description of the operational concept allows you to import pictures and other operational elements, such as Operational Nodes, Human Operational Nodes, Operational Activities and the relations among them.
OV-2	Operational	Operational Node Connectivity Description	This product shows the connections and flows among operational nodes. This is not an activity diagram. If desired, activity diagrams (OV-5s) or state machines (OV-6b) can be used to detail how the operational nodes and activities behave. These diagrams are the primary source of info for the OV-3.
OV-3	Operational	Operational Information Exchange Matrix	This product shows information exchanged between nodes and the relevant attributes of that exchange. OV-3 is generated from the information shown in OV-2 and other operational diagrams. This information is stored as a CSV file and can be added to any product.
OV-5	Operational	Operational Activity Model	This product details the behavior of operational nodes or more commonly, operational activities.
OV-6a	Operational	Operational Rules Model	This product is a textual description of “business rules” for the operation. It is a controlled file. One of three products used to describe the mission objective.
OV-6b	Operational	Operational State Transition Description	This product is a statechart that can be used to depict the behavior of an operational element (node or activity). One of three products used to describe the mission objective.

Architecture Product	Viewpoint or View	Product Name	Product Description
OV-6c	Operational	Operational Event Trace Description	This product is a sequence diagram that captures the behavioral interactions among and between operational elements and (in the Harmony process) captures the operational contracts among them. One of the three products used to describe the mission objective.
OV-7	Operational	Logical Data Model	This product is a class diagram that shows the relations among Informational Elements (data classes). This is similar to entity relationship diagrams, but is more powerful. This is not an activity diagram
System View	Package		This optional stereotyped package is similar to other views, but contains system elements.
SV-1	Systems	Systems Interface Description	This product is a structure diagram that contains System nodes, systems, and system parts and the connections between them (links). These can be used with or without ports.
SV-2	Systems	Systems Communications Description	This product is a structure diagram that shows the connections among systems via the communications systems and networks.
SV-3	Systems	Systems-Systems Matrix	This product is generated from the information in the other system views. SV-3 assumes that there are links between items stereotyped SystemNode, System, or System Part and represents these in an N2 diagram
SV-4	Systems	Systems Functionality Description	This product represents the connection between System Functions and Operational Activities. This is done by drawing a Realize dependency from the System Function to the Operational activity on the diagram. System Functions are mapped onto the system elements that support them by making the System Function parts (drawn within), the system elements. Note that System elements can also realize system functions. Note that here, as in almost all the other views, you can use Performance Parameters (bound to their constrained elements via anchors) to add performance data. This is summarized in SV-7. This is not an activity diagram.
SV-5	Systems	Operational Activity to Systems Function Traceability Matrix	This product is a spreadsheet-like generated view summarizing the relations among system elements (system nodes, systems and system parts), system functions that they support, and the mapping to operational activities.

Architecture Product	Viewpoint or View	Product Name	Product Description
SV-6	Systems	Systems Data Exchange Matrix	This product shows the information in the flows (information exchanges) between system elements. They may be embedded flows (bound to the links) or they may be flows independent of links. This is a spreadsheet-like generated product.
SV-7	Systems	Systems Performance Parameters Matrix	This is a generated spreadsheet-like product, showing all the performance parameters and the elements that they constrain.
SV-8	Systems	Systems Evolution Description	This product is the system evolution description. This is an activity diagram (there is a SystemProject element stereotype to serve as the "base" for this activity diagram). SV-8 depicts the workflow for system development, object nodes for products released, and performance parameters for things like start and end dates, slack time, etc.
SV-9	Systems	Systems Technology Forecast	This product is a text document - a stereotype of a Controlled File.
SV-10a, SV-10b, SV-10c	Systems	Systems Rules Model, Systems State Transition Description, Systems Event Trace Description	These products are similar to the OV-6a, OV-6b, and OV-6c products, but they are separately identified, even though they are structurally identical.
SV-11	Systems	Physical Schema	This product is similar to the OV-7 class diagram. This product uses a class diagram to show physical schema (data representation).

Index