

Telelogic **Rhapsody**

Getting Started



IBM®

Rhapsody®

Getting Started 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 Telelogic Rhapsody 7.4 and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1997, 2008.

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

Contents

Rhapsody Basics	1
Rhapsody's Scope	1
Starting Rhapsody	2
Creating a New Project	3
Rhapsody Guided Tour	4
Locating Rhapsody Icons	5
Output Window	6
Managing Windows	7
Rhapsody Diagrams	8
Standard UML Diagrams	8
Systems Engineering Diagrams	8
Rhapsody Samples	9
Search and Replace in Models	19
Searching Models	19
Working with Search Results	22
Replacing	23
Using the Rhapsody Browser	24
Repositioning the Browser	24
Filtering the Browser Display	25
Re-ordering the Browser Elements	25
Changing a Browser Item's Name	26
Adding Browser Items	28
Moving and Copying Browser Items	28
Creating a List of Favorites	29
Deleting Browser Items	29
Creating Backups	30
Naming Conventions and Guidelines	31
Standard Prefixes	31
Guidelines for Naming Model Elements	31
Saving the Project	32
Closing the Project and Exiting Rhapsody	32

Parallel Development	33
DiffMerge Tool Features	33
Starting the DiffMerge Tool	34
Rhapsody and Eclipse Basics	35
Standard Rhapsody and <i>the Platform Integration</i>	35
Creating a <i>Rhapsody</i> Project in Eclipse	36
Guided Tour of <i>the Eclipse Platform Integration</i>	39
Creating Java Plug-ins	41
Rhapsody for Software Development	43
Quick Start with C	44
Opening the Elevator Model	44
Saving the Sample C Project	45
Examining the Use Case Diagram	45
Defining Messages Using Sequence Diagrams	47
Animating the Elevator Model	49
Stopping the Program	58
Quick Start with C++	59
Opening the Radio Model	59
Saving the Sample C++ Project	59
Viewing Diagrams	60
Animating the Radio Model	65
Stopping the Program	72
Quick Start with Ada	73
Opening the RS232 Sample Project	73
Saving the Sample Ada Project	73
Creating a Class	74
Creating an Object Model Diagram	75
Checking the Configuration	78
Generating Code	78
Building the Application	79
Running the Application	82
Examining the Code	83
Closing the Project	84
Quick Start with Java	85
Opening the HomeAlarm Sample Project	85
Saving the Sample Java Project	85
Viewing Diagrams	86
Creating an Animated Sequence Diagram	90
Animating the HomeAlarm Model	91

Comparing Animated Sequence Diagrams	93
Animating a Statechart	94
Rhapsody for Systems Engineers	97
Rhapsody for Systems Engineers	97
Diagrams for Systems Engineering	98
Special Options and Wizards	98
More Systems Engineering Information	98
Rhapsody for DoDAF and MODAF Development	99
MODAF Viewpoints	99
DoDAF Viewpoints	100
Reports	105
ReporterPLUS	105
Starting ReporterPLUS	106
Examining and Customizing ReporterPLUS Templates	107
Generating Reports	109
Standard Templates	110
Using the Internal Reporting Facility	112
Producing an Internal Report	112
Using the Internal Report Output	113
Technical Support and Documentation	115
Contacting Telelogic Rhapsody Support	115
Accessing the Automated Problem Report Form	116
Automatically Generated Problem Reports	117
Calling Telelogic Rhapsody Technical Support	117
Contacting IBM Rational Software Support	118
Accessing the Rhapsody Documentation	119
Help Menu Options	119
Rhapsody Reference Documentation	121
Rhapsody Version Release Documents	122
Additional Reference Materials	123
More Training and Learning Resources	124
Rhapsody Training Classes	124
Rhapsody eLearning Courses	124
Index	125

Rhapsody Basics

Welcome to Telelogic Rhapsody®!

To give you a quick start using Rhapsody, you may use the following:

- ◆ On Demand Webinars provide self-training and research opportunities.
- ◆ You can attend hands-on training one of our corporate training facilities or in your office.
- ◆ eLearning courses provide self-paced, online Rhapsody courses (available at an extra cost).
- ◆ Tutorials for different jobs and types of development environments step you through building a functional model.
- ◆ This *Getting Started Guide* provides a Rhapsody functional overview for designers, developers, and engineers creating specialized models.

Rhapsody's Scope

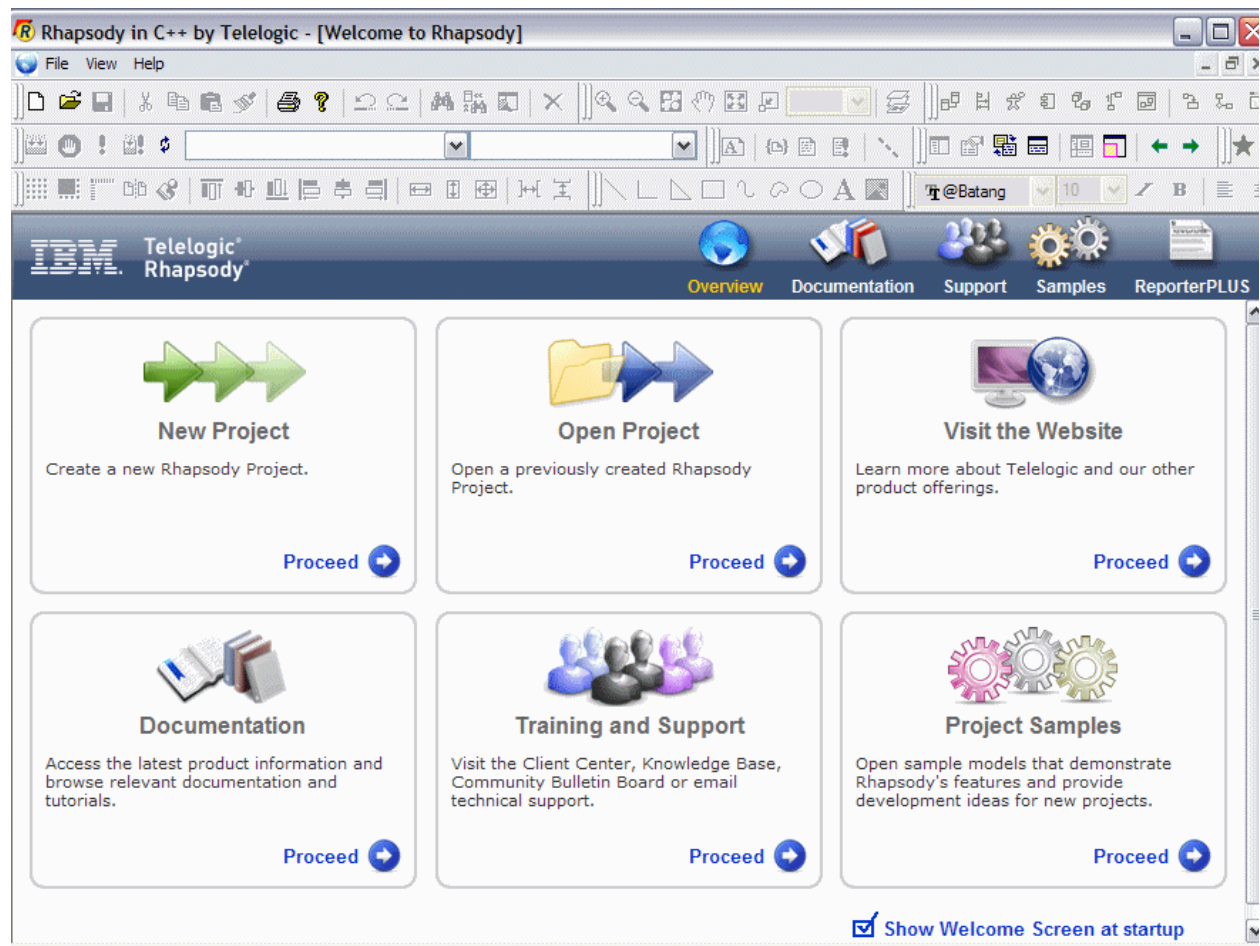
Software developers, systems engineers and systems architects use Rhapsody to create either embedded or real-time systems. Rhapsody provides a visual design environment to create requirements and model systems using the Unified Modeling Language™ (UML™) diagrams and SysML™ diagrams. Rhapsody allows you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

- ◆ **Analysis**—Define system requirements, identify necessary objects, and define their structure and behavior.
- ◆ **Design**—Trace requirements to the design, taking into account architectural, mechanistic, and detailed design considerations.
- ◆ **Implementation**—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.

Starting Rhapsody


To start Rhapsody for developers, click the **Rhapsody** icon or from the *Windows* Start menu select **Telelogic > Telelogic Rhapsody version # > Rhapsody Development Edition > Rhapsody in <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.

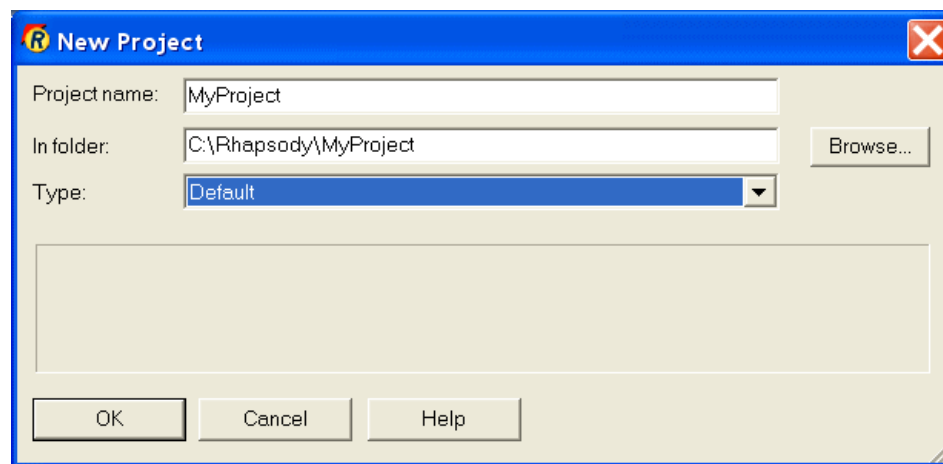
- ◆ Create a **New Project** or **Open a Project** created previously
- ◆ Visit our corporate Web site
- ◆ Access product **Documentation**, view the list of available technical **Training** courses, or contact technical **Support**
- ◆ View the official Rhapsody **Project Samples** delivered with the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



Creating a New Project

To create a new project, follow these steps:

1. With 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.



3. The **Default Type** provides all of the basic UML structures and is useful for most Rhapsody projects. If you want to create a specialized project, you may select one of the other profiles or create a profile of your own. For more information about the profiles, refer to the *Rhapsody User Guide*.
4. Click **OK**. If the directory does not currently exist, Rhapsody asks whether you want to create it. Click **Yes** to create a new directory for the project.

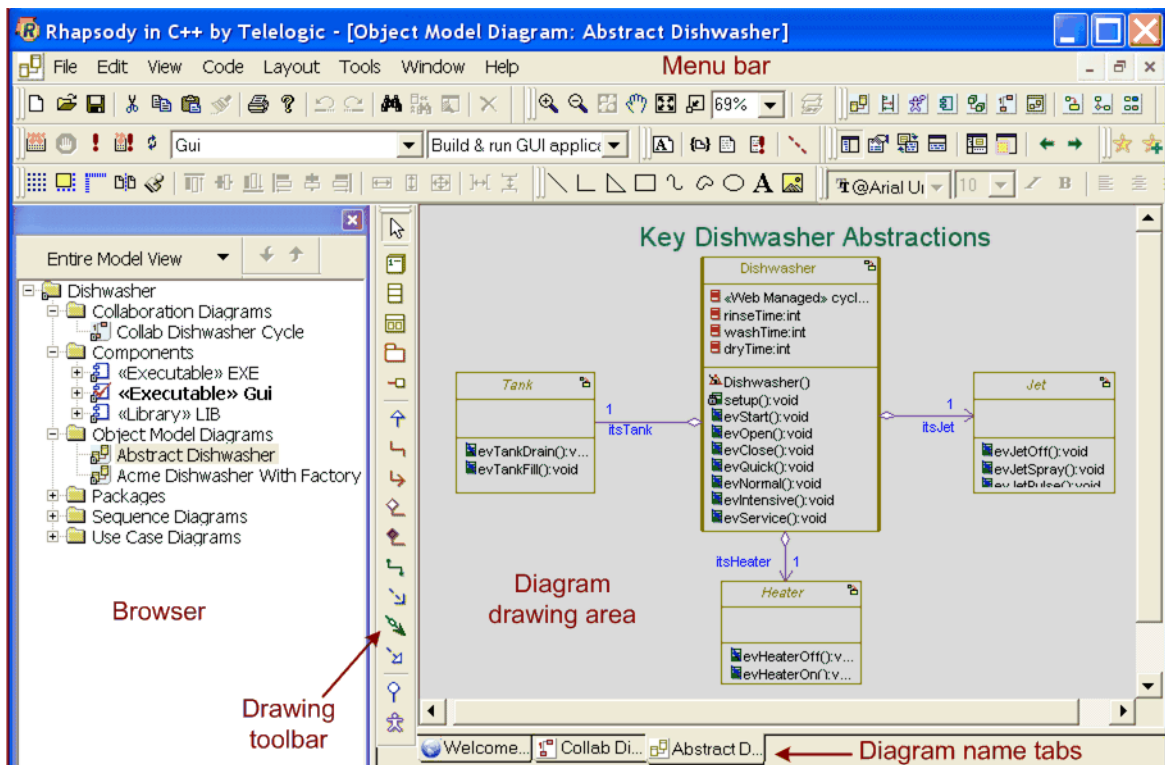
Rhapsody creates a new project in the `MyProject` subdirectory and opens the new project. The project name in the directory is `MyProject.rpy`.

See the [Rhapsody Guided Tour](#) for descriptions of the areas in the Rhapsody interface.

Rhapsody Guided Tour

The 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, 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. For information about Rhapsody displayed in the Eclipse interface, see the [Guided Tour of the Eclipse Platform Integration](#).

The **Drawing** toolbar in the center of the window contains the icons appropriate for the type of diagram displayed in the drawing area. Therefore, the icons change when you display a different diagram. The following figure is a labeled tour of the Rhapsody interface using a C++ project.



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.

Locating Rhapsody Icons


The icons are grouped in two areas:

- ◆ Under the menu bar across the top of the Rhapsody window
- ◆ In the center of the window between the browser and the drawing area.

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 Rhapsody window by dragging and dropping the double-bar, left edge of a group of icons.

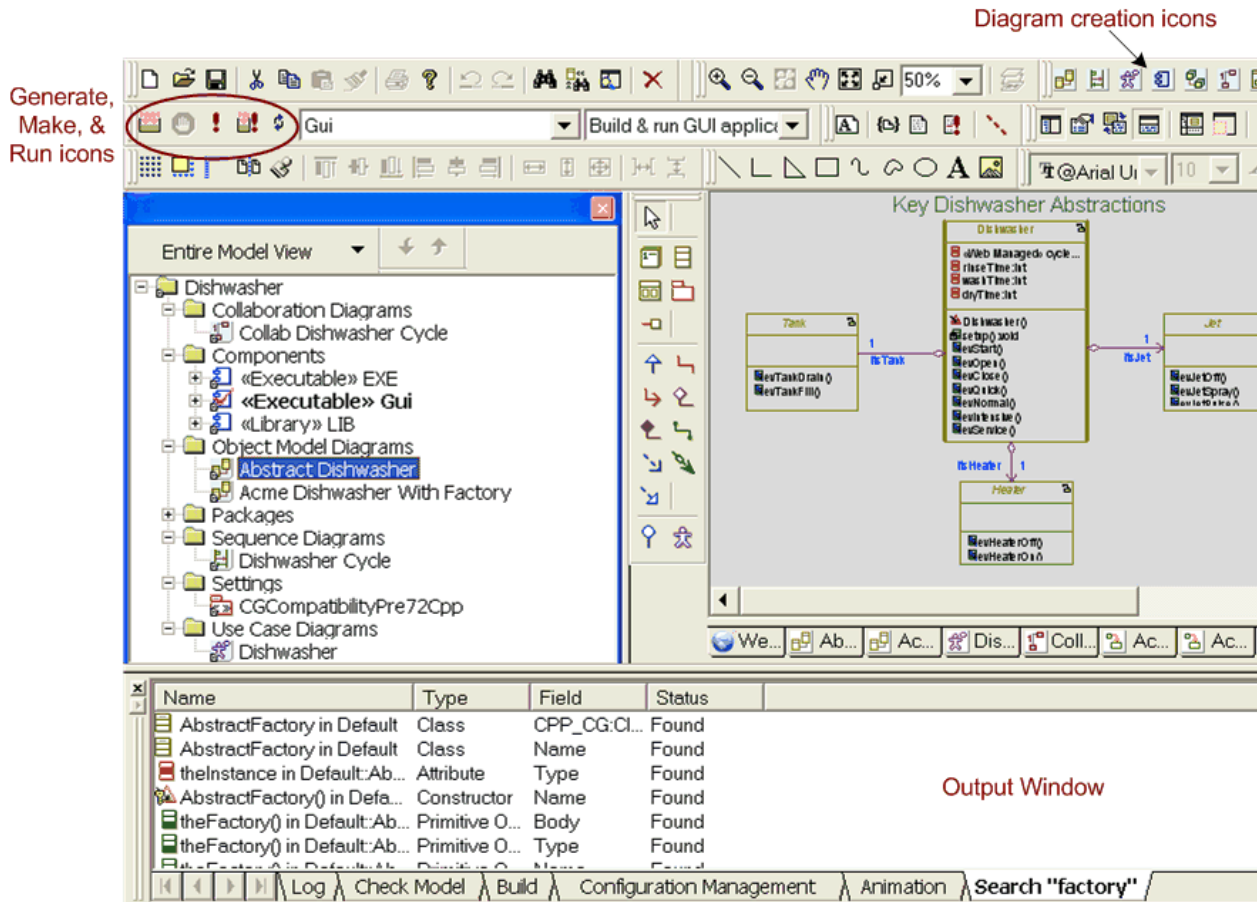
The icons on the interface are perform functions that are also available from menus. For example, you may select the **Tools > Sequence Diagram** menu option or click the diagram creation icon for the Sequence Diagram  to create a new sequence diagram. The diagram icons are shown in the following illustration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

Note

Remember that you may position your cursor over any icon to see the name of the icon displayed.

Output Window

When testing your model with the Generate, Make, and Run icons (circled in the following figure) or those options on the **Code** menu, Rhapsody displays Log output at the bottom in the **Output window**. The Output window tabs display different types of information, such as the Search results shown in the lower portion of the following figure.



Note

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

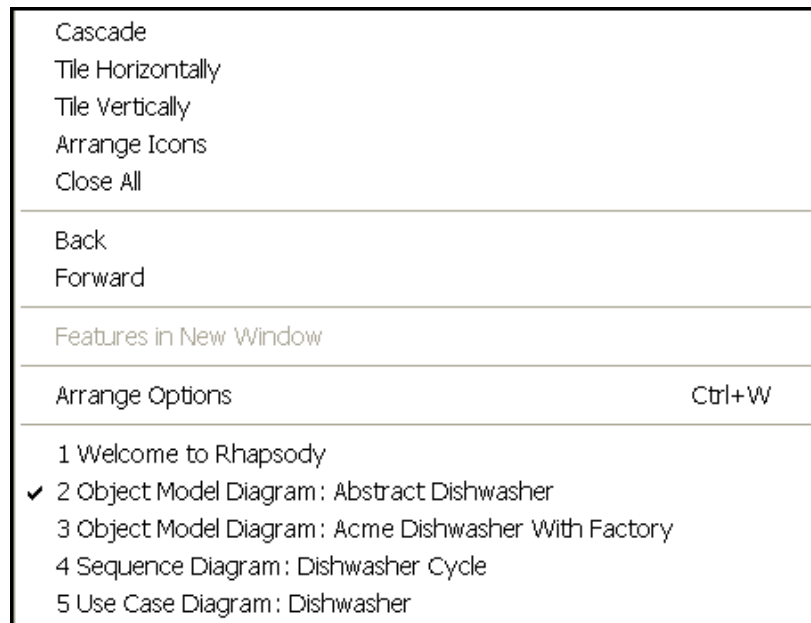
Managing Windows

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.

At the bottom of the menu is a list of all *open windows*. Click a window name in this list to make it the currently active window.



Rhapsody Diagrams

Rhapsody diagrams types are UML and SysML standard diagrams, as listed below. Refer to the *Rhapsody User Guide* for more detailed information about each diagram.

Standard UML Diagrams

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

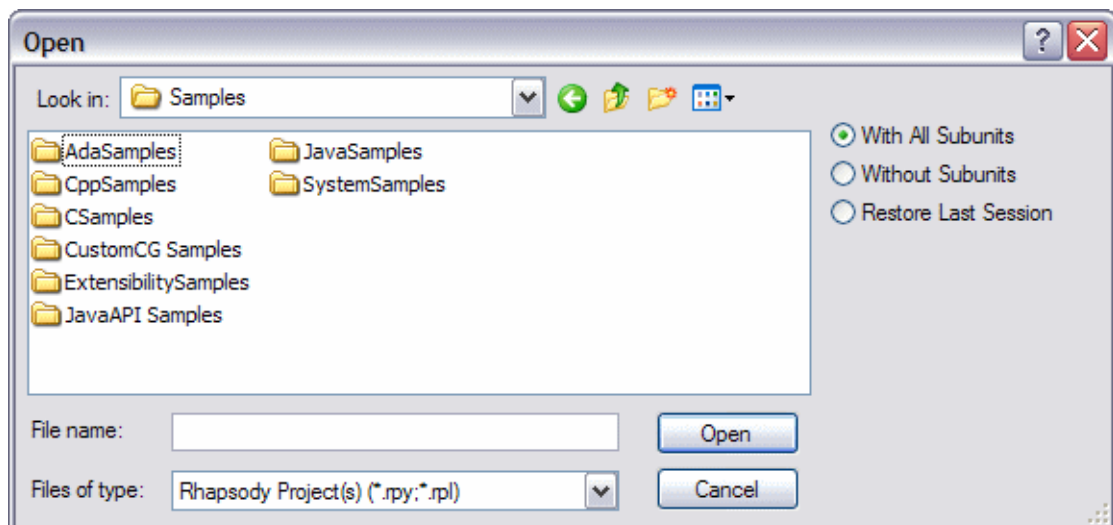
These two profiles also provide additional specialized diagrams:

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

Rhapsody Samples

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

1. Select **File > Open**.
2. Navigate to your Rhapsody installation and select the Samples directory to display the samples installation with your software. This example shows all of the folders that may be installed.



If there are no sample directories, reinstall the Rhapsody software 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, refer to the instructions in the `Samples\CPP\Readme.txt` file to recompile that code in your environment.

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

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 Rhapsody samples with any altered versions.

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. Refer to the <i>API Reference Manual</i> for more information.
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.
Command Line Interface	C++	This directory contains a sample script to run 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 Rhapsody to create CORBA features including a CORBA interface.

Sample Name	Environment	Description
DesignPatterns	C++	<p>The project contains samples of common model patterns that can be copied into your projects and modified as needed:</p> <ul style="list-style-type: none"> • WatchDogPattern • TimeSlicingPattern • StaticPriorityPattern • StaticAllocationPattern • SanityCheckPattern • SafetyExecutivePattern • ProxyPattern • PriorityCeilingPattern • PreemptiveMultitaskingPattern • PollingPattern • MonitorActuatorPattern • MicroKernelPattern • MasterSlavePattern • HomogeneousRedundancyPattern • HeterogeneousRedundancyPattern • HandshakePattern • FixedSizeBlockAllocationPattern • ExecutionControlPattern • DynamicPriorityPattern • CyclicExecutivePattern • BrokerPattern
DiffMerge	C++	<p>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. Refer to the <i>Team Collaboration Guide</i> for more detailed information about this tool.</p>
Dishwasher	C++	<p>This project defines dishwasher parts and a GUI and contains a sequence diagram, collaboration diagram, and abstract and implementation versions of the dishwasher design.</p>
Elevator	C++	<p>This project contains executables for the GUI, host, and target. It also includes one main use case and several sequence and collaboration diagrams.</p>
Handset	C++	<p>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.</p>
hhs	C++	<p>The Home Heating System (hhs) project contains a GUI, object model diagrams, sequence diagrams, and collaboration diagrams.</p>

Sample Name	Environment	Description
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 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.
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 system's functional requirements.
vba	C++	This model demonstrates the uses of VBA with 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.

Sample Name	Environment	Description
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: <ul style="list-style-type: none"> • 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. Refer to the <i>Team Collaboration Guide</i> for more detailed information about this 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.
FlowChart	C	The project contains the following basic flowchart patterns that can be used and modified for your projects: <ul style="list-style-type: none"> • 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.

Sample Name	Environment	Description
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 SimulinkC (automatically added profile) and requires a Simulink license.
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. Refer to the <i>Team Collaboration Guide</i> for more detailed information about this 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.
CPP	Extensibility Samples for Callback API	This simple application implements the EventListenerPlugin interface and registers with Rhapsody in order to receive callbacks. For more information, refer to the <code>readme.txt</code> file in the Samples/ExtensibilitySamples/CallbackAPI Samples/ CPP/EventListenerPlugin directory and the <i>API Reference Manual</i> .
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.

Sample Name	Environment	Description
SPARK	Ada	<p>This model contains two projects to illustrate the SPARK standard's support in Rhapsody's Ada. The model includes the following special features:</p> <ul style="list-style-type: none"> • Stack state machine • Stack class as an abstract data type • Monitoring class showing how a data type can be extended.
Java	Extensibility Samples for Callback API	<p>In the Samples/ExtensibilityChecks/CallbackAPI Samples/Java directory are two subdirectories with three applications in each:</p> <ul style="list-style-type: none"> • <code>Plug-in</code> contains the <code>ApplicationListenerPlugin</code>, <code>CodeGenerationListenerPlugin</code>, and the <code>RoundtripListenerPlugin</code>. • <code>Stand-alone</code> contains applications that implement Listener interfaces in order to receive callbacks. <p>For more information, refer to the <code>readme.txt</code> files in the subdirectories containing the application files and the <i>API Reference Manual</i></p>
VB	Extensibility Samples for Callback API	<p>This simple application implements the <code>EventListenerTest</code> interface and registers with Rhapsody in order to receive callbacks. The <code>EventListenerTest</code> API uses Rhapsody to listen for specified events (for example, <code>beforeProjectClose</code>, <code>afterProjectClose</code>, <code>onDiagramOpen</code>, <code>onFeaturesDialogOpen</code>, <code>onCodeGenerationCompleted</code>, and <code>beforeRoundtrip</code>). Use 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.</p> <p>For more information, refer to the <code>readme.txt</code> file in the Samples/ExtensibilityChecks/CallbackAPI Samples/VB/EventListeners directory and the <i>API Reference Manual</i>.</p>

Sample Name	Environment	Description
VBA	Extensibility Samples for Callback API	This simple application implements the <code>EventListenerTest</code> interface and registers with Rhapsody in order to receive callbacks. As in VB Callback API, the VBA Application Listener Client listens to Rhapsody for specified events (for example, <code>beforeProjectClose</code> , <code>onDiagramOpen</code> , <code>onFeaturesDialogOpen</code> , <code>onCodeGenerationCompleted</code> , and <code>beforeRoundtrip</code>). However, in the VBA sample, the <code>afterProjectClose</code> event is not used since a VBA project is part of a Rhapsody project. If Rhapsody is closed, the VBA project would also be closed. For more information, refer to the <code>readme.txt</code> file in the <code>Samples/ExtensibilityChecks/CallbackAPI/Samples/VBA/VBA_EventListeners_SampleRhpProject</code> directory and the <i>API Reference Manual</i> .
VB	Extensibility Samples for ExternalChecks Samples	This project runs <code>rpyexternalchecks.exe</code> to check the model's 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 <code>.hep</code> file with a path to the check in the rhapsody project directory, as well as a property pointing to the <code>.hep</code> file.
.settings	Extensibility Samples for Simple Plug-in	This directory contains an Eclipse JDT preferences file. For more information about creating plug-ins for Rhapsody, refer to the <code>How to create a plug-in.rtf</code> document in the Simple Plug-ins directory.
com	Extensibility Samples for Simple Plug-in	This directory contains two sample plug-ins for Rhapsody: <ul style="list-style-type: none"> • <code>SimplePlugin.class</code> • <code>SimplePlugin.java</code>
APIExtension	Java API	This project demonstrates how to extend Rhapsody's Java-API in a stand-alone application. For more information, refer to the <code>readme.txt</code> file in the <code>Samples/JavaAPI/Samples/APIExtension</code> directory and the <i>API Reference Manual</i> . To extend the API in a plug-in application, refer to samples in the <code>Samples/JavaAPI/Samples/Plug-in</code> directory.
ClassDumper	Java API	This sample contains a batch file to dump a Rhapsody class. For more information, refer to the <code>readme.txt</code> file in the <code>Samples/JavaAPI/Samples/ClassDumper</code> directory and the <i>API Reference Manual</i> .

Sample Name	Environment	Description
com.telelogic.rhapsody.wfi.rhapsodyListenersExample	Java API	This plug-in demonstrates how to use the Rhapsody Listener extension point to receive notifications on messages and commands from Rhapsody. This plug-in implements “rhapsodyListeners” extension points. For more information, refer to the <code>readme.txt</code> file in the Samples/JavaAPI Samples/com.telelogic.rhapsody.wfi.rhapsodyListenersExample directory and the <i>API Reference Manual</i> .
JavaPlug-in	Java API	This diagram formatter sample demonstrates how to create a Java plug-in and how to extend Rhapsody's 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, refer to the <code>readme.rtf</code> file in the Samples/JavaAPI Samples/Plug-in directory.
Adms	Systems	This sample describes the ADMS (Aircraft Defense Management Model) in a Rhapsody project. Refer to the model overview and requirements documents within the project for a complete description of this sample.
Distiller	Systems	The Distiller model uses Rhapsody's 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. Refer to http://www.omg.sysml.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, refer to the instructions in the <code>Net Centric Mini Tutorialv1.1.doc</code> in this directory.
SysMLHandset	Systems	This is a SysML version of the Rhapsody Handset model.
VB_Post_Simplifier	CustomizeCG	The TestModel subdirectory contains a Rhapsody project for post simplification.

Sample Name	Environment	Description
Statechart_Simplifier_Writer	CustomizeCG	<p>This directory contains four sample directories:</p> <ul style="list-style-type: none"> • Statechart_ALT_Simplifier - This 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 <code>C.CG::Statechart::Simplify</code> property to the "ByUser" value. • Statechart_Java_Simplifier - This 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 <code>C.CG::Statechart::Simplify</code> property to the "ByUser" value. • Statechart_VB_Simplifier - This 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 <code>C.CG::Statechart::Simplify</code> property to the "ByUser" value. • Statechart_Writer_Rules - This package <code>Statechart_Generation</code> provides all of the rules related to the statechart generation. You must make some modifications to enable this process. <p>For more information, refer to the individual <code>readme.txt</code> files in the four directories and the <i>API Reference Manual</i>.</p>

Search and Replace in Models


Engineers and developers can use Rhapsody's 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.

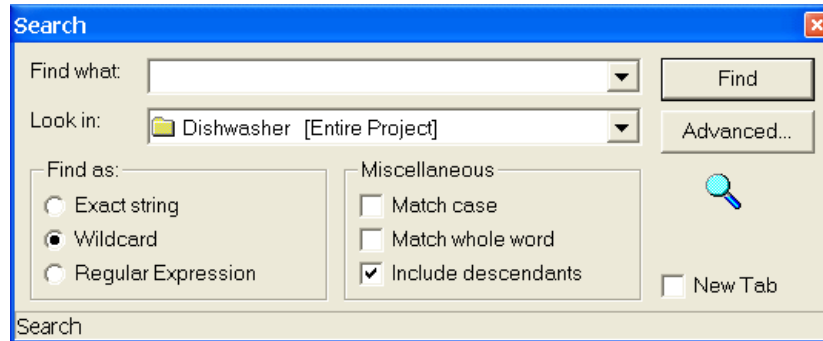
Searching Models

To search models, follow these steps:

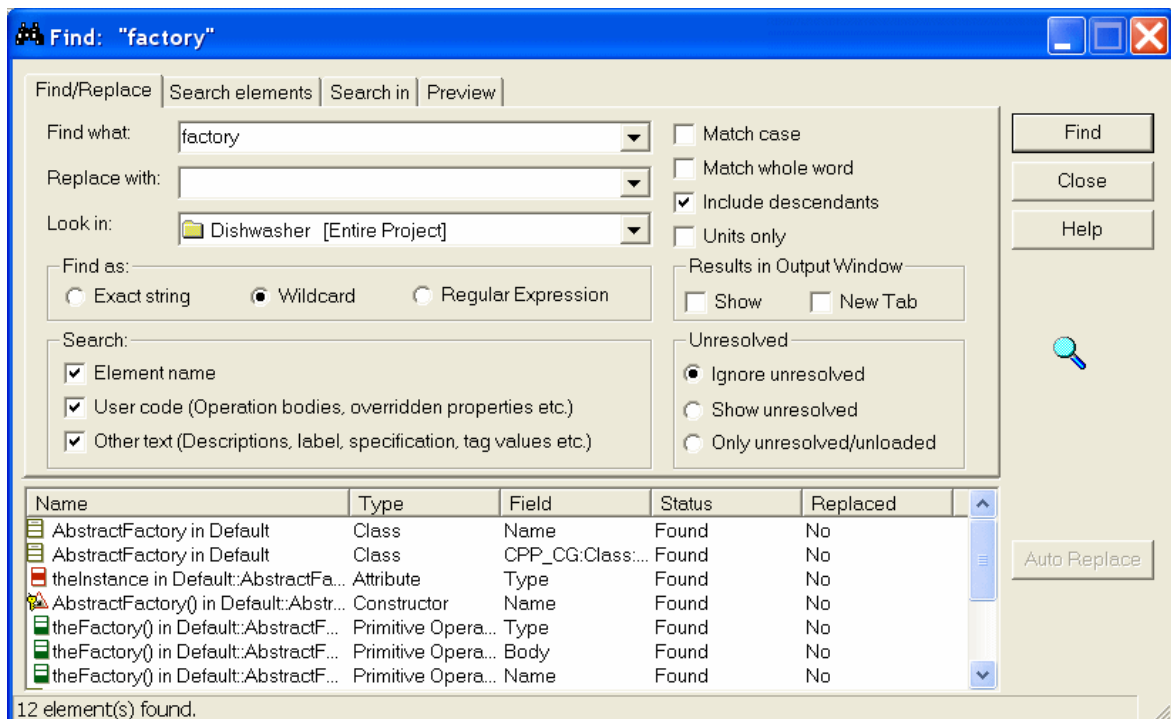
1. With the model displayed in Rhapsody, there are three methods to launch the Search facility: select the **Edit** menu (as shown in the following figure) and then select the **Search** option, click the binoculars icon , or press **Ctrl + F**.

Undo	Ctrl+Z
Redo	Ctrl+Y
Delete	Del
Format	
Add New	▶
Features ...	Alt+Enter
Search...	Ctrl+F
Advanced Search Replace...	Ctrl+H
Search inside Selected...	Ctrl+Alt+F
References...	Ctrl+R
Locate in Browser	Ctrl+L
Locate in IDE	Ctrl+Alt+K

2. This displays the Search dialog box (as shown in the following figure) 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's 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 (as shown in the previous figure). Both methods display the Advanced Search dialog box (as shown in the following figure). This dialog box provides the Unresolved and Units only search features.



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

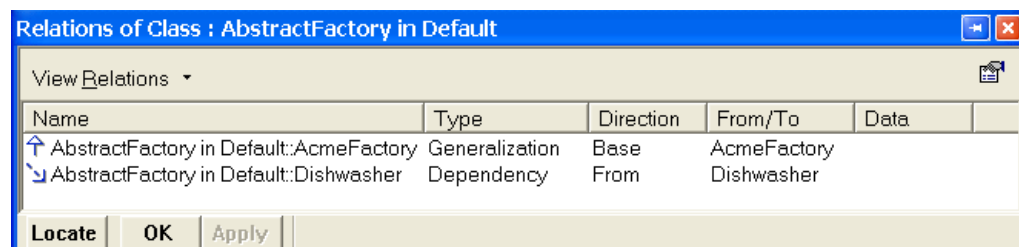
After locating elements using the Search facility, you may perform these operations in the Search dialog box or in the Output window:

- ◆ 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, follow these steps:

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.



To delete an item located in search process, follow these steps:

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

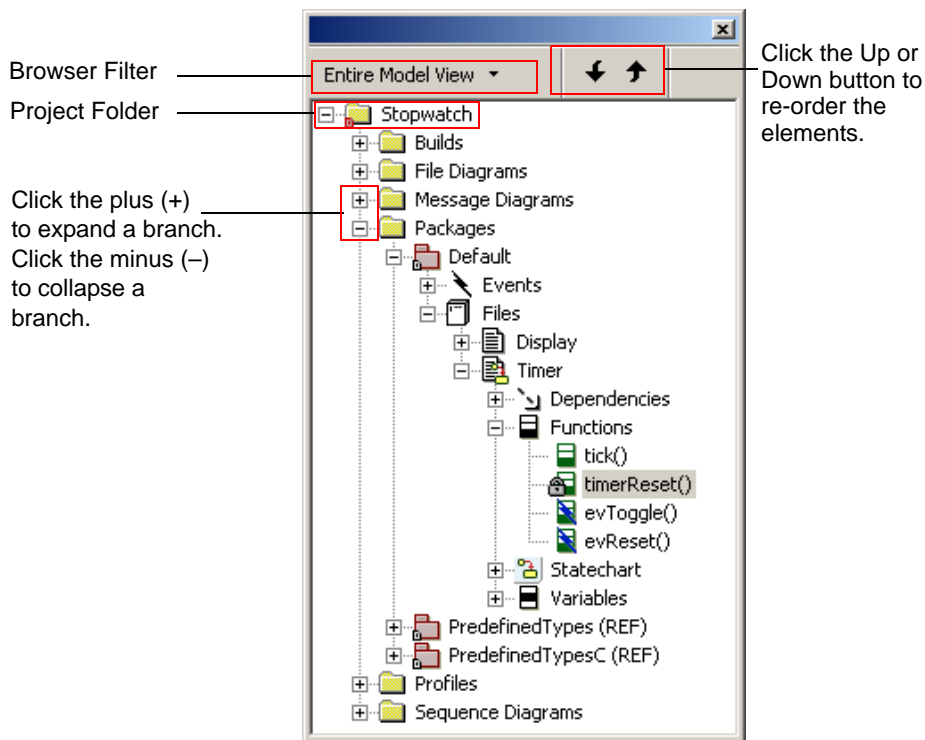
If you want to replace item names or other terminology throughout the model, follow these steps:

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.

Using the Rhapsody Browser

The Rhapsody browser shows the contents of the project in an expandable tree structure. By default, it is the upper, left section of the 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. Rhapsody automatically creates a default package called **Default**, which it uses to save model parts unless you specify a different package. The following figure shows an example of the browser.



Repositioning the Browser

To make more room to work on diagrams, you can move the browser outside of the Rhapsody GUI to reposition it as a separate window on the desktop. To reposition the 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:

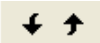
- ◆ **Entire Model View** displays all model elements in the browser and is the default view.
- ◆ **Use Case View** displays use cases, actors, sequence diagrams, use case diagrams, and relations among use cases and actors.
- ◆ **Component View** displays components, nodes, and packages that contain 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.

Refer to the *Rhapsody User Guide* for information.

Re-ordering the Browser Elements

You can re-order the elements in the 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 .

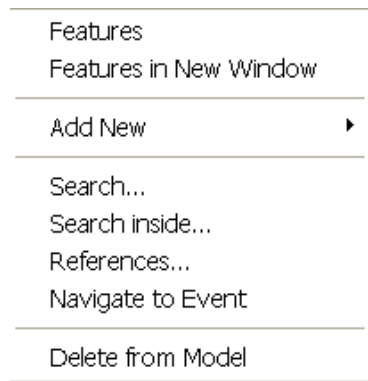
Changing a Browser Item's Name

You may change an item's name in the browser, but you must always keep the [Guidelines for Naming Model Elements](#) in mind when making these changes.

The quick method to change a browser item's name is to click on the name and type the new name over the existing name. This technique is most often used to when you have added a new item and you want to replace the system generated name with a meaningful name.

To change the name of an item listed in the browser that does not permit the quick method or when you want to make other changes using the Features dialog box, follow these steps:

1. Highlight the item in the browser. This example uses an event in a C++ model.
2. Right-click to display this menu and select the Features option or double-click to displays the Features dialog box immediately.



3. If long names are being displayed, you may resize this dialog box to make the names display fully.



4. To change the event's name, click **L** (Label) beside the **Name** field to display the **Name and Label** dialog box, as shown in the following figure.



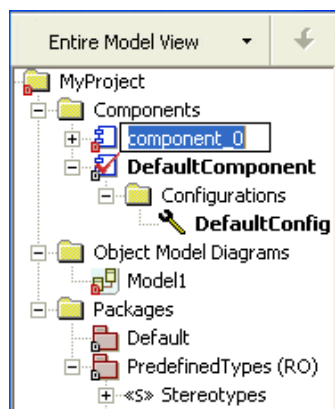
5. Type the new label name for the browser item, and click **OK** to save the change and close the dialog.
6. You may make other changes in the Features dialog box, such as adding a Description of the item using the area displayed when you click the **Description** tab.
7. Click **OK** to close the Features dialog box and save all of your changes.

Adding Browser Items

To add new items to a category in the browser, follow these steps:

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.

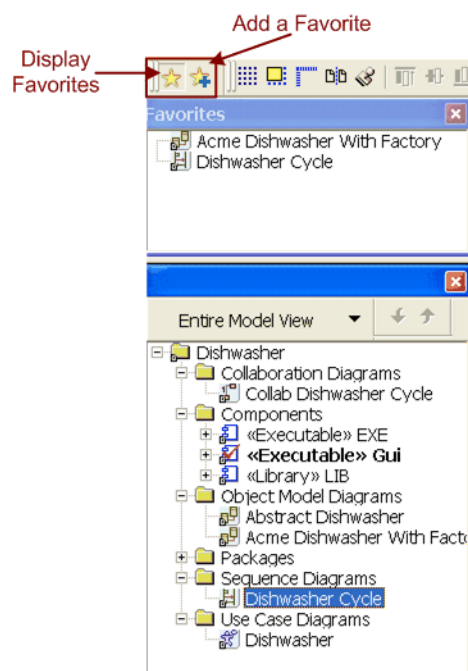
You can copy all elements within the browser. To copy an element onto a new owner, follow these steps:

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, follow these steps:

1. Display the project in 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, as shown in this example.



Deleting Browser Items

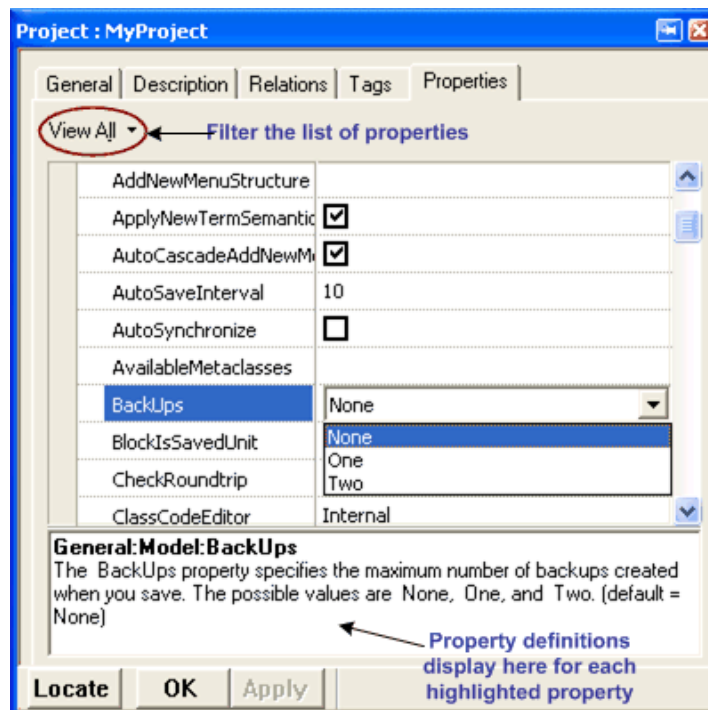
To delete an item from your project, follow these steps:

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.

Creating Backups

To set up automatic backups for your new project, follow these steps:

1. In the browser, right-click the `MyProject` item in the browser list.
2. Select **Features** from the pop-up menu.
3. Click the **Properties** tab at the top of the dialog box that then displays.
4. Click the drop-down arrow and select **All**.
5. Expand the **General** and then the **Model** property lists. (Rhapsody properties descriptions use a notation method with double colons to identify the location of a specific property, for example, `General::Model::BackUps`.)
6. Locate the **BackUps** property and select **Two** from the pull-down menu to create up to two backups for each project. Note the Filter and definition features, shown in this illustration.



7. Click **OK** to save this property change.

After this change, saving a project more than once creates `<projectname>_bak2.rpy`, which contains the most recent backup and the previous version is in `<projectname>_bak1.rpy`. To *restore* an earlier version of a project, you can open either of these backup files.

Naming Conventions and Guidelines

To assist all members of your team in understanding the purpose of individual items in the model, it is a good idea to define naming conventions. These conventions help team members to read the diagram quickly and remember the model element names easily.

Note

Remember that the names used in the Rhapsody models are going to be automatically written into the generated code. Therefore, the names should be simple and clearly label all of the elements.

Standard Prefixes

Lower and upper case prefixes are useful for model elements. The following is a list of common prefixes with examples of each:

- ◆ Event names = “ev” (evStart)
- ◆ Trigger operations = “op” (opPress)
- ◆ Condition operations = “is” (isPressed)
- ◆ Interface classes = “I” (IHardware)


Guidelines for Naming Model Elements

The names of the model elements should follow these guidelines:

- ◆ Class names begin with an upper case letter, such as “System.”
- ◆ Operations and attributes begin with lower case letters, such as “restartSystem.”
- ◆ Upper case letters separate concatenated words, such as “checkStatus.”
- ◆ The same name should not be used for different elements in the model because it will cause code generation problems. For example, no two elements, such as a class, an interface, and a package, should not have exactly the same name.

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 Rhapsody

To close the project, follow these steps:

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**.

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. However, source control software does not support graphical comparisons.

The 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.

A Rhapsody unit is any project or portion of a project that can be saved as a separate file. The following are some examples of Rhapsody units with the file extensions for the unit types:

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

DiffMerge Tool Features

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 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 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

Starting the DiffMerge Tool

The DiffMerge tool can be launched from inside the Rhapsody interface or outside Rhapsody. There are slight differences in the file selection process of the tool depending on whether it is launched inside or outside Rhapsody. Otherwise, all of the DiffMerge capabilities are the same.

Note

For more information about the DiffMerge tool, refer to the *Team Collaboration Guide*.

Rhapsody and Eclipse Basics

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

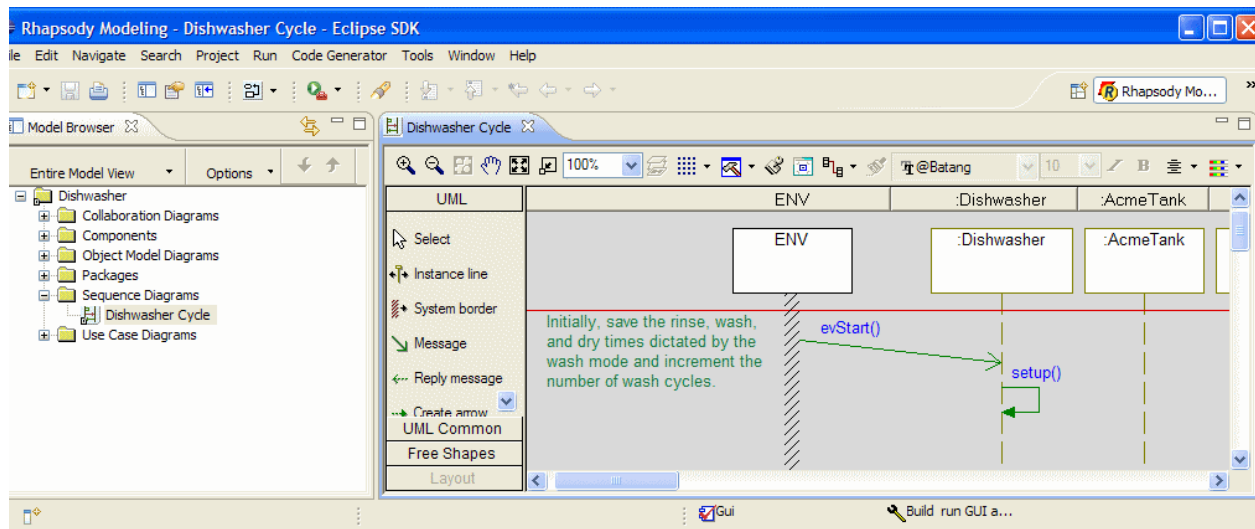
- ◆ **Workflow Integration:** allows you to import a Rhapsody C or C++ project into *Eclipse* using the Rhapsody plug-in
- ◆ **Platform Integration:** allows you to create and work with your *Microsoft Windows* C, C++, or Java project in the *Eclipse* interface

For additional installation and setup instructions, refer to the *Rhapsody Installation Guide*.

Standard Rhapsody and the Platform Integration

The Eclipse Platform Integration of Rhapsody requires a multi-language Rhapsody license. The standard Rhapsody and the Eclipse Platform Integration both use the same repository so that you may switch between the two interfaces if desired.

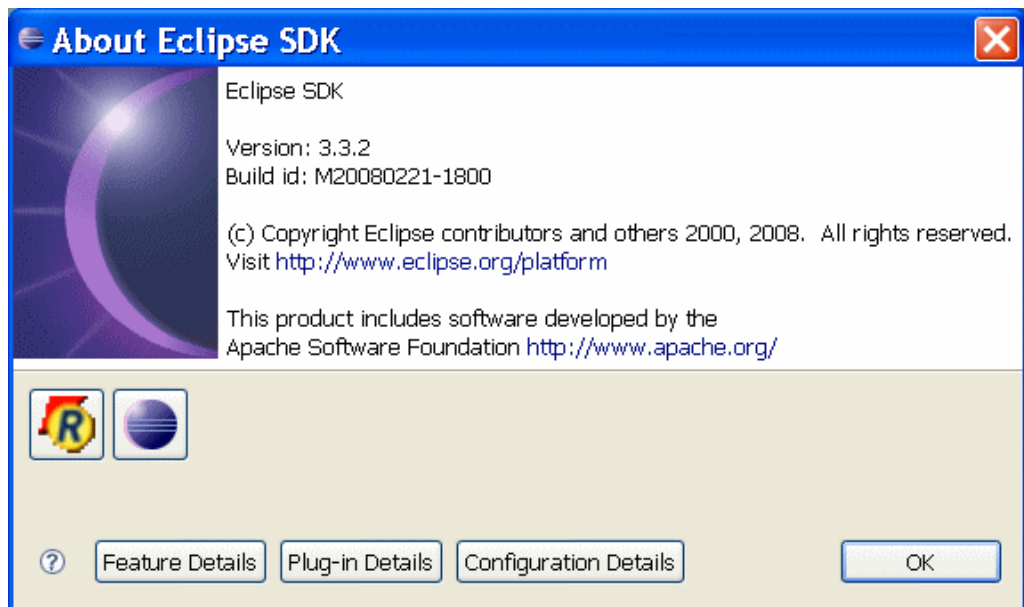
The standard Rhapsody interface elements displayed in Eclipse have the same features as in the stand-alone version except that the icons associated with a specific window are displayed at the top of the window, as in this example.



Creating a Rhapsody Project in Eclipse

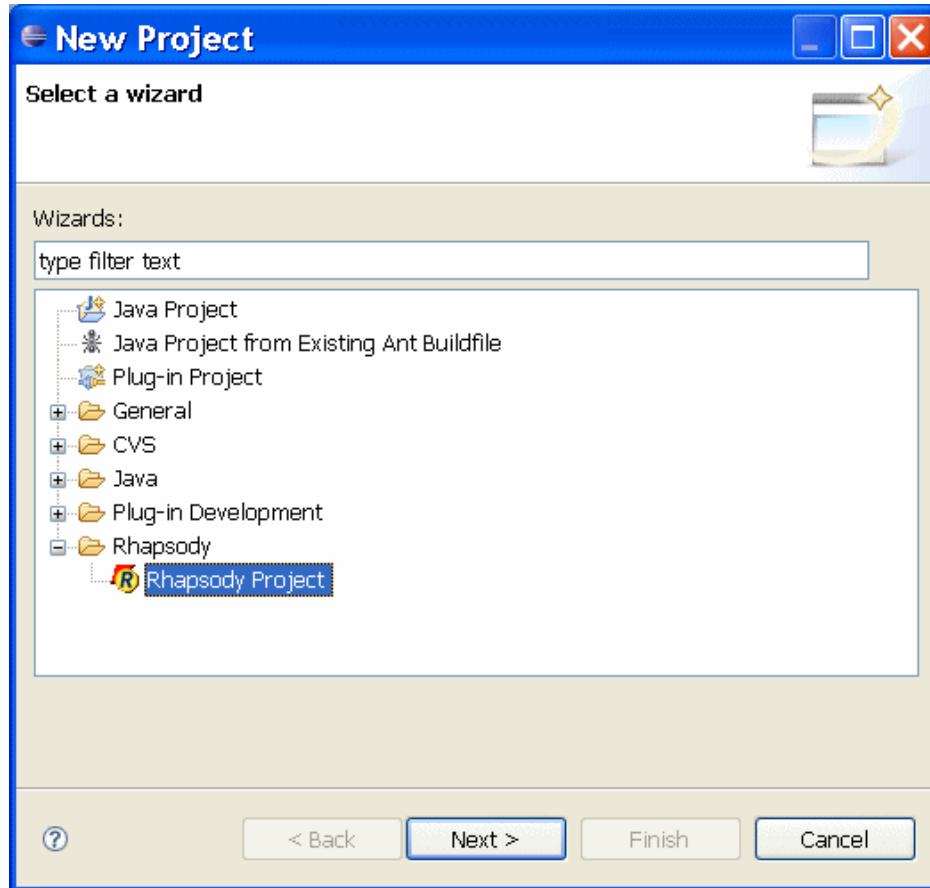
To create a new Rhapsody project in Eclipse, follow these steps:

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, refer to the Eclipse setup instructions in the *Rhapsody Installation Guide*.

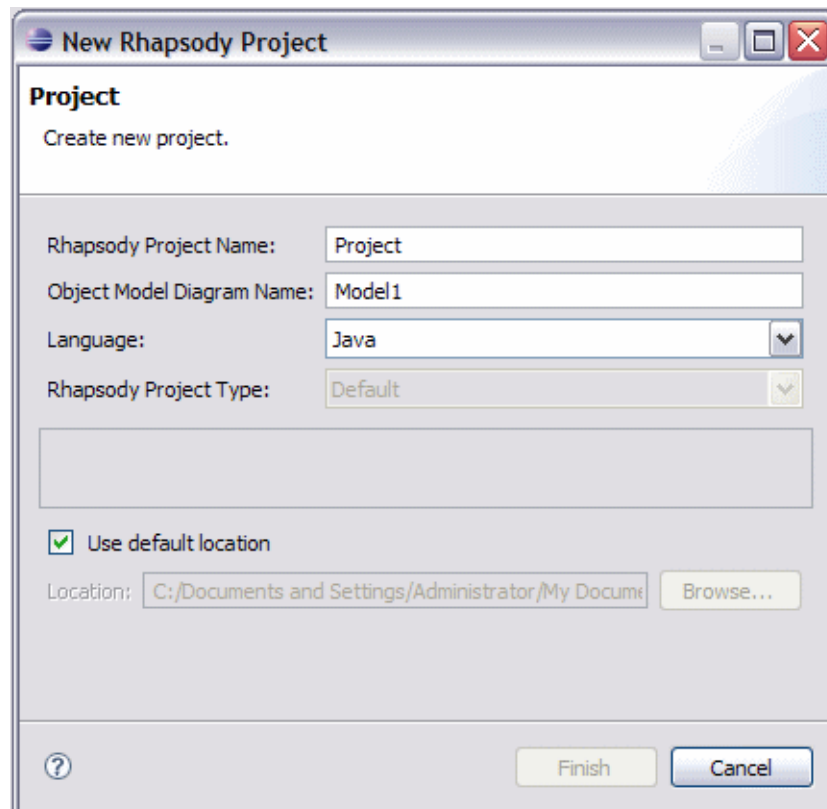


3. To create a new Rhapsody project within Eclipse, select **File > New > Project**.

4. In the New Project dialog box (shown below), 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 default project type. 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.

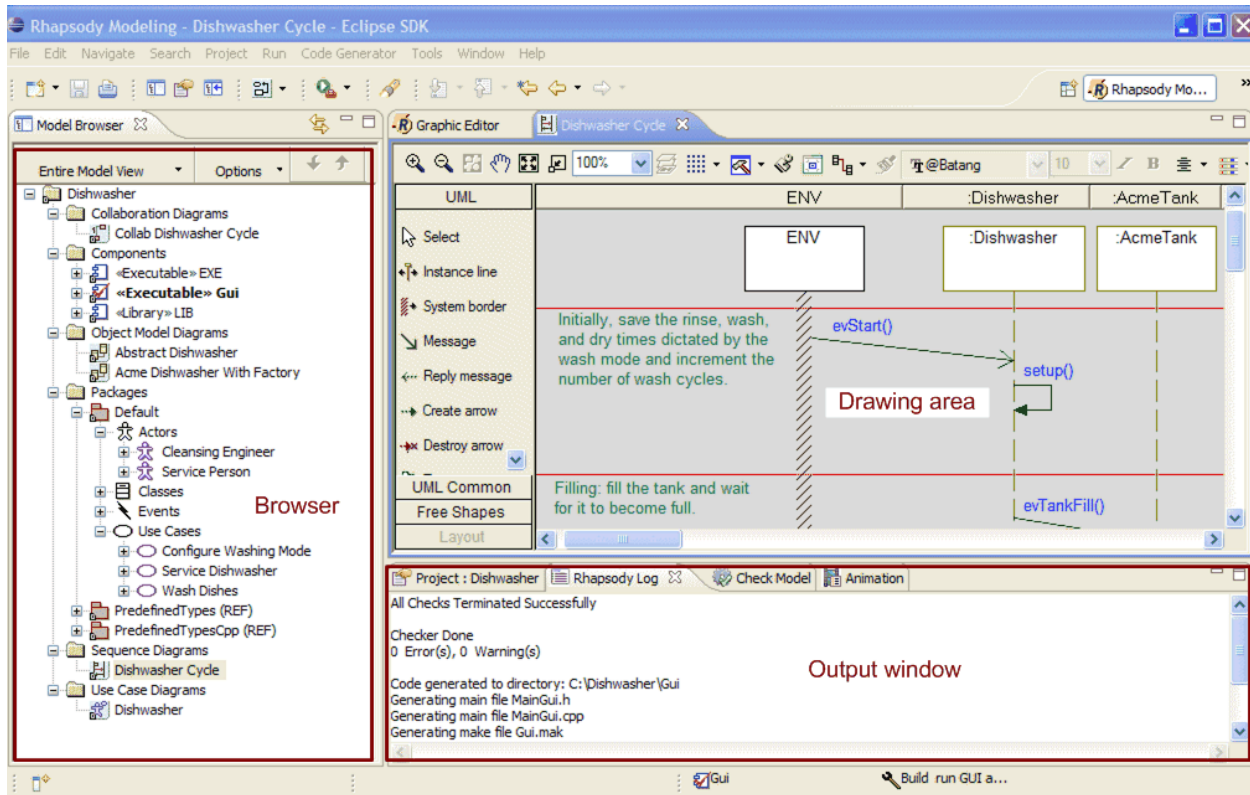
Guided Tour of the Eclipse Platform Integration

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

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

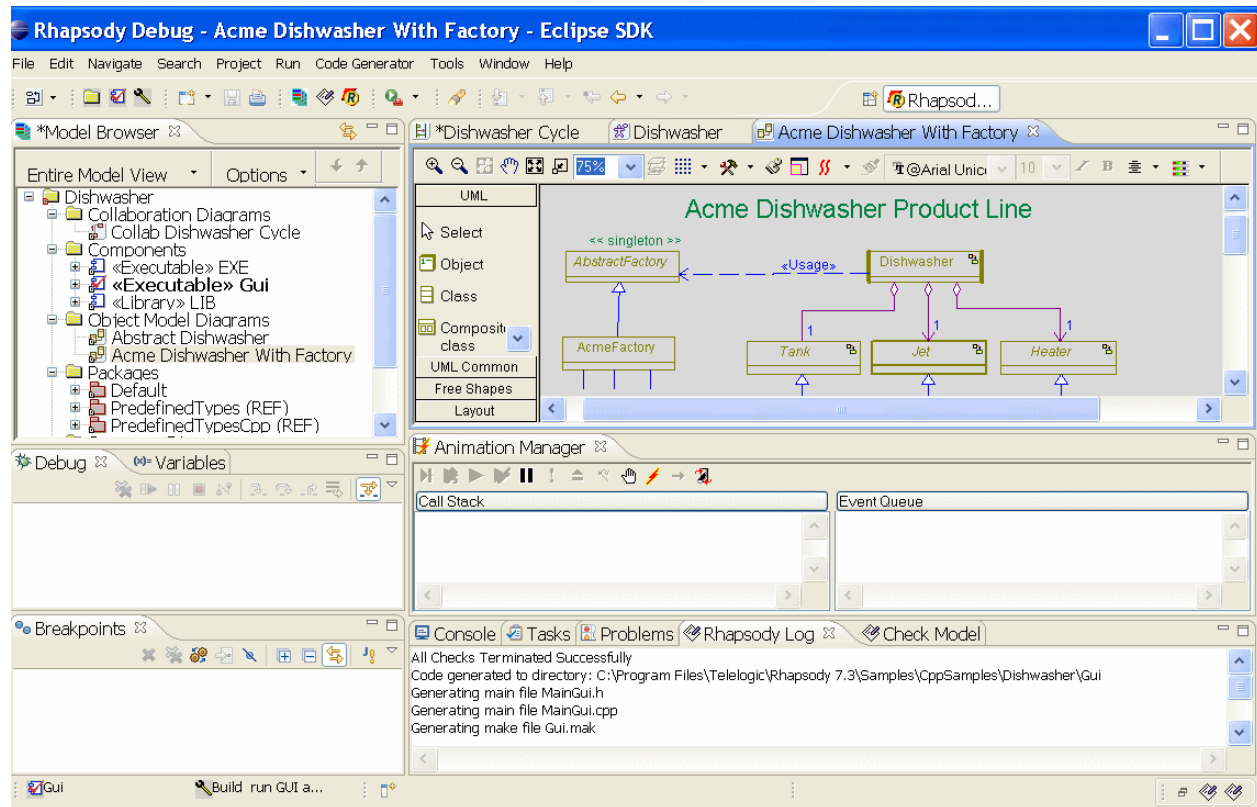
The Eclipse interface displays the following three main Rhapsody interface components (labeled below):

- ◆ Browser (Model Browser tab in Eclipse)
- ◆ Drawing area
- ◆ Output window



Rhapsody and Eclipse Basics

The 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 below.



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

Creating Java Plug-ins


You may create your own 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's 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. Refer to the list of [Rhapsody Samples](#) for more information.

Rhapsody for Software Development

This section provides information specific to developers using the four supported languages in Rhapsody: C, C++, Ada, and Java. The official sample models demonstrate Rhapsody's 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 Rhapsody <version>\Samples directory in your installation to examine the official Rhapsody sample models.
- ◆ Click **Proceed** in the Project Samples area of the Welcome screen.
- ◆ Click the Samples  icon above the Welcome screen.

Note

The official sample models in the Samples directory (see the [Rhapsody Samples by Development Environment](#) list) are different from the models created when following the instructions in the language-specific tutorials in the 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 Rhapsody product to demonstrate the overall scope of the product. To provide a quick start in developing with Rhapsody, this section familiarizes developers with these basic Rhapsody features:

- ◆ Using the browser
- ◆ Accessing diagrams
- ◆ Using the **Drawing** toolbar
- ◆ Creating diagrams
- ◆ Using the Features dialog box
- ◆ Different types of diagrams and their uses
- ◆ Generating code

It is helpful to read through the quick start information for your development code before progressing to the full tutorial for that language. You may also want to examine the quick start information for other languages since those sections all have different Rhapsody examples.

Quick Start with C

For an in depth understanding of using Rhapsody in C, refer to the *Rhapsody in C Tutorial* to creates a elevator model from beginning to end and includes simulations of both statecharts and sequence diagrams. To access the C tutorial, in Rhapsody's **Help** menu select **List of Books** and click the **Rhapsody in C Tutorial** item in the list.

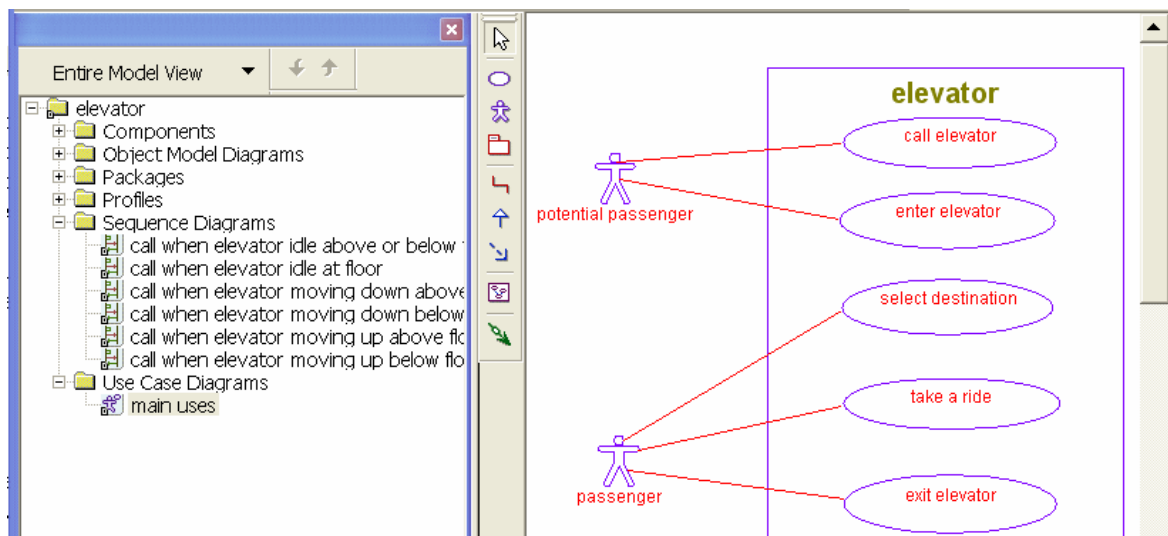
The following quick start for C developers explores the official Rhapsody elevator sample provided in your Rhapsody installation.

Opening the Elevator Model

To start Rhapsody and open the elevator model, follow these steps:

1. Launch Rhapsody as described in [Starting Rhapsody](#).
2. Select **File > Open** from the main menu and navigate to the `Samples\CSamples\Elevator` directory in the Rhapsody directories.
3. Select the `elevator.rpy` project file and the project opens.
4. To display the diagrams in the drawing area, expand the `Sequence Diagrams` and `Use Case Diagrams` items in the browser on the left. Double-click to open the first sequence diagram and the use case diagram. At this point, your project should resemble the following diagram.

Note: Two external actors are shown interacting with the elevator system that contains five use cases



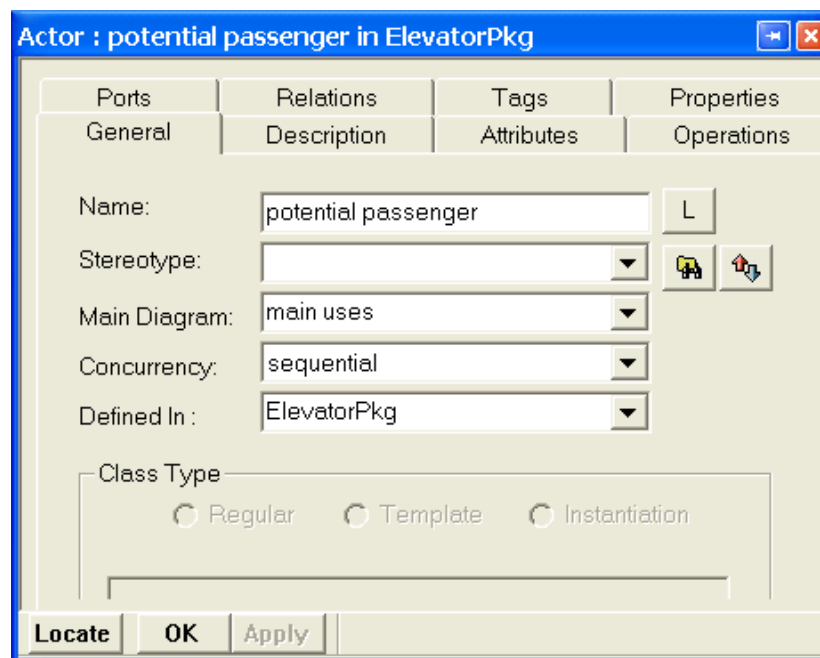
Saving the Sample C Project

In order to avoid changing the official Rhapsody in C elevator model, save the opened project in a different directory. To save a copy of the official sample, follow these steps:

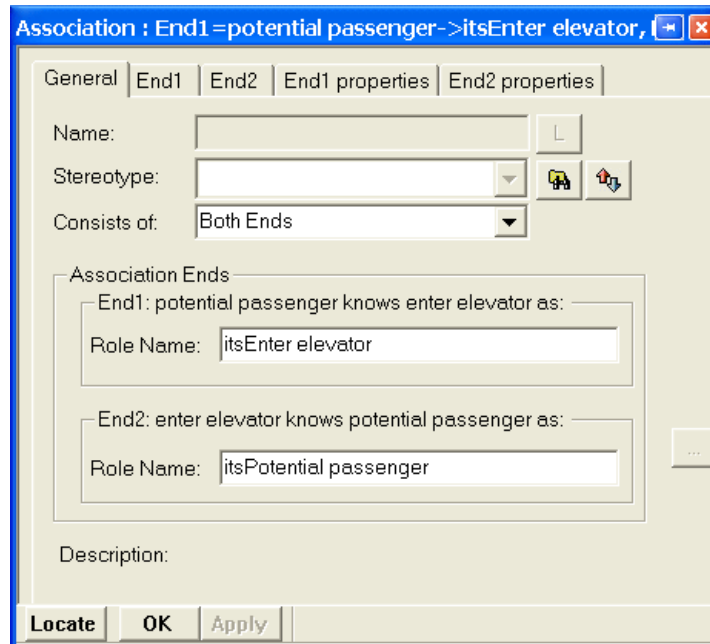
1. Select **Save As** from the **File** main menu.
2. Use the pull-down menu at the top of the dialog box to select a different directory, such as `C:\Temp`. Click **Save** to put the copy of the model into the selected directory.
3. Save the `gui`, `Radio_gui`, and `Sounds` folders in the same manner.

Examining the Use Case Diagram

1. Double-click the `Potential Passenger` actor in the use case diagram to display the Features dialog box (as shown in the following figure) defining the actor. Examine all of the settings in the tabs.



2. Double-click one of the lines between the actor and an oval to examine the description of that relationship, as shown in this example. Click through the tabs to see the descriptions of the lines and their properties.



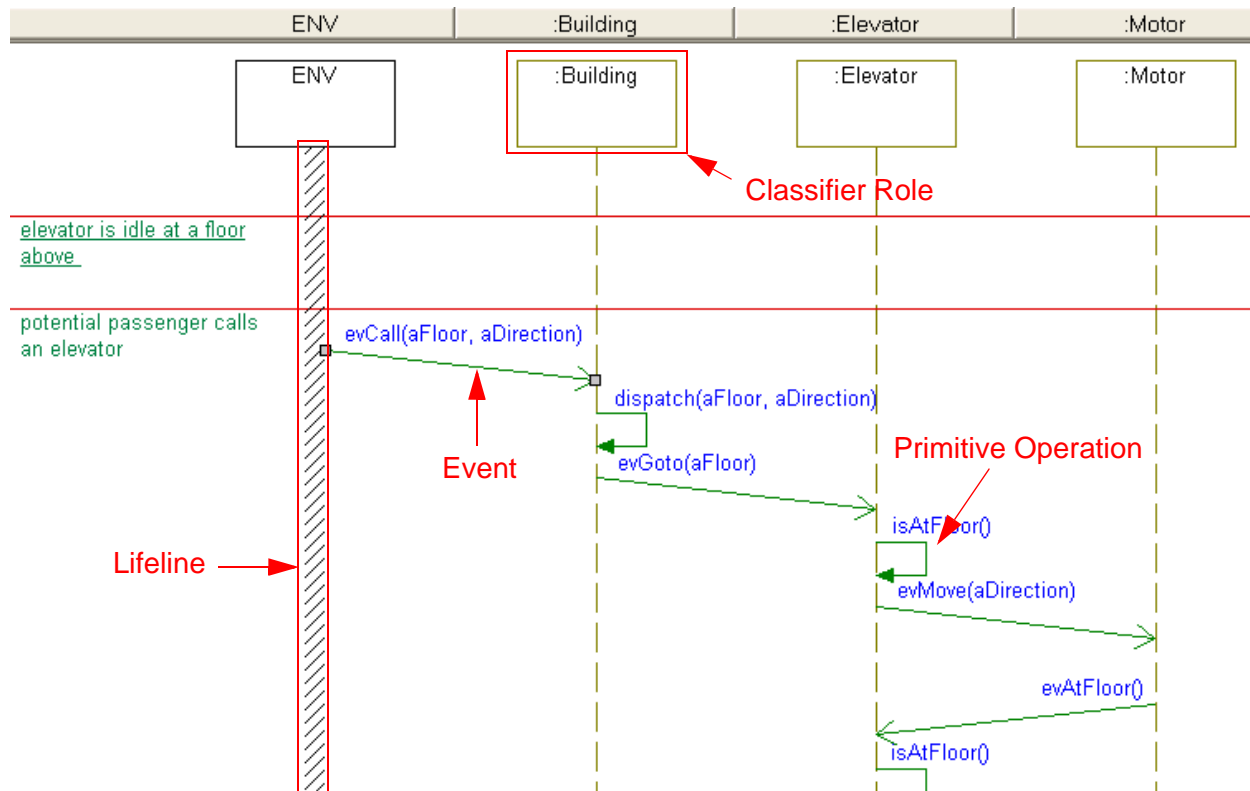
3. Move one end of the line and then examine the description of the line to see how Rhapsody automatically changed its description.
4. Examine the other items in the use case diagram to see what the model is expected to do. Change items to see that effects on the relationships and definitions of items.

Defining Messages Using Sequence Diagrams

Sequence diagrams define the message exchanges in your model. These diagrams show the scenarios of message exchanges between the actors and items defined in the use case diagram. Sequence diagrams have many uses including analysis and design scenarios, execution traces, and expected behavior in test cases.

To examine the messages in the C sample elevator model, follow these steps:

1. In the browser, double-click the call when elevator idle above or below floor sequence diagram in the Sequence Diagrams section. The sequence diagram appears, as shown in the following figure.



2. The arrows shown in the previous figure running between the lifelines are all `messages`. The closed, solid arrows are `primitive operations`. The arrows that are open are `events`.
 - ◆ `Events` can carry data and are used to trigger various statechart behaviors. For example, the `evGoto` event is sent from the building to the elevator and causes the elevator to go from `idle` to a moving state and carries data (`aFloor`) which can be extracted for the same purpose.
 - ◆ `Primitive Operations` are used to carry executable instructions or commands from one object to another.


Animating the Elevator Model

You can view and debug the behavior of the elevator by animating the model as follows:

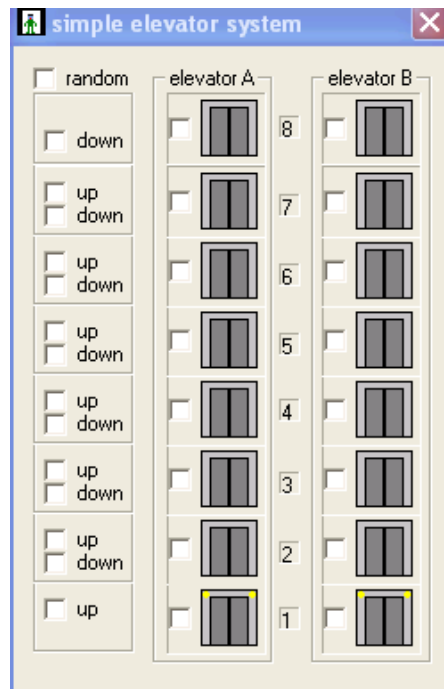
1. In the browser, expand the **ElevatorPkg** package, the **Classes** category, and the **Elevator** class so you can see the underlying categories.
2. Select **Code > Run gui.exe**. A console window opens; minimize the console window.

Running the Animated Model

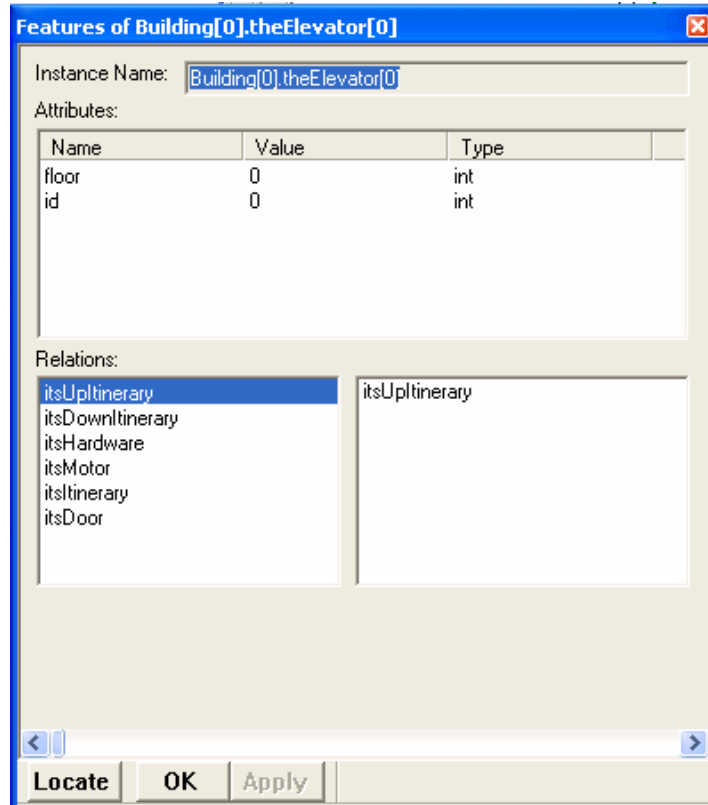
To continue running animation, follow these steps:

1. Click the **Go** icon  in the **Animation** toolbar to begin executing Elevator.
2. In the browser, the **Instances** category displays under the **Elevator** object, which contains the `Building[0].theElevator[0]` instance created as your application runs in animation mode.

A GUI application displays that you will use to send events to the animated elevator model, as shown in the following figure.



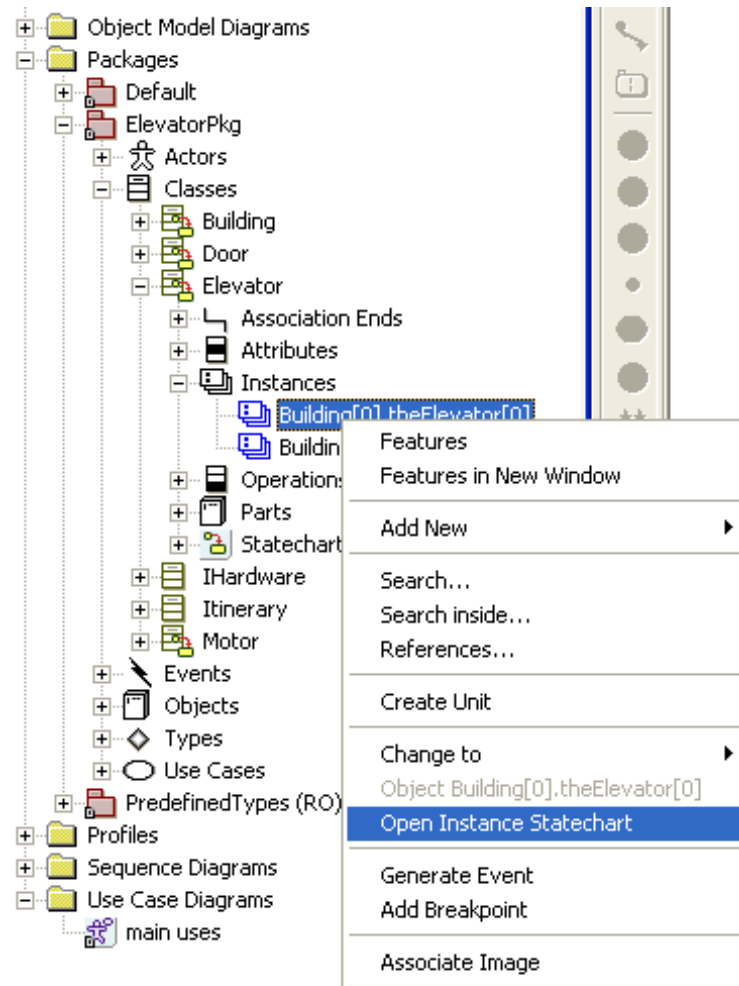
3. In the `Elevator` class, double-click the `Instance` to view its features, as shown in the following figure.



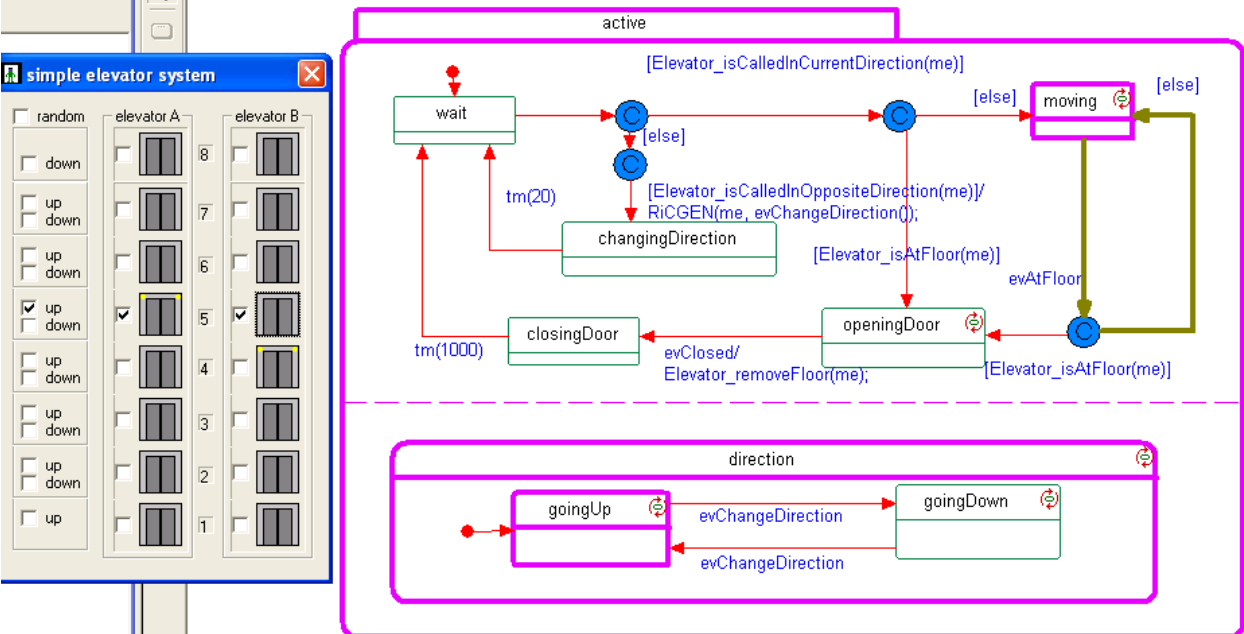
4. Click **OK** to close the Features dialog box.

Animating a Statechart

Next, create an animated statechart by right-clicking the `Building[0].theElevator[0]` instance in the browser and selecting **Open Instance Statechart** from the pop-up menu.



Rhapsody displays an animated statechart in the drawing area, as shown in the following figure.




Note

The active state is highlighted in magenta; inactive states are highlighted in green.

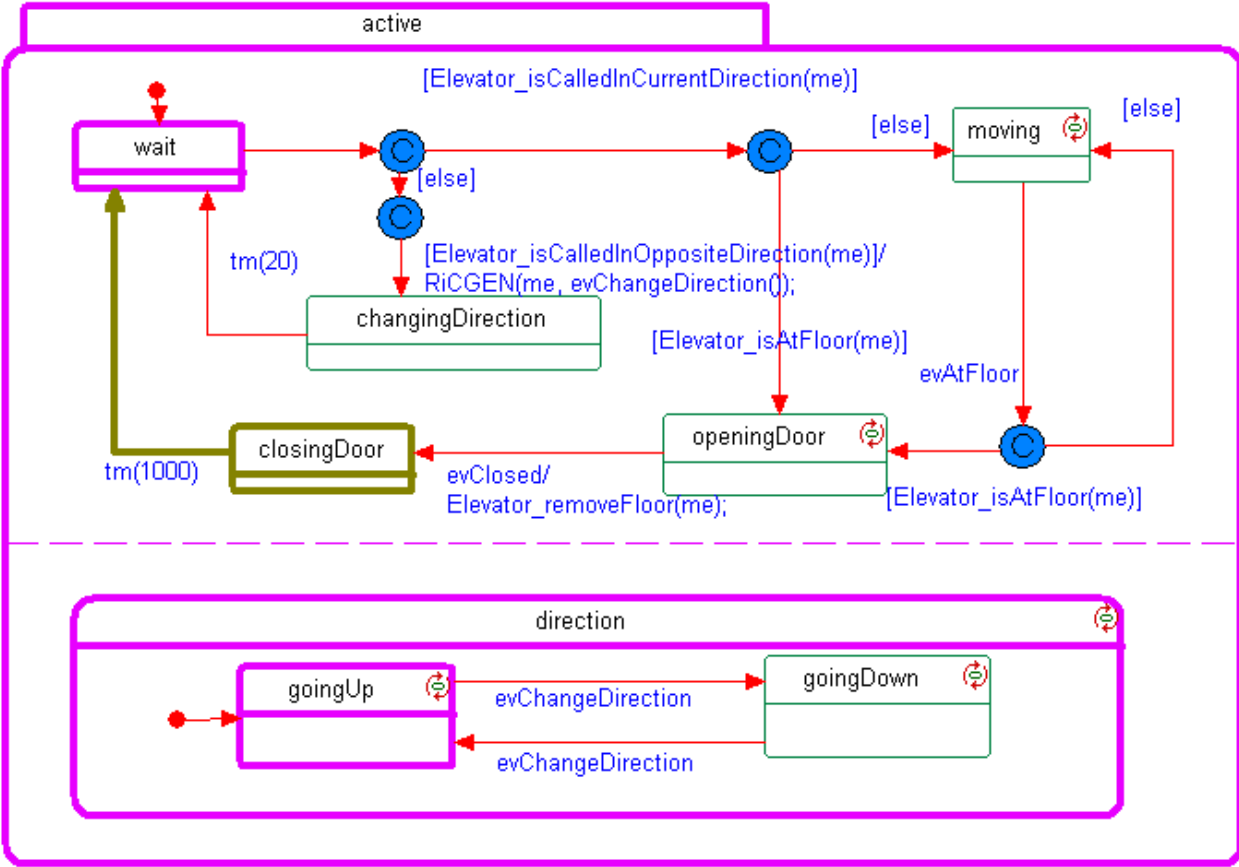
Generating Events

You can debug the animated statechart using events. In this example, you generate the `evCall` event to transition the model from the wait state to the moving state.

To send an `evCall` event to the animated statechart, follow these steps:

1. Click the **Event Generator** icon  in the **Animation** toolbar. The Events dialog box opens.
2. Click **Select**. The Instances Selection dialog box opens.
3. Select `Building[0]` and then click **OK** to dismiss the Instances Selection dialog box and return to the Events dialog box.
4. Select `evCall` from the **Event** pull-down in the Events dialog box.
5. Double-click the argument named `aFloor` and type in a value between 0 and 7. Click **OK**.
6. Double-click on the `aDirection` and type in the value UP or DOWN. Click **OK**.

The statechart indicates that the elevator has been called to the floor indicated by the integer entered in step 5 and is waiting to be given an input (floor number) by the user.



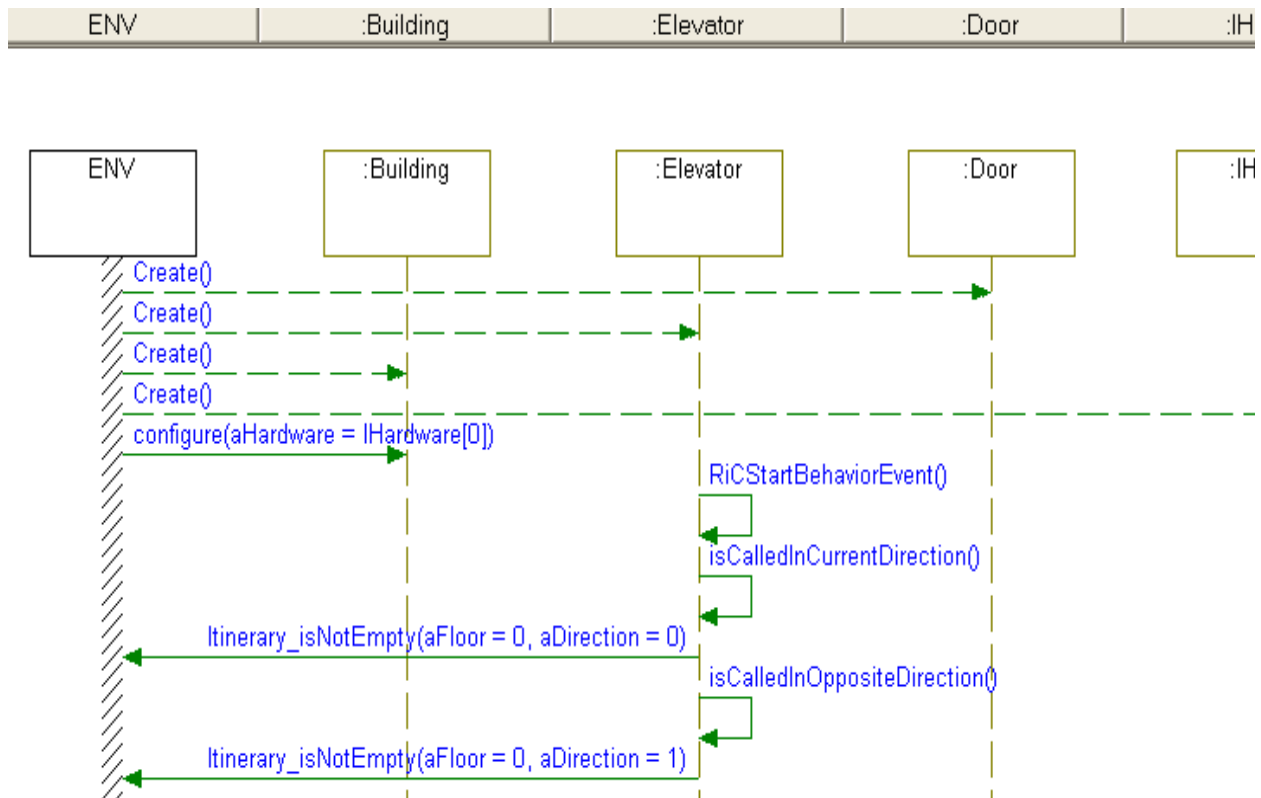
Creating an Animated Sequence Diagram

An *animated sequence diagram* (ASD) shows the messages passed between instances as the application runs.

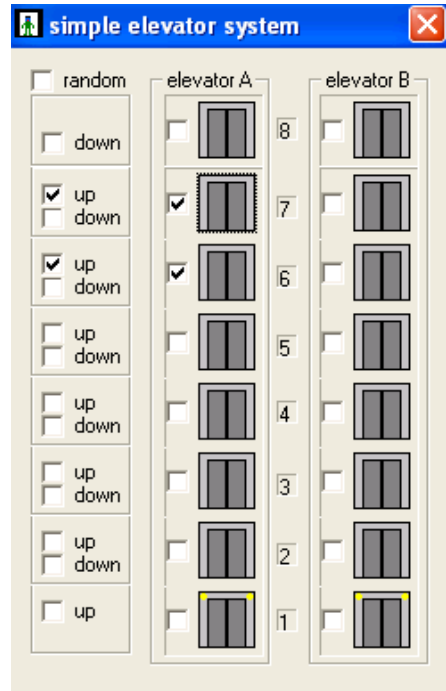
To create an ASD, follow these steps:

1. Select **Tools > Animated Sequence Diagram**. The Open Sequence Diagram dialog box is displayed.
2. Select `call` when `elevator` idle above or below floor, then click **Open**.

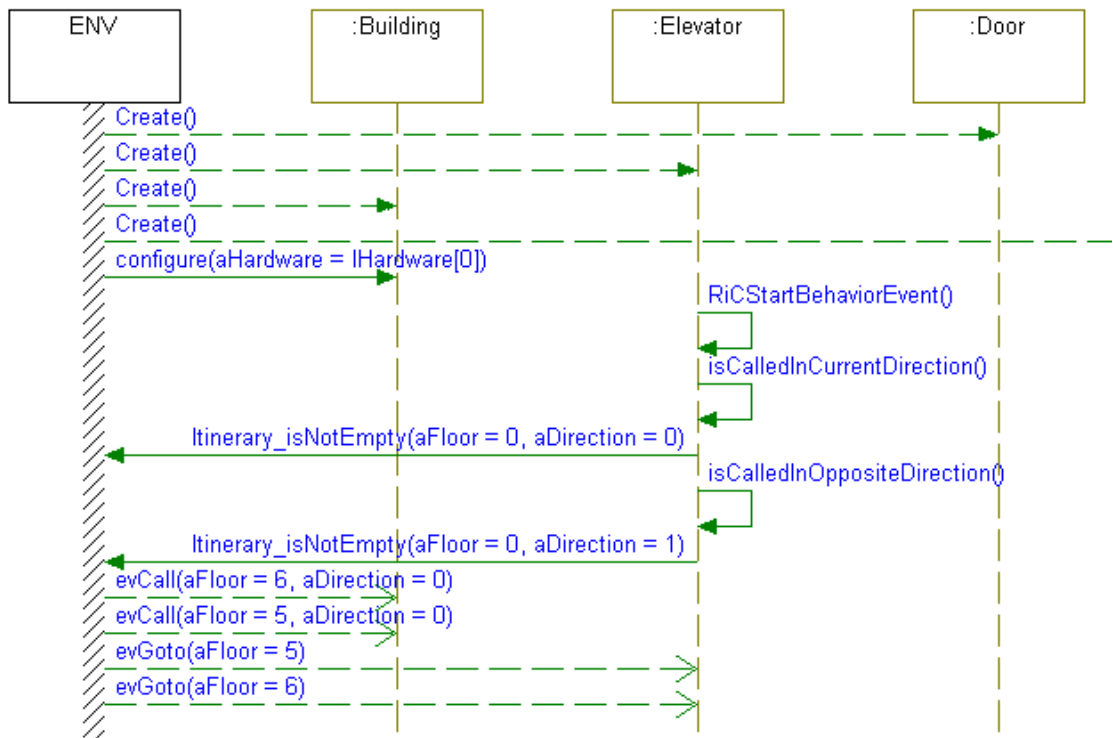
The ASD opens in the drawing area and animates the sequence of events, as shown in the following figure.



- Using the Elevator GUI, click the buttons to send events to the animated elevator. The following example shows the GUI.



The ASD displays the collaboration between the instances in the form of function calls, events, and timeouts as shown in the following example.



Stopping the Program

To stop animation, select **Code > Stop Execution**.

Note

When you quit an animation session or close an ASD, you can save the ASD for future reference. This is useful to compare the results of the current session to those of different execution scenarios. When you close the project or the ASD, Rhapsody asks whether you would like to save your ASD. Click **Yes** to save it.

Quick Start with C++

To gain a more in depth understanding of using Rhapsody in C++ to produce models, use the *Rhapsody in C++ Tutorial* to create a handset model from beginning to end. That tutorial is accessed from Rhapsody's **Help** menu. Select **List of Books** and click the **Rhapsody in C++ Tutorial** item in the list.

This section uses an official Rhapsody sample model of a radio to demonstrate using Rhapsody to analyze, model, design, implement, and verify the behavior of embedded C++ software systems.

Opening the Radio Model

To start Rhapsody and open the radio model, follow these steps:

1. Launch Rhapsody as described in [Starting Rhapsody](#).
2. Select **File > Open** from the main menu and navigate to the `Samples\CPPSamples\Radio` directory in the Rhapsody directories.
3. Select the `Radio.rpy` project file and the project opens.

Saving the Sample C++ Project

In order to avoid changing the official Rhapsody in C++ radio model, save the opened project in a different directory. To save a copy of the official sample, follow these steps:

1. Select **Save As** from the **File** main menu.
2. Use the pull-down menu at the top of the dialog box to select a different directory, such as `C:\Temp`.
3. Click **Save** to put the copy in the selected directory.
4. Save the `gui`, `Radio_gui` and `Sounds` folders from the original Radio project using the same method as the project folder.

Note

You must manually save all of the original project folders to the new directory so that the animation can be successful in later procedures.

Viewing Diagrams

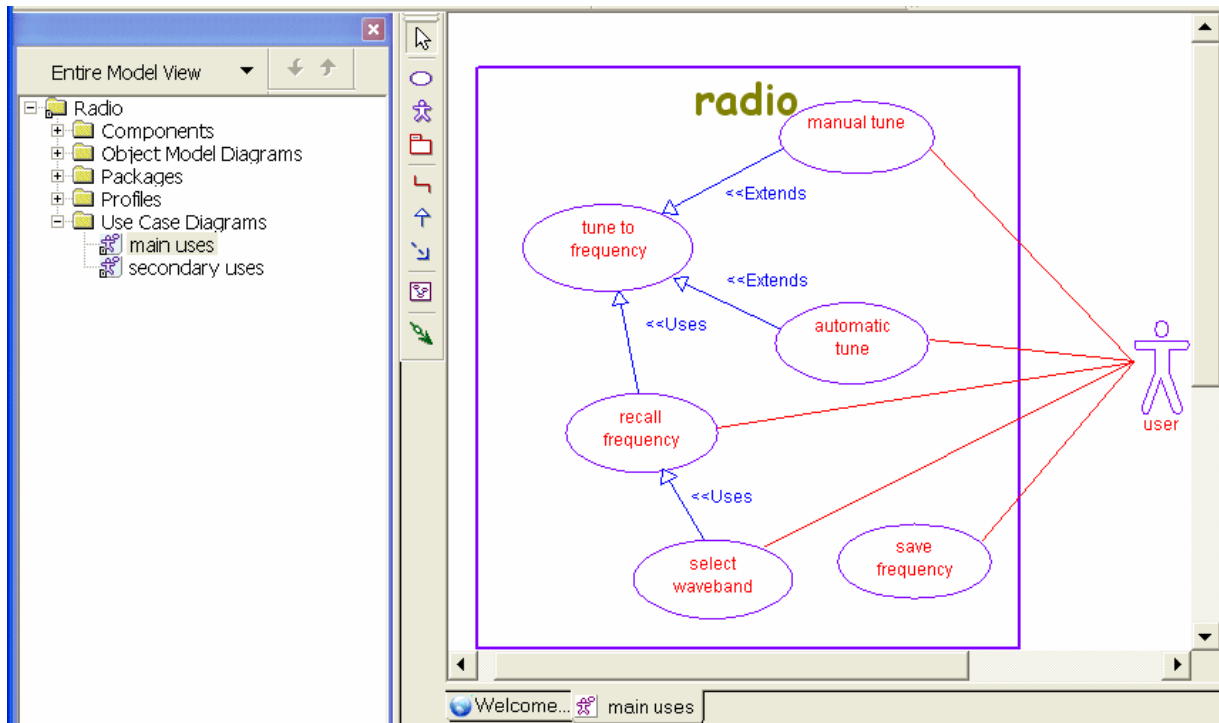
This section demonstrates important features of the diagrams that construct the Radio model using C++.

Use Case Diagrams

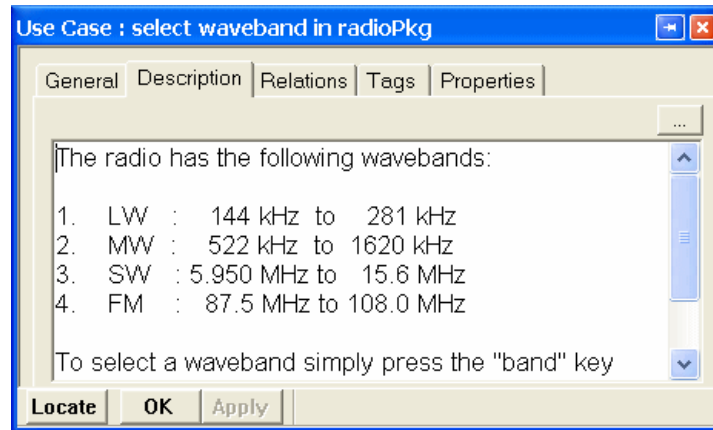
A *use case diagram* (UCD) captures the functional requirements and external actors for a system. The diagrams display in the drawing area.


To view a UCD, follow these steps:

1. In the browser, expand the Use Case Diagrams category.
2. Double-click the main uses UCD. The UCD opens in the drawing area.



3. Double-click the `select waveband` oval in the use case diagram to view its features. The Features dialog box opens. Select the **Description** tab to view the information, as shown in the following figure.

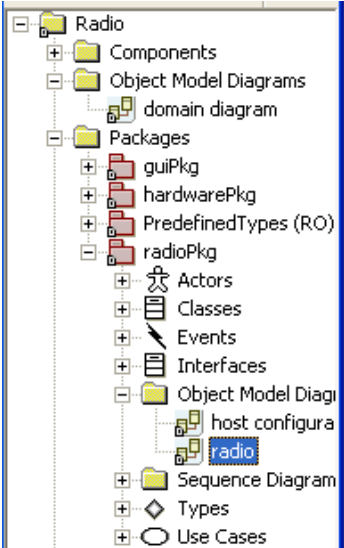


4. To use the Text Editor for the Description field, click the ellipsis button , and the full text editor appears for you to make changes.
5. Click **OK** to close the text editor and **OK** again to close the Features dialog box.

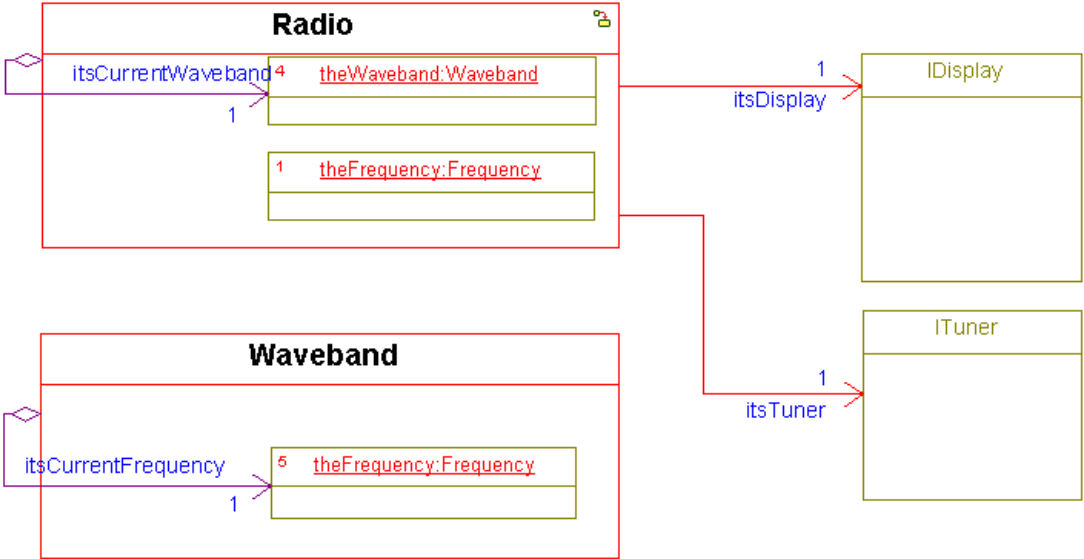
Object Model Diagrams

An *object model diagram* (OMD) shows the static structure of the classes and instances in an object-oriented software system and the relationships between them. To open the radio OMD, follow these steps:

- 1. In the browser, expand the Packages category, the radioPKG, and the Object Model Diagrams category to reveal this structure.



- 2. Double-click `radio` in the Object Model Diagram folder. The radio OMD displays in the drawing area, as shown in the following figure.



The `Radio` class (the large, red box at the top of the diagram) is the composite class containing instances of the classes `Waveband` and `Frequency`. Each of these classes has a number in its upper, left-hand corner to indicate the instantiated number of instances. In this case, `Radio` has one frequency of its own and four wavebands.


`Radio` is related to `theWaveband` through an aggregation relation. This enables `Radio` to point to one individual instance of `Waveband` as its current `Waveband` (`itsCurrentWaveband`).

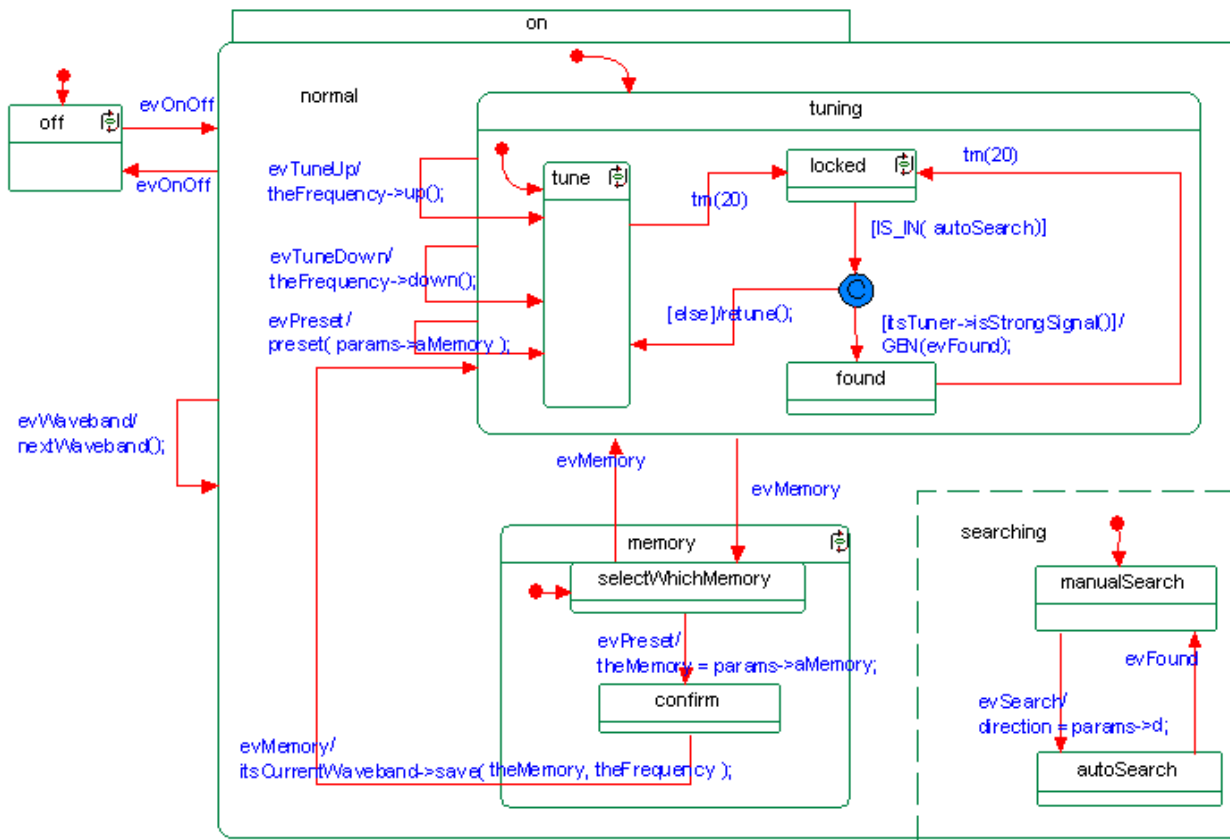
The class definition for the `Waveband` class is shown under the `Radio` class, which defines the `Waveband` instance (`theWaveband`) inside of `Radio`. It is shown here because `Waveband` is also a composite class containing five instances of the `Frequency` class, `theFrequency`. An aggregation relation is used to relate a `Waveband` instance to one individual instance of `Frequency` as its current `Frequency` (`itsCurrentFrequency`).

Because `Rhapsody` shows only one layer of nesting, the `Waveband` class is shown to the side of `Radio` as if it were a separate class. However, when taken together, the radio OMD shows that an instance of `Radio` contains a single frequency of its own along with four wavebands, and each waveband contains five frequencies.

Statecharts

A *statechart* defines the behavior of individual classes in the system.

- To open the statechart for the radio in the Object model diagram, right-click the Radio class, then select **Open Statechart** from the pop-up menu, or click the statechart icon  in the upper right corner of the Radio class. The Radio statechart opens in the drawing area.



A statechart has the following elements:

- ◆ **States**—Each state contains a label and specified code to execute on entry (actions on entry) and exit (actions on exit) from the state. Also included are reactions in state, which are specified like transitions. A state is shown as a green box in the statechart (for example, the tune state).
- ◆ **Transitions**—Transitions enable the model to go from state to state during execution. Each transition has a trigger, an optional guard, and optional code to execute. When the state receives an event of type equal to the trigger specified, the transition takes place.

Transitions are shown as red arrows. A transition's trigger, guard, and action code are shown adjacent to the transition in blue text in the following format:

```
trigger [guard] /code
```

- ◆ **Default connector**—A default connector indicates the initial state and is shown as a red arrow with a point on the end.

You can double-click the states or transitions to see their features.

Animating the Radio Model


You can view and debug the behavior of the radio by animating the model as follows:

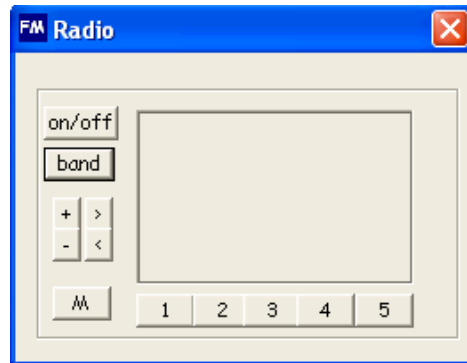
1. Close the Radio statechart and radio OMD.
2. In the browser, expand the `RadioPkg` package, the `Classes` category, and the `Radio` class so you can see the underlying categories.
3. Select **Code > Run gui.exe**. A console window opens; minimize the console window.

In the browser, the `Instances` category displays under the `Radio` object, which contains the `Radio[0]` instance created as your application runs in animation mode.

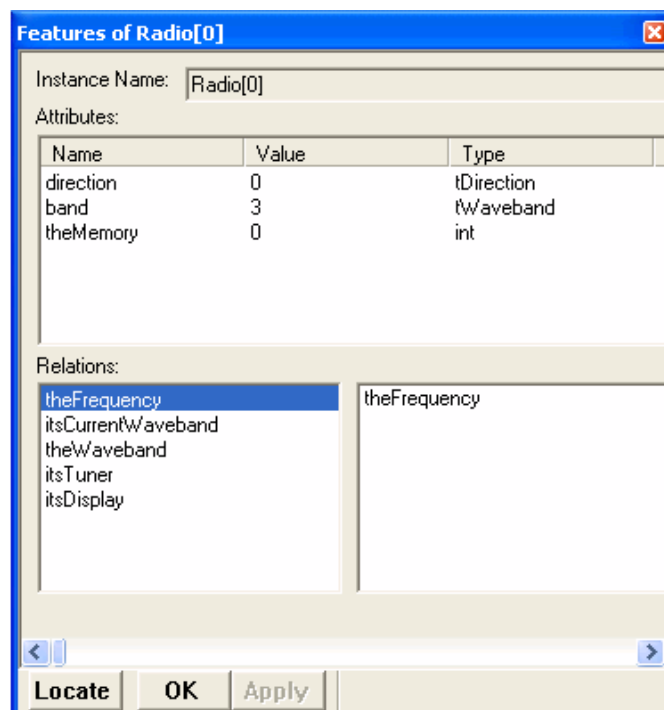
Running the Animated Model

To continue running animation, follow these steps:

1. Click the **Go** icon  in the **Animation** toolbar to begin executing Radio. A GUI application displays, as shown below, that you use to send events to the animated radio model.



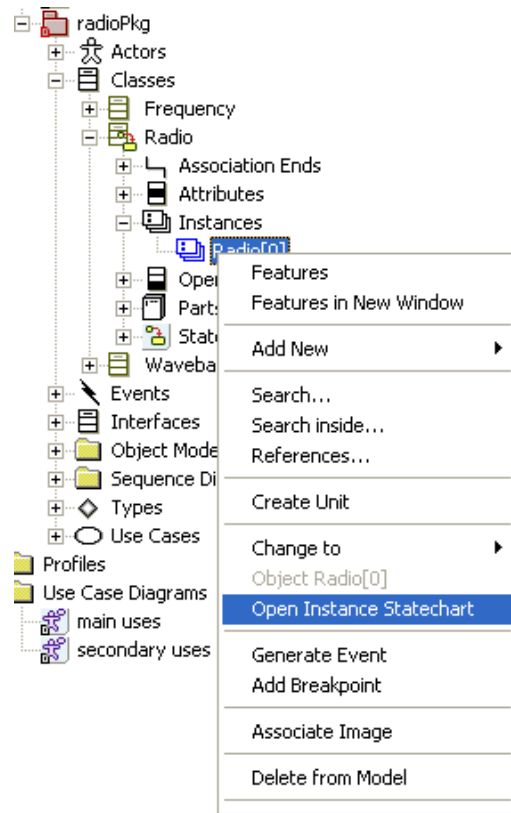
2. In the `Radio` class, double-click the Instance to view its features, as shown here.



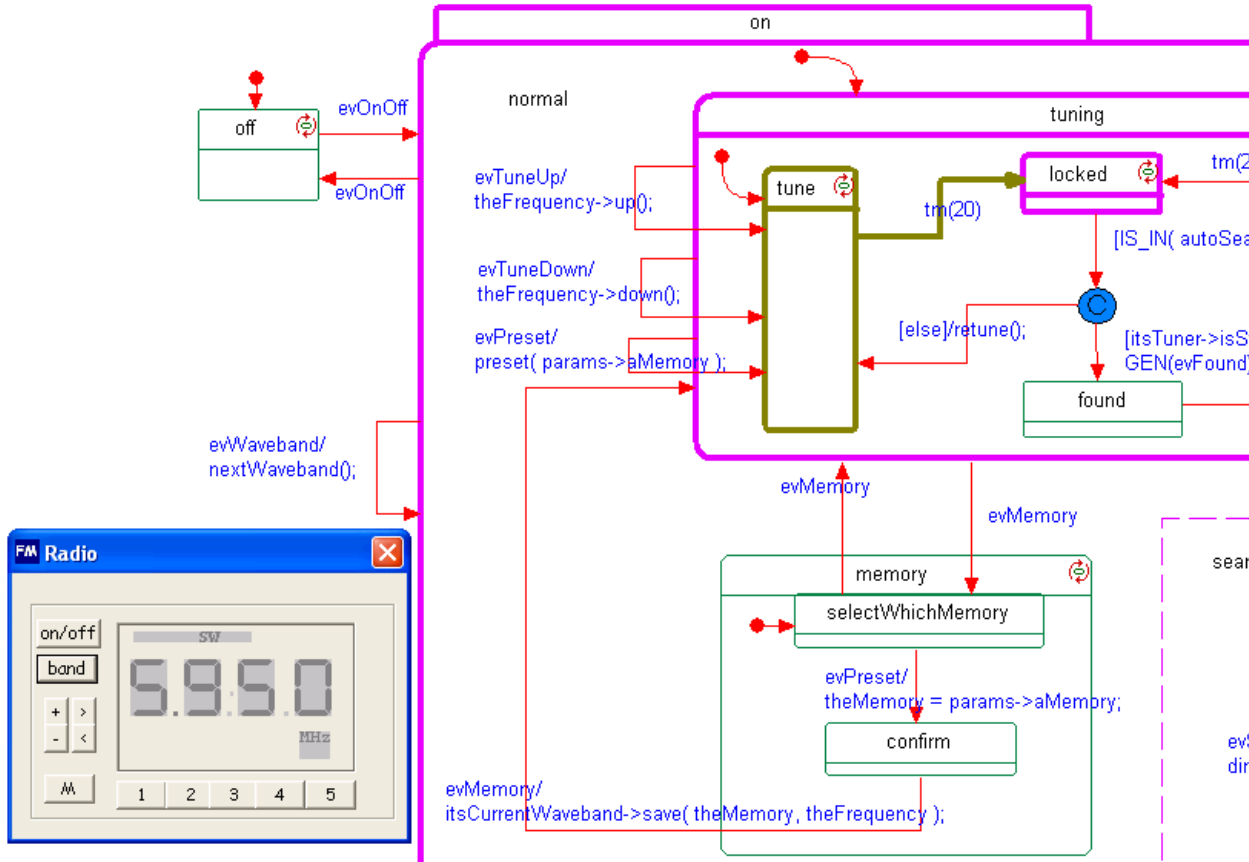
3. Click **OK** to close the Features dialog box.

Animating a Statechart

Next, create an animated statechart by right-clicking the `Radio[0]` instance in the browser and selecting **Open Instance Statechart** from the pop-up menu.



Rhapsody displays an animated statechart in the drawing area, as shown in the following figure.




Note

The GUI display with the animated statechart.

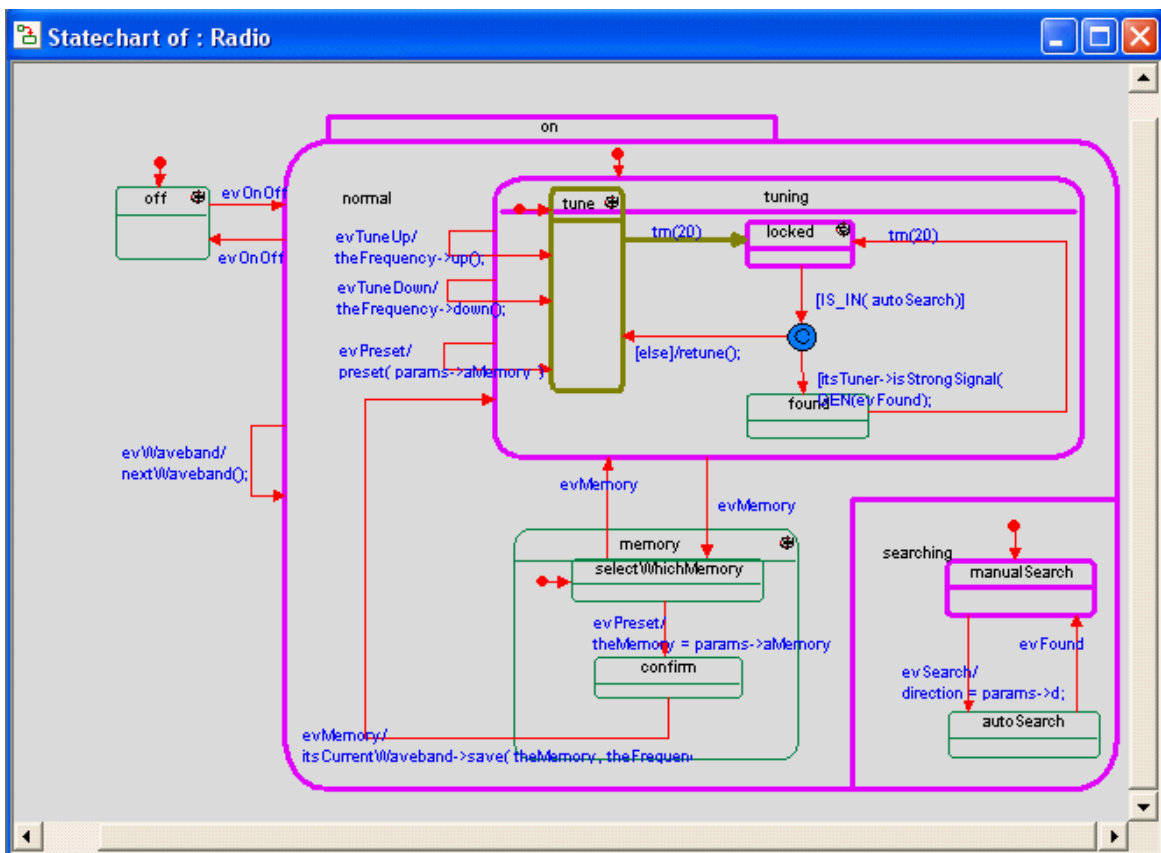
Generating Events

You can debug the animated statechart using events. In this example, you generate the `evOnOff` event to transition the model from the off state to the on state.

To send an `evOnOff` event to the animated statechart, follow these steps:

1. Click the **Event Generator** icon  in the **Animation** toolbar. The Events dialog box opens.
2. Click **Select**. The Instances Selection dialog box opens.
3. Select the instance `Radio[0]`, then click **OK** to dismiss the Instances Selection dialog box and return to the Events dialog box.
4. Select `evOnOff` from the **Event** pull-down, then click **OK**.

The statechart indicates that the radio has been turned on, as shown in the following example.



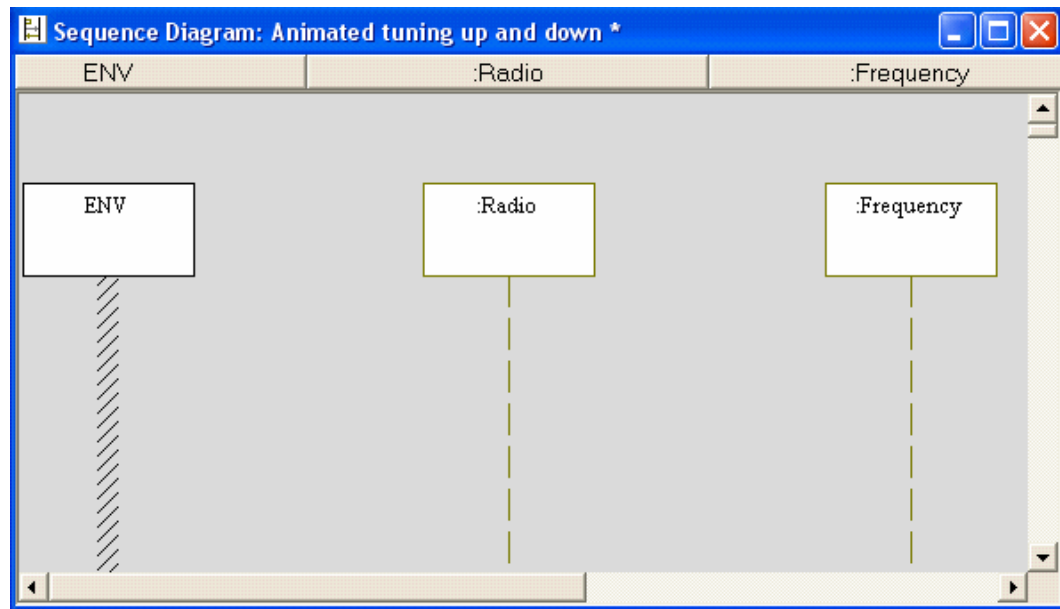
Creating an Animated Sequence Diagram

An *animated sequence diagram* (ASD) shows the messages passed between instances as the application runs.

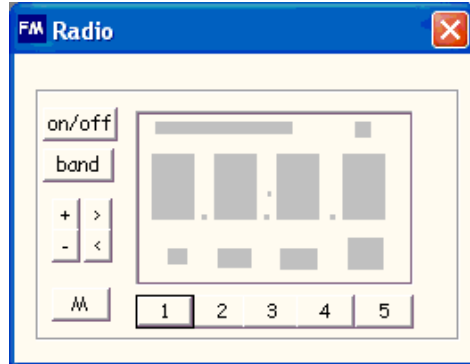
To create an ASD, follow these steps:

1. Select **Tools > Animated Sequence Diagram**. The Open Sequence Diagram dialog box is displayed.
2. Expand `radioPkg`, select `tuning up and down`, then click **Open**.

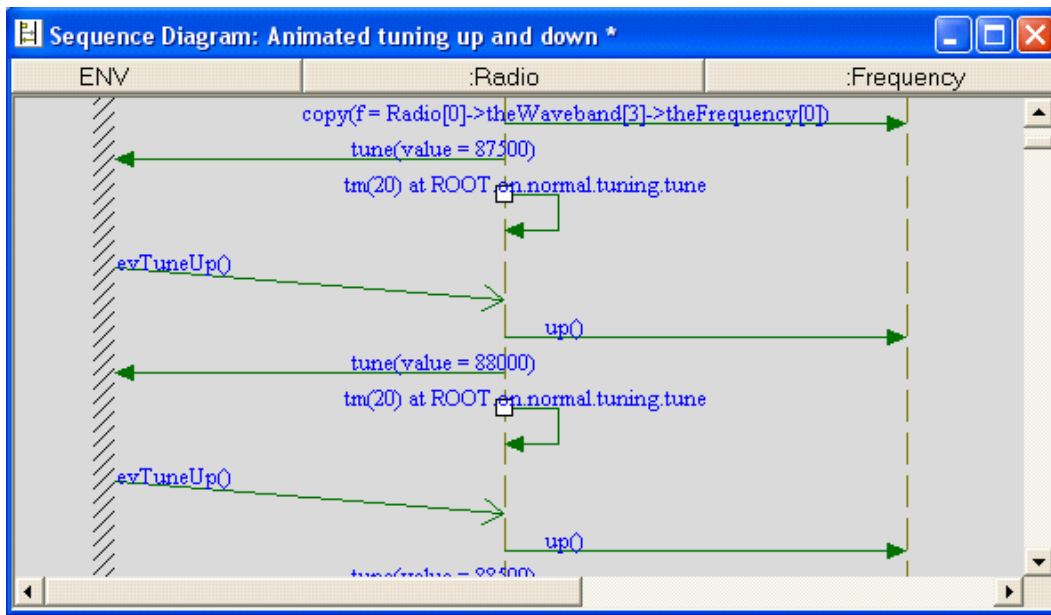
The ASD opens in the drawing area, as shown in the following figure.



- Using the Radio GUI, click the buttons to send events to the animated radio. The following example shows the GUI.



The ASD displays the collaboration between the instances in the form of function calls, events, and timeouts, as shown in the following example.



Stopping the Program

To stop animation, select **Code > Stop Execution**.

Note

When you quit an animation session or close an ASD, you can save the ASD for future reference. This is useful to compare the results of the current session to those of different execution scenarios. When you close the project or the ASD, Rhapsody asks whether you would like to save your ASD. Click **Yes** to save it.

Quick Start with Ada

To gain a more in depth understanding of using Rhapsody in Ada to produce models, use the *Rhapsody in Ada Tutorial* to create a dishwasher model from beginning to end. To access that tutorial, select the Rhapsody **Help** menu and select **List of Books**. Click the **Rhapsody in Ada Tutorial** item in the list to launch a PDF version of the manual.

This section uses an official Rhapsody sample, the RS232 project, to demonstrate some Rhapsody features to analyze, model, design, implement, and verify the behavior of embedded systems software.

Opening the RS232 Sample Project

Follow these steps to locate and open Rhapsody's Ada RS232 sample:

1. Launch Rhapsody as described in [Starting Rhapsody](#).
2. From the main menu, select **File > Open**.
3. In the dialog box, navigate to the `Samples\AdaSamples\RS232` directory in the Rhapsody directories.
4. Select the `RiA_RS232.rpy` file. The project opens in Rhapsody.

Saving the Sample Ada Project

In order to avoid changing the official Rhapsody in Ada RS232 project, save the opened project in a different directory. To save a copy of the official sample, follow these steps:

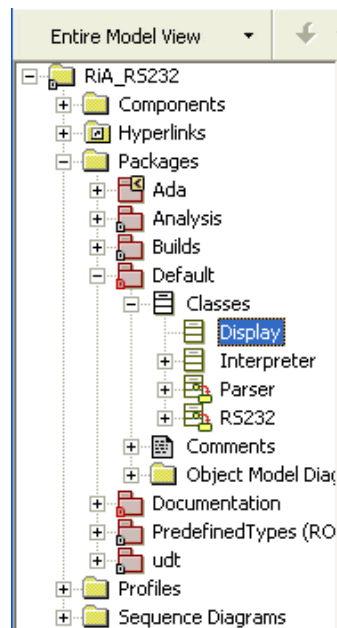
1. Select **Save As** from the **File** main menu.
2. Use the pull-down menu at the top of the dialog box to select a different directory, such as `C:\Temp`.
3. Click **Save** to put the copy in the selected directory.

Creating a Class

Classes define properties that are common to all objects of the type. Classes can contain attributes, operations, events, relations, components, superclasses, types, actors, use cases, diagrams, and other classes.

To create a class, follow these steps:

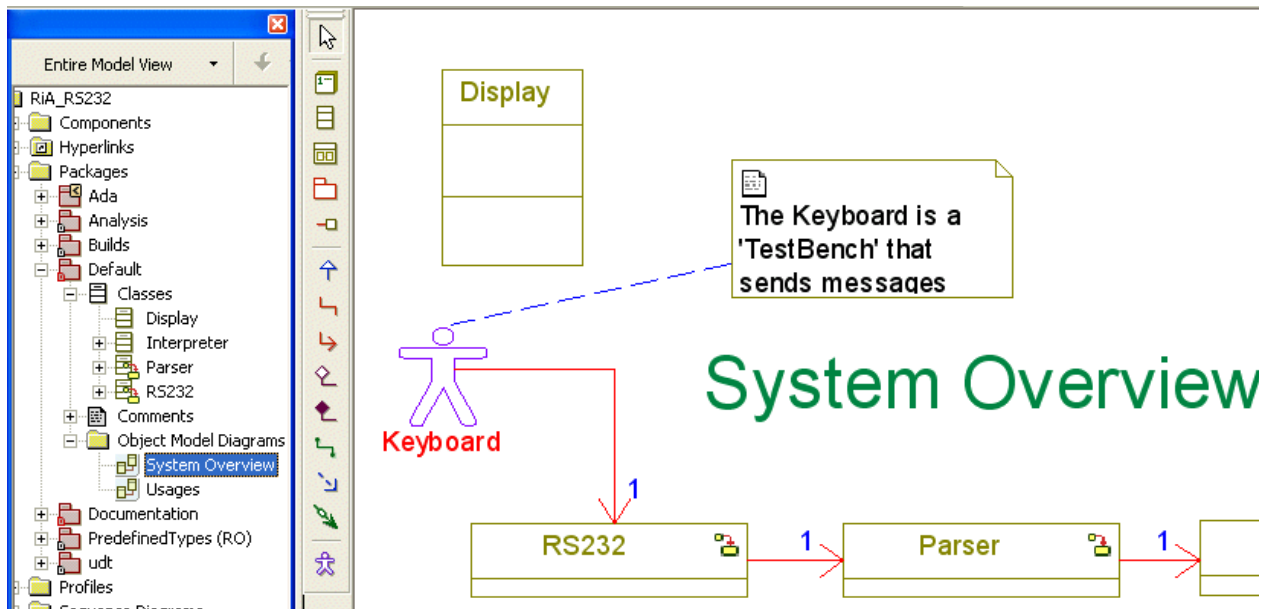
1. In the browser, expand `Packages`.
2. Right-click the `Default` package and select **Add New > Class** from the pop-up menu. Rhapsody creates a new class named `class_n`, where n is greater than or equal to 0. The new class is located in the browser under the `Classes` category.
3. Rename `class_n` to `Display` by typing over the name and then press `Enter`. The following example shows the `Display` class in the browser.



Creating an Object Model Diagram

Object model diagrams (OMD) define the static structure of the system including the software classes and their relationships. In this example, you add your new Display class to the System Overview model diagram with these steps:

1. In the browser, expand the Object Model Diagrams category.
2. Select the SystemOverview from the list, as shown in the left portion of the following figure.
3. Drag the new Display class from the browser onto the SystemOverview diagram. Your diagram should be similar to the right portion of the following figure.




In the OMD, when you move the cursor over the `Display` class border, the box becomes highlighted; when you click on the class name, the name is highlighted. This lets you select and modify either the class or its name.

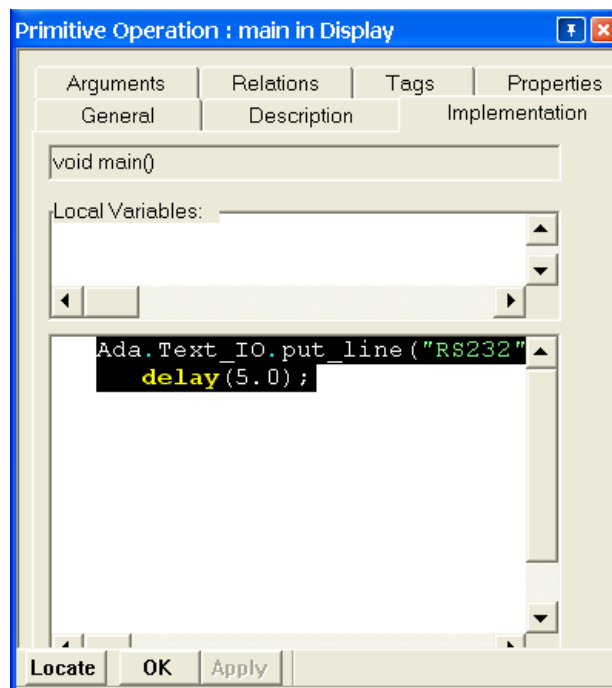
Adding an Operation

The `Display` class needs attributes or operations to specify behavior and allow you to generate, compile, and link code for the `Display` class.

To add an operation to the class, follow these steps:

1. In the browser, right-click the `Display` class, then select **Add New > Operation** from the pop-up menu. The new operation is displayed in the browser.
2. Rename the operation `main`. This operation will serve as the entry point procedure in the Ada executable.
3. Double-click the `main` operation or click the Invoke Features Dialog icon  for the new operation.
4. Select the **Implementation** tab, and type the following code in the text area. The Implementation tab should be similar to the example below.

```
Ada.Text_IO.put_line("RS232");  
delay(5.0);
```

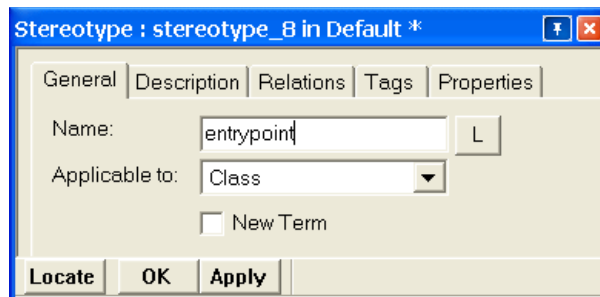


5. Click **OK** to apply the changes and dismiss the dialog box.

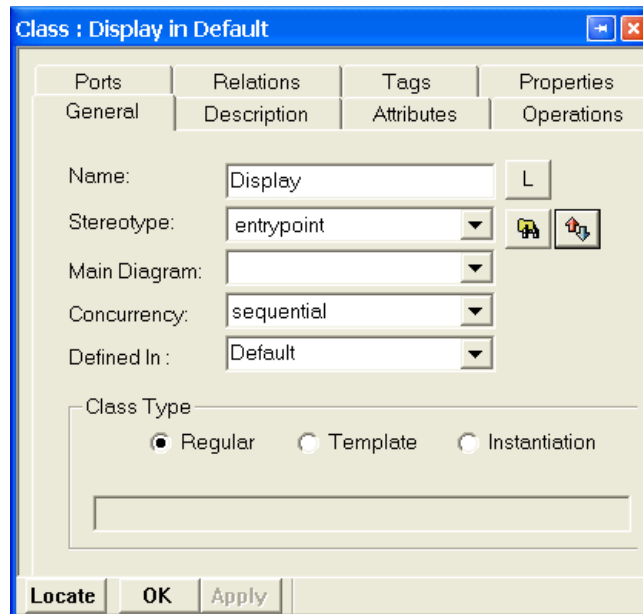
Setting an Entry Point

You can use the «entrypoint» stereotype to direct the code generator to make the main Ada procedure the application entry point. A *stereotype* is a type of modeling element that extends the semantics of the UML metamodel by typing UML entities. To set an entry point, follow these steps:

1. In the browser, right-click the `Display` class and then select **Features** from the pop-up menu. The Features dialog box opens.
2. In the **General** tab, select <<New>> from the **Stereotype** pull-down list and enter `entrypoint` into the dialog box (as shown below).



3. Click **OK** to add this stereotype to the model. The Features dialog box shows the changes, as shown in the following figure. If a stereotype name is long, you may stretch the size of this dialog box to display long names fully.



4. Click **OK**. Note that the diagram changes to display <<entrypoint>> above Display in the class diagram.

Checking the Configuration

The code generation component requires a configuration that provides information on the scope, initial instances, code checks, and settings for the build.

Check the configuration in the sample using these steps:

1. In the browser, expand the `Test` component and the `Configurations` category. You will see the three configurations.
2. Highlight the `GnatRelease` configuration.
3. Right-click and select **Set as Active Configuration** from the pop-up menu.

This ensures that the `GnatRelease` configuration is used for code generation. You can also select the active configuration from the **Code** toolbar.

You have created the entire model and are now ready to generate code for the `RS232` application.

Generating Code

Rhapsody generates implementation code from your UML model. To generate code, follow these steps:

1. Select **Code > Generate > GnatRelease**.
2. The subdirectory `GnatRelease`, which Rhapsody needs to store the generated files, does not yet exist. Rhapsody asks you to confirm its creation. Click **Yes**.


Rhapsody generates the code and displays output messages in the **Build** tab of the Output window.

The messages inform you of the code generation status, including:

- ◆ Success or failure of internal checks for the correctness and completeness of your model. These checks are performed before code generation begins.
- ◆ Names of files generated for classes and packages in the configuration.
- ◆ Names of files into which the `main()` function is generated.
- ◆ Location of the generated make file.
- ◆ Completion of code generation.

Building the Application

Once you generate code, you are ready to build the application using either of the following methods:

- ◆ Select **Code > Build Test.exe**.
- ◆ Click the **Make** icon  in the toolbar above the work area.

Rhapsody invokes the `make` command for the appropriate compiler and displays compiler messages in the Build tab of the Output window. As you can see from the output, there is a build error.

Analyzing a Build Error

If you receive errors, double-click the error message in the Output window to go to the error source. The source of the error appears as a highlighted element. Rhapsody navigates to the problem area in the `display.adb` file.

By default, line numbers are not shown in the generated code. To display line numbering in the source editor, follow these steps:

1. Right-click in the source window and select **Properties** from the pop-up menu.
2. In the **Misc** tab, select the numbering style.
3. Click **OK** to apply the changes and dismiss the dialog box.

Adding Predefined Packages

When you examine the source, you can see that the file is missing the appropriate Ada context clause. To build correctly, the file needs the line `with Ada.Text_IO` at the beginning of the file. You can use a UML graphical notation to express the context clause to generate the proper code.

Adding the Context Clause to the Model

To add the context clause to the model, follow these steps:

1. Select **File > Add to Model**. The Add To Model dialog box opens.
2. Navigate to `<Rhapsody>\Share\LangAda83\model\RIAServices_rpy`.
3. Show all the file types, then select `ADA.sbs`.
4. Accept the default settings and click **Open**. The ADA package is added to the browser.


Adding a Class

To add a class, follow these steps:

1. Right-click the ADA package, then select **Add New > Class** from the pop-up menu.
2. Name the new class `Text_IO`.
3. Drag-and-drop the `Text_IO` class to the `SystemOverview` OMD and place it to the right of the `Display` class.

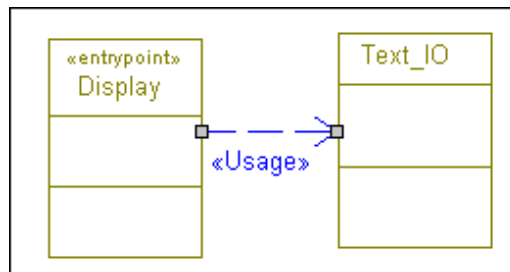
Drawing the Dependency and Setting the Stereotype

To draw a dependency and set its stereotype, follow these steps:

1. Click the **Dependency** icon  in the **Drawing** toolbar.
2. Draw a dependency from the `Display` class to the `Text_IO` class.

A *dependency* is a direct relationship in which the function of an element requires the presence of and may change another element.

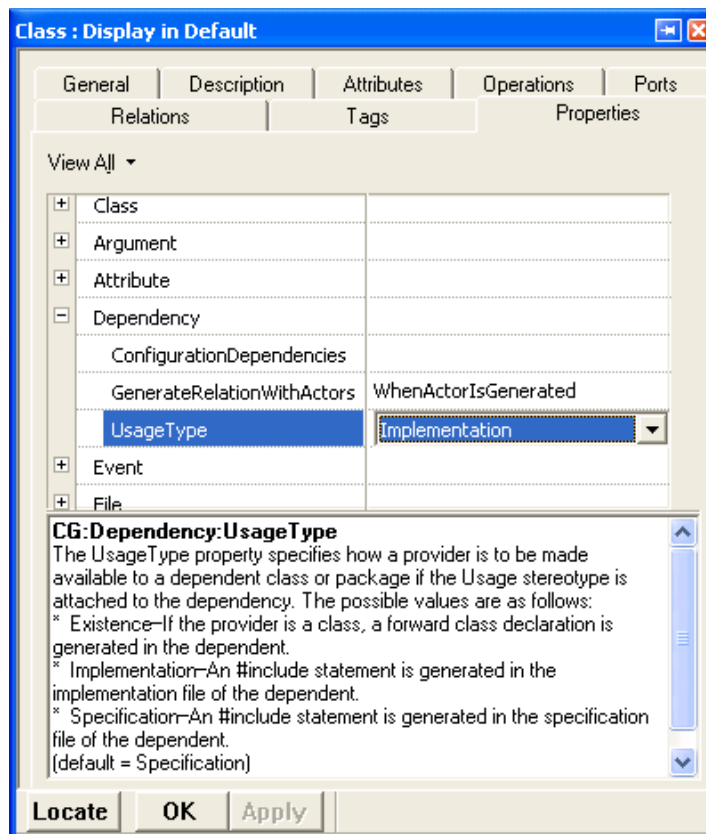
3. Right-click the dependency line, then select **Features** from the pop-up menu. The Features dialog box opens.
4. Select `«Usage»` from the **Stereotype** pull-down list.
5. Click **OK** to apply the changes and dismiss the dialog box. The `Display` portion of the `SystemOverview` diagram should be similar to this example.



Adjusting the Package Dependency

Rhapsody enables you to specify whether the dependency should generate the context clause in the source specification or the implementation body. Because the `Display` class has only a body, you must set the context clause generation to implementation body. To set the context clause generation, follow these steps:


1. Right-click the `Display` class and select **Features** from the pop-up menu. The Features dialog box opens. Select the **Properties** tab.
2. From the **View** filter, select **All**.
3. Browse to the property `CG: :Dependency: :UsageType`, click the value field and select `Implementation` from the pull-down list. Note that the definition of the highlighted property is displayed at the bottom of the window.




4. Click **OK** to apply the changes and dismiss the dialog box.
5. Regenerate your code and verify that the proper context clause generates.
6. Rebuild the code and verify that it builds without errors.

Running the Application


You are ready to run the application using either of the following methods:

- ◆ Select **Code > Run Test.exe**
- ◆ Click the **Run executable** icon .

A console window displays the program output. This application stops after a five second delay. You can also end the application using one of the following methods:

- ◆ Close the application window.
- ◆ Select **Code > Stop**.
- ◆ Click the **Stop execution** icon  in the **Code** toolbar.

Instead of incrementally compiling, building, and running your application in three separate steps, you can perform all three tasks in one step, using either of the following methods:

- ◆ Select **Code > Generate/Make/Run**.
- ◆ Click the **GMR** icon  in the toolbar.

Examining the Code

The `RS232` directory contains the generated code for this example. You can examine the generated code using either an internal or external editor.

Using the Internal Editor

To view generated source files in the browser using the Rhapsody internal editor, do one of the following:

- ◆ Select the package or class whose code you want to view, then select **Code > Edit > Selected classes**.
- ◆ Right-click the `Default` package, and select **Edit Code** from the pop-up menu.
- ◆ Right-click the `Display` class, and select **Edit Code** from the pop-up menu.

Selecting an External Editor

To view long text fields, such as operation bodies, or to view the generated code, Rhapsody calls an external editor.

To specify the external editor, follow these steps:

1. Select **File > Project Properties**. The Features dialog box opens. Select the **Properties** tab.
2. From the View filter, select **All**.
3. Browse to the `General` subject and `Model` metaclass.
4. Click the ellipsis (...) button in the `EditorCommandLine` value field to browse to your preferred editor, such as `C:\Windows\System32\notepad.exe`.
5. Browse to the property `General::Model::ClassCodeEditor` and set it to the value `CommandLine`.
6. Click **OK** to apply the changes and dismiss the dialog box.

When you edit code, Rhapsody opens it in the specified external editor.

Closing the Project

To close the project, follow these steps:

1. Select **File > Close**. Rhapsody asks whether you want to save changes to the project.
2. Click **Yes**. Rhapsody saves and closes the project, without exiting.
3. To exit Rhapsody, select **File > Exit**.

Quick Start with Java

To gain a more in depth understanding of using Rhapsody in Java to produce models, use the *Rhapsody in Java Tutorial* to create a dishwasher model from beginning to end. To access that tutorial, select the Rhapsody **Help** menu and select **List of Books**. Click the **Rhapsody in Java Tutorial** item in the list to launch a PDF version of the manual.

This section uses an official Rhapsody sample HomeAlarm project to demonstrate using Rhapsody in Java.

Opening the HomeAlarm Sample Project

Follow these steps to locate and open the Java HomeAlarm sample:

1. Launch Rhapsody as described in [Starting Rhapsody](#).
2. From the main menu, select **File > Open**. In the dialog box, In the dialog box, navigate to the `Samples\JavaSamples\HomeAlarm` directory in the Rhapsody directories. Select the `HomeAlarm.rpy` file. The project opens in Rhapsody.

Saving the Sample Java Project

In order to avoid changing the official Rhapsody in Java home alarm project, save the opened project in a different directory. To save a copy of the official sample, follow these steps:


1. Select **Save As** from the **File** main menu.
2. Use the pull-down menu at the top of the dialog box to select a different directory, such as `C:\Temp`.
3. Click **Save** to put the copy in the selected directory.

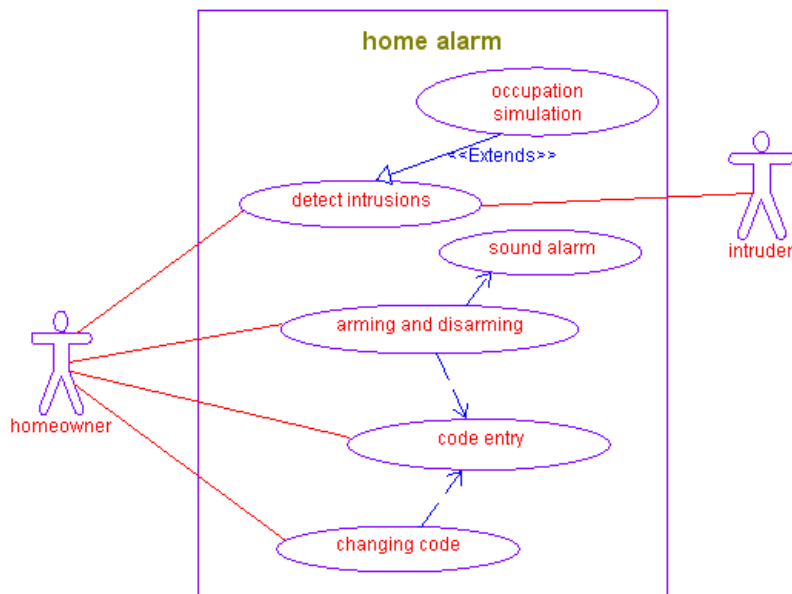
Viewing Diagrams

This section describes how to explore the diagrams of the Home Alarm model.

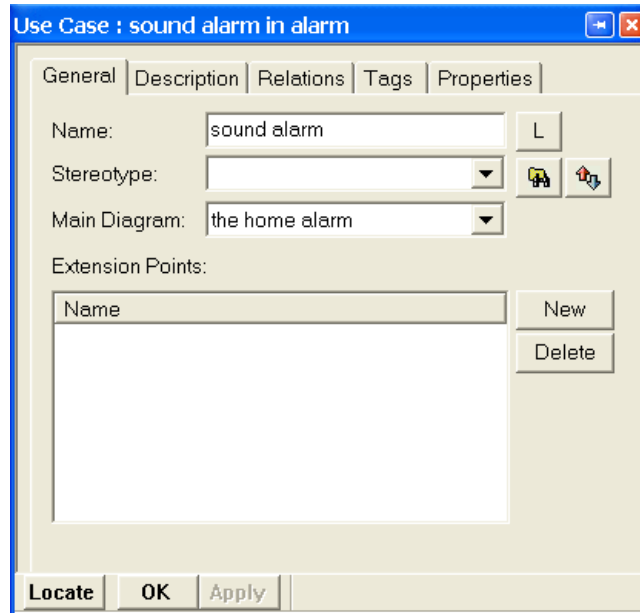
Use Case Diagrams

A *use case diagram* (UCD) captures the functional requirements and external actors for a system. The diagrams display in the drawing area. To view a UCD, follow these steps:

1. In the browser, expand the **Use Case Diagrams** category.
2. Double-click the home alarm UCD. The UCD opens in the drawing area.
3. Click the **Zoom to Fit** icon . The diagram resizes to fit in the drawing area, as shown in the following figure.



4. Double-click the sound alarm use case to view its features. The Features dialog box opens, as shown in the following figure.



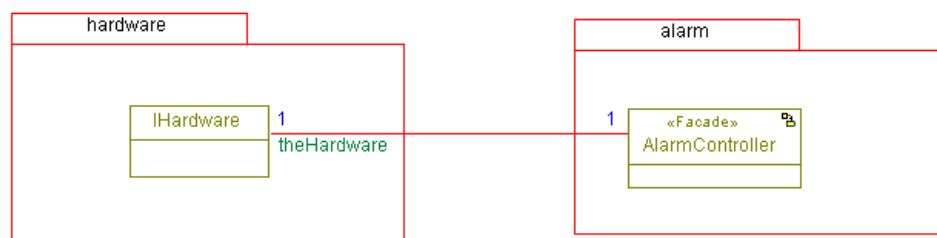
5. Click the **Description** tab to view the definition of this use case.
6. Click **OK** to close the dialog box.

Object Model Diagrams

An *object model diagram* (OMD) is a logical view that shows the static structure of the classes and instances in an object-oriented software system and the relationships between them.

To open the OMD, follow these steps:

1. In the browser, expand the Object Model Diagrams category.
2. Double-click the domain diagram OMD. The domain diagram OMD opens in the drawing area, as shown in the following figure.



The OMD has two packages:

- ◆ hardware—Contains IHardware, a hardware interface class that describes the operations that can be called on the hardware.

The hardware package has its own OMD, Possible hardware implementations. To view this OMD, expand the hardware category and the Object Model Diagram category in the browser, then double-click the OMD.

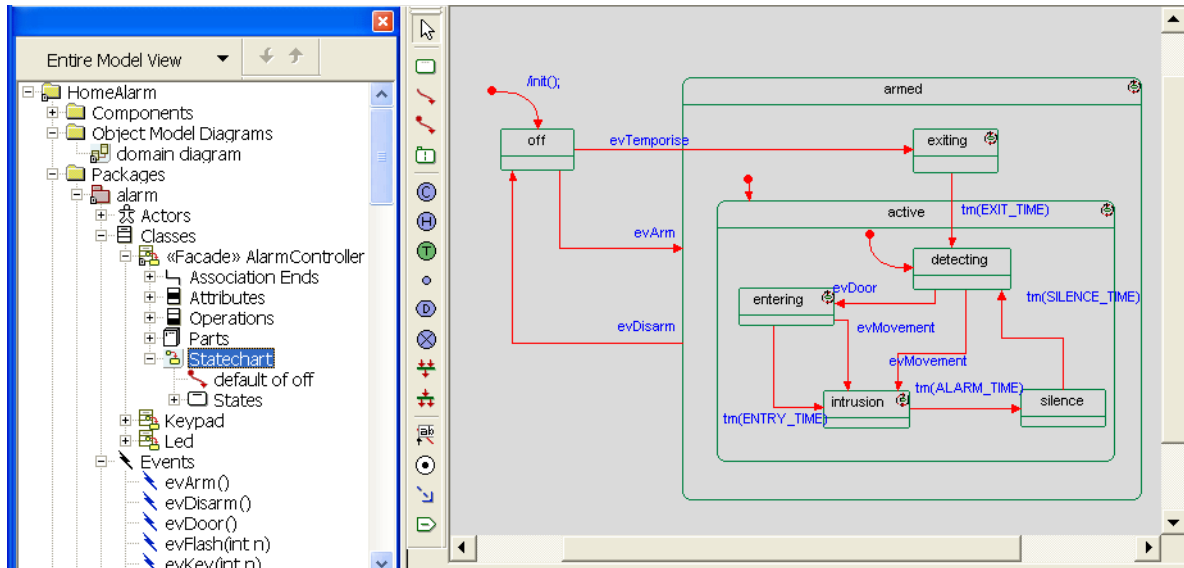
- ◆ alarm—Contains AlarmController, a class that controls the alarm system. The alarm package has its own OMD, home alarm overview.

The alarm package is related to hardware through an association. This enables alarm to point to one individual instance of hardware as its hardware (theHardware).

Statecharts

A *statechart* defines the behavior of individual classes in the system.

To open the statechart for the alarm controller, right-click the <<Facade>> AlarmController class in the browser, then double-click the Statechart listed below. The statechart opens in the drawing area, as shown in the following figure.



A statechart has the following elements:


- ◆ **States**—Each state contains a label and specified code to execute on entry (actions on entry) and exit (actions on exit) from the state. Also included are reactions in state, which are specified like transitions. A state is shown as a green box in the statechart (for example, the detecting state).
- ◆ **Transitions**—Transitions enable the model to go from state to state during execution. Each transition has a trigger, an optional guard, and optional code to execute. When the state receives an event of type equal to the trigger specified, the transition takes place. Transitions are shown as red arrows in the statechart. A transition's trigger, guard, and action code are shown adjacent to the transition in blue text in the following format:

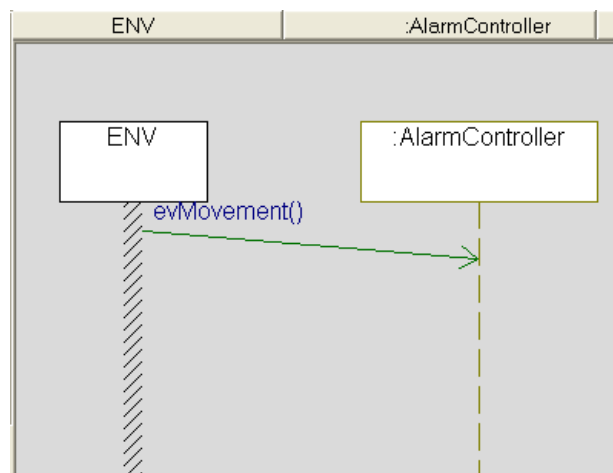
```
trigger [guard] /code
```
- ◆ **Default connector**—A default connector indicates the initial state and is shown as a red arrow with a point on the end.

You can double-click the states or transitions to examine the details for each in the Features dialog box.

Creating an Animated Sequence Diagram

An *animated sequence diagram* shows the messages passed between instances as the application runs. To create a simple animated sequence diagram, follow these steps:

1. Select **Tools > Animated Sequence Diagram**. The Open Sequence Diagram box opens.
2. Under the `alarm` package, select **Presence Simulation**, then click **Open**. The diagram opens.
3. Click the **Event Generator** icon  in the **Animation** toolbar to open the Events dialog box.
4. Click **Select**. The Instances Selection dialog box opens.
5. Select the instance of the **AlarmController** class. Click **OK** to close the Select Instances dialog box and return to the Events dialog box.
6. Select **evMovement** from the **Event** pull-down. Click **OK** to simulate an intruder's movement.



The animated sequence diagram displays the collaboration between the system border and the `AlarmController` instance in the form of an event.

Animating the HomeAlarm Model

You can view and debug the behavior of the home alarm by animating the model as follows:


1. Close the statechart and OMD.
2. In the browser, expand the `alarm` package and `Classes` category.
3. Select **Code > Generate/Make/Run**. A message may display asking if you want to create the animated model if it has not been created previously.

The **Animation** toolbar opens so that you can control the model animation.

In the browser, the `Instances` category contains the instances created as your application runs in animation mode.

Running the Animated Model

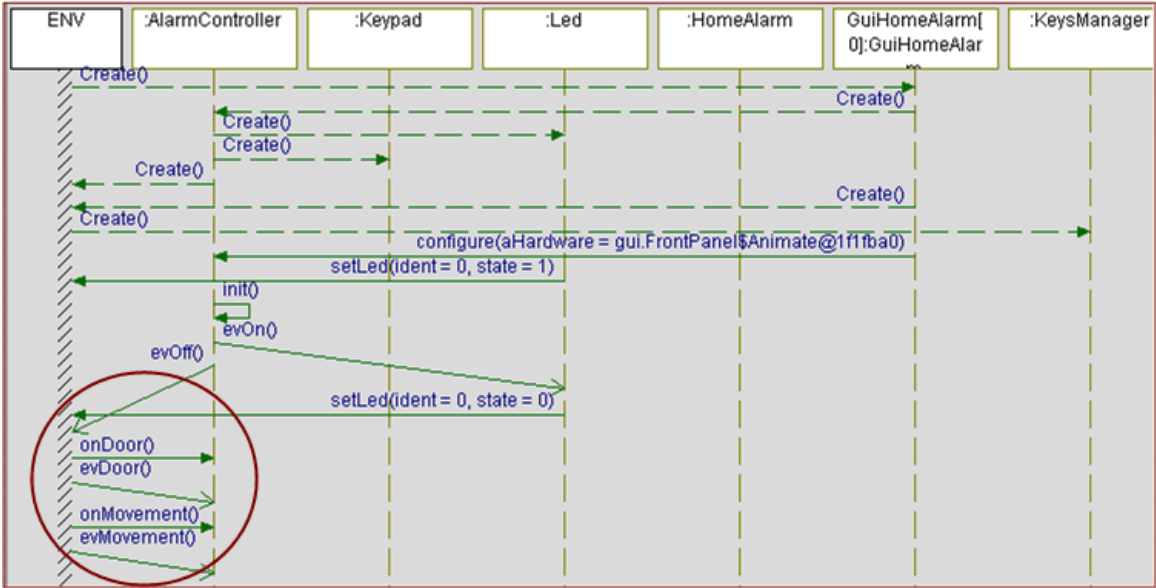
To continue running animation, follow these steps:

1. Click the **Go** icon  in the **Animation** toolbar to begin executing the home alarm program. The alarm keypad opens, as shown in the following example.




2. Click the **Door** button on the GUI image and note the addition in the sequence diagram. This simulation is shown in the sequence diagram.

3. Click the **Move** button on the GUI image and in the animated sequence diagram, note the additions for the door and movement (circled as shown in the following figure).



Stopping Animation and Saving the Diagram

To stop animation and save the animated diagram, follow these steps:

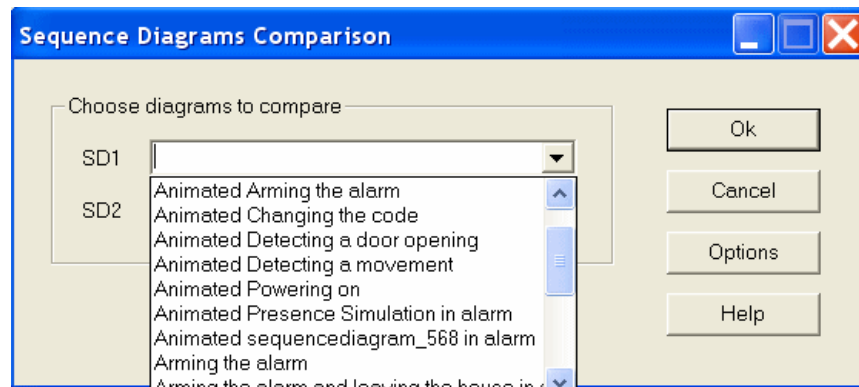
1. Select **Code > Stop Execution** or click .
2. The system displays a dialog box asking if you want to save the animated sequence diagram with the name that it will use. Click **Yes** to save the diagram.

Saving the animated diagram is useful to compare the results of the current session to those of different execution scenarios.

Comparing Animated Sequence Diagrams

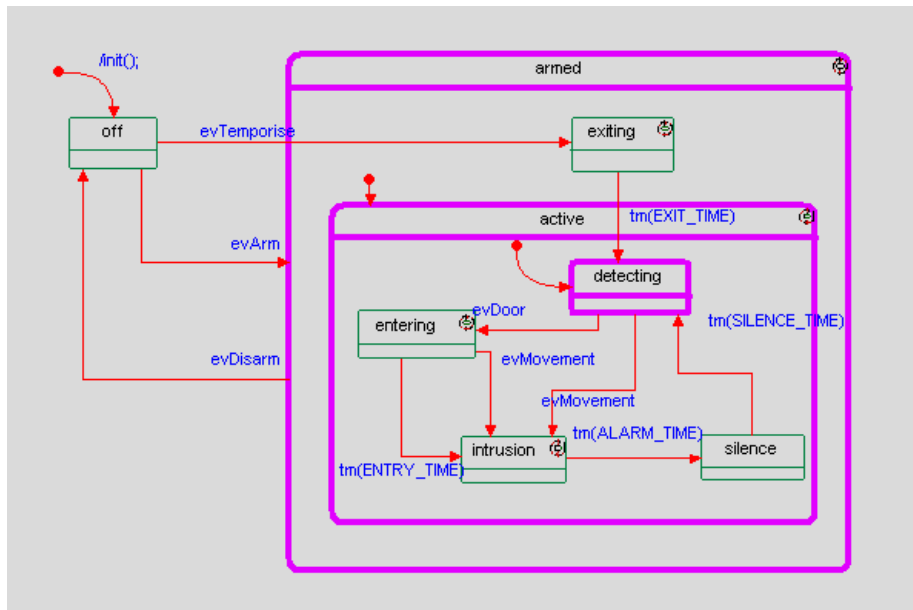
To compare the saved sequence diagrams from different animation sessions, follow these steps:

1. Select **Tools > Sequence Diagram Compare**.
2. From the pull-down menus in the Sequence Diagrams Comparison dialog box (as shown in the following figure), select the names of the diagrams you want to compare and click **OK**.



Animating a Statechart

Next, create an animated statechart by right-clicking the `AlarmController` instance in the browser and selecting **Open Instance Statechart** from the pop-up menu. Rhapsody displays an animated statechart in the drawing area, as shown in the following figure.




Note

The `armed`, `active`, and `detecting` states are highlighted in magenta; inactive `off` state is shown in green.

Generating Events

You can debug the animated statechart using events. In this example, you will generate the `evArm` event to transition the model from the off state to the armed state.

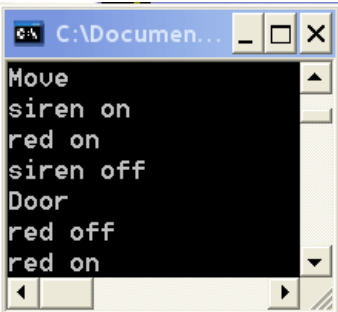
To send an `evArm` event to the animated statechart, follow these steps:

1. Click the **Event Generator** icon  in the toolbar. The Events dialog box opens.
2. Click **Select**. The Instances Selection dialog box opens.
3. Select the instance of the AlarmController class, then click **OK** to close the Instances Selection dialog box and return to the Events dialog box.
4. Select `evArm` from the **Event** pull-down list, then click **OK**. In the GUI, the LED displays both red and green lights, as shown in the following figure.

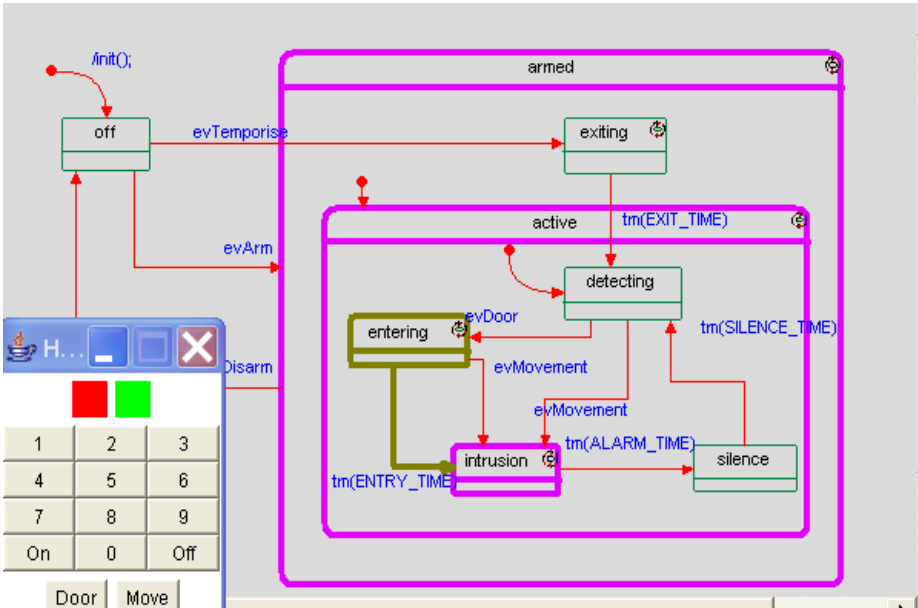


GUI Testing

Click the buttons and watch the LED display blink. This displays the commands being sent from the GUI to the system for testing. These commands are displayed in a separate scrolling window, as in this example.



To test the actions in the statechart, the activated states are highlighted and move through the stages when the alarm is tripped, as shown in the following figure.



Rhapsody for Systems Engineers

Rhapsody allows systems engineers to capture and analyze *requirements* quickly and then design and validate system behaviors. In addition, Rhapsody produces high-quality systems engineering specification documents using report templates. See the [ReporterPLUS](#) for more information about accessing templates and producing reports.

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

- ◆ Examine the UML Forum's [UML specification](#)
- ◆ Examine the OMG's [SysML specification](#)

Refer to the Systems Engineering section of the *Rhapsody User Guide* for more information about the systems engineering features and diagrams.

Rhapsody for Systems Engineers

Systems engineers may use Rhapsody's *SysML* profile or the *Harmony process* and *Harmony profile* and tools to guide software development through this iterative development process:

- ◆ Perform system analysis to define and validate system requirements
- ◆ Design and specify the system architecture
- ◆ Systems analysis and design
- ◆ Software analysis and design
- ◆ Software implementation
- ◆ Validate and simulate the model to perform detailed system testing

Rhapsody's system engineering features allow system designers to hand off their work to software developers accurately and easily.

Note

For more information about the Harmony process, refer to the *Harmony Deskbook* by Hans-Peter Hoffmann, Ph.D., 2008 (available on the corporate Web site).

Diagrams for Systems Engineering

A systems engineering 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 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

These features are only accessible for projects using the *Harmony profile*.

More Systems Engineering Information

To learn more about using Rhapsody for systems engineering, refer to the following:

- ◆ *Systems Engineering Tutorial*
- ◆ SysML eLearning course (purchased separately)
- ◆ Instructor-led training courses (purchased separately)

The Systems Engineering Tutorial provides step-by-step instructions demonstrating the tasks that systems engineers can accomplish using Rhapsody. To access this tutorial, in Rhapsody's **Help** menu select **List of Books** and click the **Systems Engineering Tutorial** item.

Rhapsody for DoDAF and MODAF Development

Rhapsody supplies DoDAF and MODAF add-ons allowing engineers to create a compliant architecture model for their national standard. The add-ons, a template driven solution, can be customized and extended to meet specific customer requirements.

Refer to the *Rhapsody User Guide* for more detailed information.

MODAF Viewpoints

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

In addition to the MODAF variations of the standard DoDAF views, Rhapsody's 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 Viewpoints

Rhapsody's DoDAF package includes 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-1 documents 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 <u>parts</u> (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 <u>anchors</u>) 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).

Reports

Rhapsody offers two ways to generate reports from the models, charts, generated code, and other items:

- ◆ A simple and quick internal RTF report generator
- ◆ A more powerful reporting tool, Rhapsody ReporterPLUS

ReporterPLUS

ReporterPLUS produces reports that are suitable for formal presentations and can be output in any of these formats:

- ◆ HTML page
- ◆ *Microsoft Word*
- ◆ *Microsoft PowerPoint*
- ◆ rtf
- ◆ text

You can save the file and view it in any program that can read the report's format. In addition, you can create custom report specifications that define the structure, content, and format of reports.

The following stylistic definitions all control a report's appearance:

- ◆ ReporterPLUS template and selected options
- ◆ Output type (Word, PowerPoint, HTML, RTF, text)
- ◆ Word or PowerPoint template
- ◆ HTML style sheet
- ◆ HTML tags in your model, ReporterPLUS template, or in an inserted file

ReporterPLUS has extensive Help information available from the interface. Use the **Help Topics** to answer your more detailed questions about the features of this specialized interface.

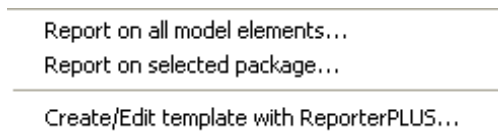
Starting ReporterPLUS

To start ReporterPLUS, you may use one of these two methods from within Rhapsody:



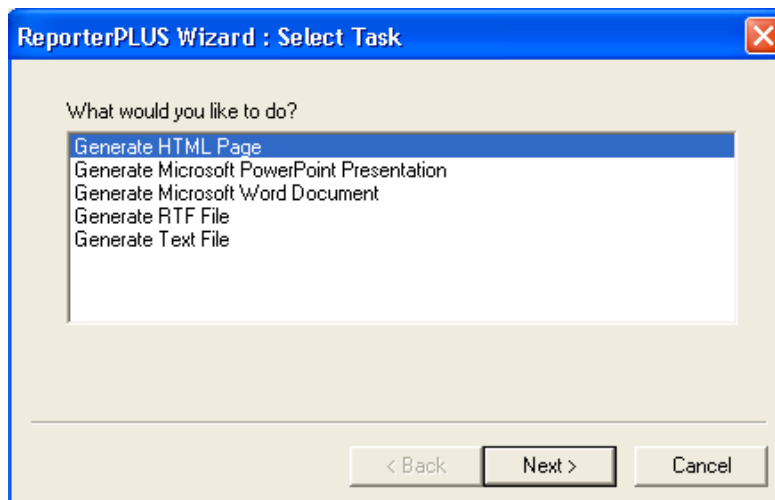
- ◆ Click the ReporterPLUS **ReporterPLUS** icon above the Welcome screen.
- ◆ Select **Tools > ReporterPLUS**.

This menu displays options for printing the model currently displayed in Rhapsody.



The **Report on selected package** option is not available from this menu unless a package in the model is highlighted in the Rhapsody browser. If one of the first two items is selected, a report can be generated using a predefined template without displaying the main ReporterPLUS GUI. If the last option is selected, ReporterPLUS starts with no model elements imported.

After starting ReporterPLUS, select one of these report generation option.



Examining and Customizing ReporterPLUS Templates

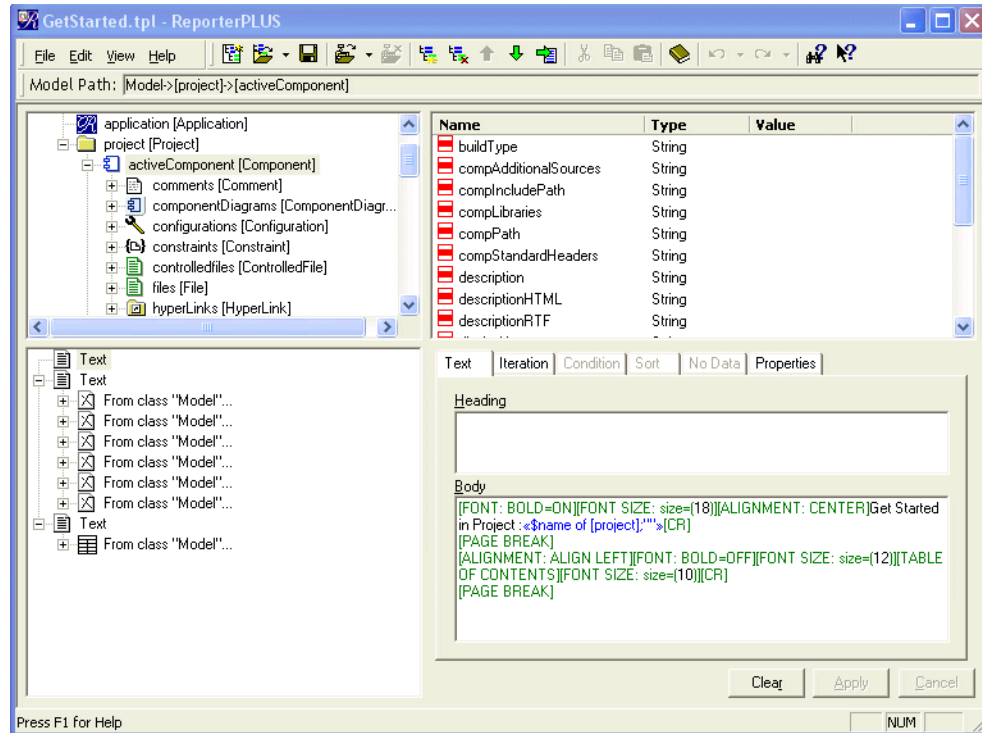
Rhapsody includes numerous pre-fabricated report templates that you may want to use as they are or customize to meet your needs.

Rhapsody models can be loaded into the ReporterPLUS interface and used to create generic or model-specific templates. This interface allows you to create and modify templates graphically using a drag-and-drop method.

To access ReporterPLUS and the templates, follow these steps:

1. Select **Tools > ReporterPLUS** from the menu.
2. From the next menu, select **Create/Edit template with ReporterPLUS**.
3. Your model displays in the upper left corner of the ReporterPLUS interface. Click the items in the tree to expand and examine the model. When you click part of the model a description of that item appears to the right of the model tree.
4. Select **File > Open Template**.
5. Select any of the [Standard Templates](#) from the dialog box. The template structure appears in the lower left area.

- Click on individual items in the template and the definition of the item appears to the right, as shown in the following figure.



- You may also want to add standard headings and text to an existing template. To add this “boilerplate” material, highlight a section of the template in the lower left window and click the **Text** tab in the lower right window. Type text in the Heading and Body sections as desired.

Note

For more complex changes, study the commands, creation of subtemplates, and the uses of the Q Language, as described in the *Rhapsody ReporterPLUS Guide*.

Generating Reports

If you are going to generate a report in *Microsoft Word*, be certain that *Word* is closed before you start this procedure to avoid a conflict between *ReporterPLUS* and *Word*.

Once you have a template that you want to use for a report, follow these steps:

1. With your project open in Rhapsody, select **Tools > ReporterPLUS** from the menu.
2. From the next menu, select one of these two options:
 - ◆ Report on all model elements
 - ◆ Report on selected elements
3. Rhapsody displays the ReporterPLUS Wizard that allows you to select the desired output format
4. In subsequent dialog boxes, select the template and directory location and name for the finished report.

You may wish to use one of the standard templates (listed in the next section) without modification to gather basic information about the model.

Standard Templates

ReporterPlus provides a large number of standard templates. These templates can be used as they are or modified to meet your needs. The following table lists some of the commonly used templates with a description of each and the preferred output formats. The rich text format (.rtf) and plain text (.txt) format are not listed in the following table because the design capabilities of those output formats is simplified and can be used for any of these templates. However, *HTML*, *Word*, and *PowerPoint* provide more formatting capabilities, so some of the generic templates look better in one or more of the other three formats.

Template Name	Preferred Output Formats	Description
class.tpl	HTML or Word	This template defines the information about the classes in the project. It prints the documentation of the following elements of the class: <ul style="list-style-type: none"> • Attributes • Operations • Relations • Events • Statechart and Activity Diagrams (including the diagrams)
FullDetailedProjectReport.tpl	Word, PowerPoint, or HTML	This generic template defines the information about the complete project. It includes all the model elements of the project and all the diagrams of the project.
GetStarted.tpl	Word & PowerPoint	Shows all diagrams and classes in a model.
ModelMetrics.tpl	Word or HTML	This prints out metrics for the entire model and each package in the model
PackageReport.tpl	Word	This template defines the information related to the packages and all the elements of in these packages: <ul style="list-style-type: none"> • All packages (nested structure) • All the diagrams in each package • Elements contained in the packages • Sub Packages and their elements It does not include project level information.
ProjectReport.tpl	Word	This generic template defines the information about the complete project. It includes all the model elements and all the diagrams in the project. It includes and title page and table of contents.
RequirementsTable.tpl	Word	This template lists the requirements, use cases, actors, and all diagrams.

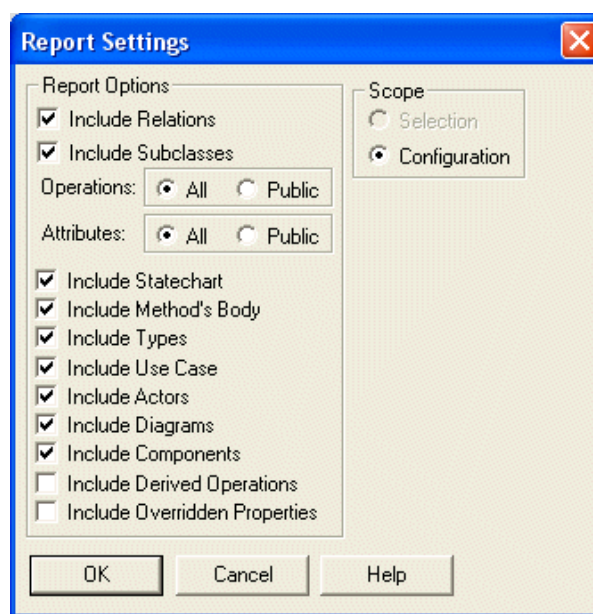
Template Name	Preferred Output Formats	Description
Rhapsody HTML Exporter.tpl	HTML only	The template generates a comprehensive HTML report of the project. The template defines the information about the complete project including model elements by label.
SysMLreport.tpl	Word or HTML	<p>This template uses the SysML profile to provide the underlying stereotypes to generate a document. If SysML was not selected as the Project Type when it was created, you cannot take full advantage of the features of this template.</p> <p>The main sections the document produces are in the following order (if they exist):</p> <ul style="list-style-type: none">• Requirements Diagrams• Use case diagrams• Sequence Diagrams• Structure Diagrams• Object Model Diagrams• External Block Diagrams• Internal Block Diagrams• Parametric Diagrams• Element Dictionary• Model Configuration <p>The document produced is hyperlinked where appropriate.</p>
UseCaseReport.tpl	Word	This template prints all of the use cases and use case diagrams in a model.

Using the Internal Reporting Facility

The internal reporting facility is particularly useful for quick print-outs that the developer needs to use for debugging the model. The reports are not formatted for formal presentations.

Producing an Internal Report

To create a report using the simple, internal reporter, select **Tools > Report on model**. The Report Settings dialog box opens, as shown in the following figure.



The dialog box contains the following fields:

- ◆ **Report Options**—Specifies which elements to include in the report. The possible values are as follows:
 - **Include Relations**—Include all relationships (associations, aggregations, and compositions). By default, this option is checked.
 - **Include Subclasses**—List the subclasses for each class in the report. By default, this option is checked.
- ◆ **Scope**—Specifies the scope of the report. The possible values are as follows:
 - **Selection**—Include information only for the selected elements.
 - **Configuration**—Include information for all elements in the active component scope. This is the default value.

- ◆ **Operations**—Specifies which operations to include in the report. The possible values are as follows:
 - **All**—Include all operations. This is the default value.
 - **Public**—Include only the public operations.
- ◆ **Attributes**—Specifies which attributes to include in the report. The possible values are as follows:
 - **All**—Include all attributes. This is the default value.
 - **Public**—Include only the public attributes.

Select the portions of the model you want to be included in the report, then click **OK** to generate it. The report is displayed in the drawing area with the current file name in the title bar.

Using the Internal Report Output

When you generate a report in Rhapsody using **Tools > Report on model**, the initial result uses the internal RTF viewer. To facilitate the developer's research, this output may be used in the following ways:

- ◆ To locate specific items in the report online, select **Edit > Find** from the menu and type in the search criteria.
- ◆ To print the initially generated report, select the **File > Print** menu option.
- ◆ The initially generated report is only a view of the RTF file that the facility created. This file is located in the project directory (parallel to the `.rpy` file) and is named `RhapsodyRep<num>.rtf`. If you wish, open the RTF file using a word processor that handles RTF format, such as Microsoft Word.

Technical Support and Documentation

Rhapsody software continues to be supported by the Telelogic support group for customers who licensed Rhapsody before November 1, 2008. If you are one of these “heritage” customers, you may use the familiar support procedures described in [Contacting Telelogic Rhapsody Support](#).

After November 1, 2008, all Telelogic Rhapsody customers also have access to the IBM technical support team and resources, as described in [Contacting IBM Rational Software Support](#).

Contacting Telelogic Rhapsody Support

The Telelogic Rhapsody technical support group assists heritage Rhapsody customers through the online problem report form, automatically generated problem reports, direct contact by [Calling Telelogic Rhapsody Technical Support](#) or contacting them through IBM’s Support site, as described in [Contacting IBM Rational Software Support](#).

Note

Assistance from the technical support staff for purchased products is only available to companies that have paid for ongoing maintenance.

Accessing the Automated Problem Report Form

Follow these steps to send the automated problem report to the technical support staff:

1. In Rhapsody click **Help** on the menu bar.
2. Select the **Generate Support Request** option from the menu.
3. The following form appears with some of your product information filled in.

Generate Support Request

Problem Details
Please complete below with as much detail as possible to describe your issue:

Impact: My work is not affected. I want an answer to a question.

Summary:

Problem:

Rhapsody Information

Version: 7.3
Build: 987717
Serial No: T10-293399
Language: C++
Edition: Development Edition

Rhapsody Window Snapshot...

System Information

Version: Windows XP
Service pack: Service Pack 2
Build: 2600

Screen Snapshot...

Attachment Information (Double-click Item to View)

Add Video Capture...
Add System Details
Add Product Files
Add File(s)...
Remove...
Remove All...

Item Description:

Help Just Text (No Email)...
Cancel Preview and Send

4. Check the product information to verify it is accurate.
5. From the **Impact** drop-down list box, select the severity of the problem.
6. In the **Summary** box, summarize the problem.

7. In the **Problem** box, type a detailed description of the problem.
8. If available, attach a snapshot. Click the **Rhapsody Window Snapshot** or **Screen Snapshot** button, whichever is applicable, and select the snapshot wherever you have it on your machine.
9. If possible, add the model, active component, files, and/or a video capture by using the buttons in the **Attachment Information** area.
10. Add any additional items or information to help the Technical Support staff resolve the problem.
11. Click **Preview and Send** to submit the report.

The problem report is recorded in the Rhapsody case tracking system and put into a queue to be assigned to a support representative. This representative works with you to be certain that your problem is solved.

Automatically Generated Problem Reports

If your Rhapsody system crashes, it displays a message asking if you want to send a problem report to Rhapsody technical support about this crash.

If you select to send the report, the system displays the same online form that is available from **Help > Generate Support Request**. However, this form contains information about the crash condition in addition the information that is usually filled in describing your system.

Add any more information that you can to help the support staff identify the problem and then click **Preview and Send** to submit the report.

Calling Telelogic Rhapsody Technical Support

If your company has a current Rhapsody maintenance agreement purchased before November 1, 2008, you may call the Telelogic Technical Support staff directly using the appropriate telephone number for your region listed below.

Support Location	Telephone Number	Availability
United States	(800) 577-8449	8:30 a.m. to 8:00 p.m. EST
Europe	00800 57784499	8:00 a.m. to 5:00 p.m. GMT
Europe (alternative number)	+353 1 2090154	8:00 a.m. to 5:00 p.m. GMT

Contacting IBM Rational Software Support

The IBM Rational Software Support team provides the following resources for your assistance:

- ◆ For contact information and guidelines or reference materials that you need for support, read the [IBM Software Support Handbook](#).
- ◆ For FAQs, lists of known problems and fixes, documentation, and other support information, visit the [Telelogic Rhapsody Support Site](#).
- ◆ Voice support is available to all current contract holders by dialing a telephone number in your country (where available). For specific country phone numbers, go to <http://www.ibm.com/planetwide/>.

Before you contact IBM Rational Software Support, gather the background information that you will need to describe your problem. When describing a problem to an IBM software support specialist, be as specific as possible and include all relevant background information so that the specialist can help you solve the problem efficiently. To save time, know the answers to these questions:

- ◆ What software versions were you running when the problem occurred?
- ◆ Do you have logs, traces, or messages that are related to the problem?
- ◆ Can you reproduce the problem? If so, what steps do you take to reproduce it?
- ◆ Is there a work-around for the problem? If so, be prepared to describe the work-around.

For Rational software product news, events, and other information, visit the [IBM Rational Software Web site](#).

Accessing the Rhapsody Documentation

Rhapsody documentation is accessible from three locations:

- ◆ On the [Telelogic Rhapsody Support Site](#)
- ◆ With Rhapsody installed on your computer, select Windows **Start** and the **Programs** menu
- ◆ Open Rhapsody and select the **Help** menu

The documentation is available in two formats: PDF and online. The online documentation displays in a browser from the Help Topics menu option. The PDF files display from the **List of Books** using the Adobe® Acrobat Reader™.

Note

To display these files properly, you must have version 6.0 or greater of this free PDF reader. To download the most up-to-date version of the Acrobat Reader, go to <http://www.adobe.com/products/acrobat/readstep2.html>.

In addition to this *Getting Started Guide*, the basic information needed to use Rhapsody can be found in the following general documentation:

- ◆ Rhapsody User Guide
- ◆ Tutorial for your development language

Help Menu Options

The Rhapsody Help menu, as shown in the following figure, lists all of the documentation access options in [Help Topics](#), [List of Books](#), and Getting Started. The first option on the Help menu redisplay the Welcome screen for quick access to many Rhapsody support features including the product documentation.

Help Topics

The **Help Topics** menu item launches the Help system version that combines the *Rhapsody User Guide* along with other detailed manuals, such as the *Team Collaboration Guide*, to supply a compilation of the Rhapsody product information.

You can navigate this large Help system using the Table of Contents, the Index, or the [Help Search Facility](#).

Help Search Facility

To search for information, type a word or words into the Search field and click **Go**.

The system lists all of the items that match the search items. The facility locates the text that contains any of the words entered for the search and prioritizes the results with a percentage probability that it is the desired information.

To find a specific instance of the search term in any of the search results, you can perform a “find in browser,” which is unique to every browser, but usually accomplished with **Ctrl+F**.

1. Type the search term in the search box that appears in your browser window and click “Find” to find the first instance of the search term in the window.
2. Clicking “Find Next,” brings up the next instance of the term, if there is one.

If no instance of the search term can be found, open another document from the search results in the left pane and repeat steps 1 and 2.

List of Books

The **List of Books** contains links to the Rhapsody manuals, primarily reference manuals, that are available as PDF files for easy printing. The User Guide is included on that list.

You may search all of the PDF files available on the **List of Books** using the browser Search facility. This facility includes a Boolean search capability.

Rhapsody Reference Documentation

The following reference manuals are available as PDF files with the software, from IBM, or from third-party product suppliers:

- ◆ *Installation Guide* describes how to install, launch, and upgrade Rhapsody. Licensing information is provided in the separate *Telelogic Lifecycles Solutions Licensing Guide* (available at <https://support.telelogic.com/lifecyclesolutions>).
- ◆ *Rhapsody Properties Reference Manual* documents the overall structure of properties within Rhapsody and the properties' uses to customize the Rhapsody environment. The individual definitions of properties and their defaults are displayed in the Features dialog box. You may also examine the complete list of Rhapsody property definitions in the *Rhapsody Property Definitions* PDF file available from the **List of Books**. That list can be searched along with the other PDF versions of the Rhapsody documentation to locate specific property definitions and the procedures that use them.
- ◆ *Team Collaboration Guide* describes techniques and tools to assist multiple users who are working together as a team and in parallel on Rhapsody projects.
- ◆ *ReporterPLUS Guide* provides information to use ReporterPLUS to create Microsoft® Word, Microsoft PowerPoint®, HTML, RTF, and text documents from any Rhapsody model.
- ◆ *API Reference Manual* describes the Rhapsody APIs including a set of COM interfaces supporting dual interfaces (COM and automation) and the Java API.
- ◆ *COM Development Guide* describes how to use Rhapsody to develop distributed applications using the Component Object Model (COM) from Microsoft®.
- ◆ *CORBA Development Guide* describes how to develop distributed CORBA applications with Rhapsody.
- ◆ *C++ Framework Execution Reference Guide* describes the Rhapsody OXF framework in detail and gives application developers reference material for the OXF framework layer classes, methods, and attributes.
- ◆ *Code Generation Guide* describes how Rhapsody generates C code from UML diagrams.
- ◆ *RTOS Adapter Guide* describes how to adapt Rhapsody to use a new run-time operating system.
- ◆ *Using the IDF Framework* describes a limited framework called IDF (Interrupt-Driven Framework) provided with Rhapsody in C.
- ◆ *TestConductor User Guide* explains the debugging and testing capabilities of this third-party visual validation tool for object-oriented embedded software designed in the Rhapsody framework.
- ◆ *ATG User Guide* describes the uses of the third-party Automatic Test Generation (ATG) tool to generate test suites and perform test execution of your Rhapsody C++ applications.

- ◆ *Rhapsody Gateway User's Guide* describes the requirements traceability functionality provided by this third-party tool.

Rhapsody Version Release Documents

Each new release of the product includes information specific to that release. These documents are available in the Rhapsody release installation kit, from the IBM support site, and from the Rhapsody **Help > List of Books**.

- ◆ **Readme** (release notes) lists the supported environments for Rhapsody, this release's new features, known restrictions, and any additional information for a specific release of the product. This information is available in the `readme.htm` file installed with Rhapsody.
- ◆ **Upgrade Guide** describes the changes to the framework, properties, and code generation between versions of Rhapsody.
- ◆ **Notices** provides detailed IBM copyright, licensing, and trademark information.

Additional Reference Materials

In addition to the Rhapsody product documentation, you may want to study UML, real-time systems, and development processes for those environments in the following books:

- ◆ *Doing Hard Time: Developing Real-time Systems with UML, Objects, Frameworks, and Patterns*, by Bruce Powel Douglass, Ph.D., Addison-Wesley, Boston, MA, 1999.
- ◆ *Harmony Deskbook*, by Hans-Peter Hoffmann, Ph.D., 2008.
- ◆ *Real Time UML: Advances in the UML for Real-time Systems*, Third edition, by Bruce Powel Douglass, Ph.D., Addison-Wesley, Boston, MA, 2004.
- ◆ *Real-time UML Workshop for Embedded Systems*, by Bruce Powel Douglass, Ph.D., Elsevier, Oxford, UK, 2007.

More Training and Learning Resources

In addition to the product documentation, the company provides you with other training and learning resources:

- ◆ [Rhapsody Training Classes](#)
- ◆ [Rhapsody eLearning Courses](#)

Rhapsody Training Classes

The company provides hands-on training for the Rhapsody product, available at an additional charge. For example, *Rhapsody Essential Tool Training* is a 4-day course that teaches you the core capabilities of Rhapsody to enable rapid development of applications using UML 2.0. Through the use of hands-on exercises creating “real world” applications, you build a strong foundation in UML 2.0 and Rhapsody and its automation of software development activities and artifact generation.

Note

Contact your IBM sales representative to purchase a one or more seats in a training class.

Rhapsody eLearning Courses

To learn more about Rhapsody, you may want to try the many Rhapsody eLearning courses. They provide self-paced online instruction, available at an additional charge. The courses cover the product fundamentals as well as the add-on products. The eLearning courses are complimentary to the instructor-led courses.

Note

Contact your IBM sales representative to purchase Rhapsody eLearning courses.

Index

A

- Acquisition viewpoint 99
- Acrobat Reader 119
 - version requirement 119
- Activity diagrams 98
- Actors 44
 - external 44, 60
 - for C 45
 - in classes 74
 - in use case diagrams 25
 - show messaging between 47
- Ada language 73
 - analyzing build errors 79
 - build application 79
 - configuration 78
 - creating classes 74
 - generating code 78
 - object model diagram 75
 - operation 76
 - package dependency 81
 - predefined packages 79
 - properties 81
 - running application 82
 - sample projects 14
 - setting entry point 77
- Aircraft Defense Management Model (ADMS) 17
- Analysis 1
- Animation 49, 66, 91
 - GUI 91
 - sequence diagrams 55, 70, 90, 92
 - statecharts 94
 - stopping 92
 - with Eclipse 40
- API
 - callback 14
 - documentation 121
 - Java 16
 - sample applications 16
- Applications
 - API listener 15
 - building 79
 - Java plug-in 17
 - plug-ins 15
 - running 82
 - testing 1
- Architecture

- DoDAF 100
- Archive 33
- Autosave 32

B

- Backups 32
 - setting 30
- Block Definition diagrams 98
- Blocks
 - Simulink 12
- Broker pattern 11
- Browsers 4, 24
 - add items 28
 - adding elements in 28
 - changing names in 26
 - copying items 28
 - deleting items 29
 - displayed in Eclipse 39
 - displaying 4
 - favorites 29
 - filtering display in 25
 - moving items 28
 - re-ordering elements 25
 - repositioning 24
 - right-click menu 26
- Build
 - Ada application 79
 - Ada configuration for 78
 - analyzing errors 79
 - using Generate/Make/Run 82

C

- C language 44
 - actors 45
 - animating the model 49
 - DiffMerge sample 13
 - Eclipse 35
 - events 53
 - FunctionalC sample 13
 - sample projects 13
 - sequence diagram 55
 - Simulink profile sample 14
 - stop program execution 58
 - test scenarios 14

- C++ language 59
 - animating the model 65
 - DiffMerge sample 11
 - Eclipse 35
 - events 69
 - object model diagrams 62
 - sample projects 10
 - sample with ports 12
 - sequence diagram 70
 - statecharts 64
 - use case diagrams 60
- Categories 24
- Checks
 - samples for external 16
 - sanitycheckpattern 11
- Classes 74
 - naming guidelines 31
 - report template 110
- ClearCase 33
- Close project 32
- Closing projects 84
- Code
 - Ada 73
 - C 44
 - C++ 59
 - Java 85
- Command-line interface
 - sample 10
- Configuration management (CM) 33
- Contacting technical support 117
- CORBA
 - sample project 10
- Customer support 115

- D**
- Delete
 - after search 22
 - from model 29
- Design 1
- Development 43
 - parallel 33
 - using Ada 73
 - using C 44
 - using C++ 59
 - using Java 85
- Diagrams 4, 8
 - drawing area 4
 - object model for C++ 62
 - sequence for C 47
 - systems engineering 8, 97, 98
 - UML 8
 - use case for C 45
 - use case for C++ 60
- DiffMerge sample for C 13
- DiffMerge sample for C++ 11
- DiffMerge tool 33
 - starting 34
 - uses 33
- Distributed team 33
- Documentation 119
 - access from Welcome screen 2
 - accessing 121
 - API 121
 - copyright information 122
 - download current 119
 - Help 120
 - installation 121
 - List of Books 9
 - printing PDF files 120
 - reference 121, 123
 - ReporterPLUS 121
 - Rhapsody samples 9
 - searching 120
 - third-party products' 121
 - version release 122
- DoDAF 99
 - supported viewpoints 100
- Drawing 75
 - area 4
 - icons 5
 - toolbar 4
- Drawing area
 - displayed in Eclipse 39

- E**
- Eclipse 35
 - creating new project 36
 - integrations 35
 - Rhapsody Debug perspective 40
 - Rhapsody integrations 35
 - Rhapsody Modeling perspective 39
 - samples for plugins 16
 - workarea 38
 - workflow integration 35
- eLearning courses 1, 124
- Elements
 - adding in browser 28
 - changing names in browser 26
 - copying in browser 28
 - moving in browser 28
- Embedded systems 1
- Errors 79
- Events 69
 - changing name 27
 - generating 53
 - naming conventions 31
- Extension points 17

- F**
- Favorites 29
- Features dialog box 26, 30

- access menu 26
- filter properties 30
- for name change 26
- General tab 45, 61
- Implementation tab 76
- L button 27
- Properties tab 81
- resize 77
- select Stereotype 77
- Files 32
 - .hep 16
 - backup 32
 - PDF 119
 - project 3
 - readersettings.ini 18
 - restore 30
 - Word 11
- Fixed size block allocation pattern 11
- Flowchart
 - patterns 13
 - sample 13

G

- Generate/Make/Run option 6, 82
- Generating
 - Ada code 78
 - events 53, 69
 - formal reports 105
 - internal report 112
 - Java events 95
- Graphical comparisons 33
- GUI
 - recompile sample code 9
 - Visual C++ 13
- Guidelines
 - for naming model elements 31

H

- Handshake pattern 11
- Hands-on training 124
- Harmony
 - process 97
 - profile 97, 98
- Help 119
 - generate support request 116
 - search facility 120
 - topics 119
- Help menu 119
 - List of Books 120
 - Welcome screen option 2
- HTML 105, 109
 - report template 111
- HTML documents 110

I

- Icons 4, 5
- IDF
 - in a sample 13
- Implementation 1
- Installation
 - documentation 121
 - repair 9
- Interfaces
 - MFC 12
 - naming conventions 31
 - naming guidelines 31
 - ReporterPLUS 108
 - Rhapsody 4
 - system for DoDAF 102
- Internal Block diagrams 98
- Internal editor 83

J

- Java language 85
 - animated sequence diagram 90
 - animating the model 91
 - API 16
 - create Eclipse project 38
 - creating customized plug-ins 41
 - Eclipse 35
 - Eclipse plug-in samples 16
 - object model diagrams 88
 - running animated model 91
 - sample API plug-ins 17
 - sample projects 14
 - statechart simplifier sample 18
 - statecharts 89, 94
 - use case diagram 86

K

- Keys
 - console 13

L

- Languages 43
 - Ada 73
 - C 13, 44
 - C++ 10, 59
 - Java 14, 85
- Launching Rhapsody 2
- Linux
 - forward & backward limitation 7
- List of Books 120
- Listener applications 16
- Log 6

M

- MasterSlave pattern 11
- Messages 47
- MFC user interface 12, 14
- MicroKernel pattern 11
- Microsoft
 - PowerPoint 105, 110
 - Windows 2, 4, 7
 - Word 105, 110
- MKS Source Integrity 33
- MODAF 99
 - supported viewpoints 99
- Models
 - add items 28
 - delete items 29
 - naming guidelines 31
 - samples 9, 43
 - search & replace 19

N

- Names
 - changing event 27
 - changing in browser 26
 - conventions for 31
 - model element guidelines 31
 - project 3
 - replacing 23
- NetCentric
 - sample model 17
- Nodes 25

O

- Object model diagrams 62
- Operations 74
 - Ada 76
 - naming conventions 31
 - primitive 48
 - search 22
- Output 19
- Output window 6
 - displayed in Eclipse 39

P

- Packages 24
 - in Component view 25
 - naming guidelines 31
 - report template 110
- Parallel development 33
- Parametric diagrams 98
- Parts
 - interaction points 12
- Patterns
 - broker 11

- C++ design 11
 - flowchart 13
 - master slave 11
 - MicroKernel 11
 - static allocation 11
- Plug-ins 15, 41
 - samples for 16, 17
- Polling pattern 11
- Ports
 - UML 2 12
- PowerPoint 105, 109
- Profiles 3
 - FunctionalC 13
 - Harmony 97, 98
 - simple plug-in 41
 - SimulinkInC 14
 - SysML 97
- Projects 3
 - backups 30
 - C++ design patterns 11
 - closing 32, 84
 - creating new 3
 - folder 24
 - full detailed report 110
 - in Eclipse 36
 - node 24
 - samples 2, 9, 43
 - saving 32
 - saving copies of 45, 59, 73, 85
 - types 3
 - Word files in 11
- Properties 30
 - backup 30
 - definition display 30, 81
 - dependency 81
 - filter 30, 81
 - Simplify 18
- Proxy pattern 11

R

- Real-time systems 1
 - reference book 123
- Redundancy
 - heterogeneous pattern 11
 - homogeneous pattern 11
- References 22
- Replace 19, 23
- Report templates
 - classes 110
 - full project detail 110
 - HTML 111
 - packages 110
 - requirements 110
 - SysML 111
- ReporterPLUS 105, 107
 - C sample 14

- C++ sample 12
 - documentation 121
 - generating reports 109
 - generating sample reports 12
 - Java sample 14
 - launch icon 2, 106
 - standard templates 110
 - starting 106
 - Reports 105
 - for presentations 105
 - formal 105
 - formats 109
 - generating 109
 - HTML 105, 109
 - internal output 113
 - PowerPoint 105, 109
 - samples 12
 - types 106
 - Word format 105
 - Requirements 1, 60, 97
 - diagrams 98
 - in a sample 11
 - report template 110
 - view 25
 - with tags report 12
 - Restore projects files 30
 - Rhapsody 1
 - add-ons 97
 - animation 40
 - autosave 32
 - backup files 32
 - browser 4, 24
 - close project 32
 - customer support 115
 - diagrams 8
 - documentation 119
 - drawing icons 5
 - drawing toolbar 4
 - Eclipse Debug perspective 40
 - Eclipse Modeling perspective 39
 - Eclipse platform integration 35
 - eLearning courses 1, 124
 - exiting 32
 - formal reports 105
 - generating reports 112
 - guided tour 4
 - information 1
 - internal editor 83
 - launching 2
 - official samples 2, 43
 - On Demand Webinars 1
 - output window 6
 - project saving 32
 - project types 3
 - recompile samples 9
 - repair installation 9
 - reports 105
 - restore projects 30
 - sample models 43, 44, 45, 59, 73, 85
 - samples 9
 - starting 2
 - starting ReporterPLUS 106
 - training 1, 124
 - tutorials 1
 - units 33
 - Welcome screen 2
 - window 5
- ## S
- Samples 9, 43
 - accessing from Welcome screen 43
 - Ada 14
 - API for C++ 10
 - automated railcars system 10
 - C++ design patterns 11
 - C++ Radio 12
 - CD_Player 10
 - command-line interface 10
 - CORBA 10
 - CPP for callback API 14
 - DiffMerge 11, 13
 - directory 43
 - Dishwasher 11, 13, 14
 - Elevator 11, 13
 - flowchart 13
 - FunctionalC 13
 - Handset 11
 - hhs (home heating system) 11, 13
 - HomeAlarm 12, 14
 - HomeAlarm with Ports for C++ 12
 - icon on Welcome screen 43
 - Java API 16
 - Java for callback API 15
 - Java plug-in 17
 - NetCentric 17
 - official Rhapsody 43
 - Pacemaker 12, 13
 - PingPong 12
 - plug-in 41
 - radio 14
 - recompile GUI code 9
 - ReporterPLUS 12
 - requirements 11
 - saving copies of 45, 59
 - Simulink 12
 - Simulink in C 14
 - statecharts 18
 - SysML profile 17
 - systems 17
 - systems engineering 17
 - telecommunications for C 14
 - telecommunications for C++ 12
 - Tetris 12

- UML 2.0 12
 - VB 18
 - VB for callback API 15
 - VB post simplification 17
 - VBA for callback API 16
 - Sanity check pattern 11
 - Saving
 - autosave 32
 - project 32
 - projects 45, 59, 73, 85
 - Scenarios
 - testing in C 14
 - Screen snapshot 117
 - Search 19
 - and replace 23
 - delete item 22
 - launch methods 19
 - references 22
 - results 6
 - results changes 22
 - Sequence diagrams 47, 55, 70, 98
 - comparing animated 93
 - multiple 12
 - saving animated 92
 - Simplify property 18
 - Simulation 1
 - Simulink
 - C sample 14
 - C++ sample 12
 - Snapshot 117
 - Software developers 1, 43
 - Starting 106
 - Statecharts 64, 98
 - animating 51, 67
 - Java 94
 - samples 18
 - Simplifier property 18
 - VB simplifier 18
 - writer rules 18
 - Static priority pattern 11
 - Stereotype 77
 - Stereotypes 111
 - Stopping
 - animation 92
 - Strategic viewpoint 99
 - SysML 1, 97, 98
 - handset sample 17
 - profile 97
 - report template 111
 - sample model 17
 - Systems
 - embedded 1, 73
 - engineers 1
 - real-time 1
 - samples 17
 - Systems engineering 97
 - automated tools 98
 - diagrams 8, 98
 - samples 17
 - training 98
- ## T
- Technical support 115, 117
 - current Rhapsody customers 115
 - new customers 118
 - Templates
 - DoDAF 99
 - ReporterPLUS 110
 - Testing 1, 96
 - sample project 14
 - Text editor 61
 - Time slicing pattern 11
 - Training 124
 - Tutorials 1
 - samples created in 43
- ## U
- UML 1, 97
 - ports 12
 - reference book 123
 - sample 2.0 project 12
 - Units 33
 - Use case diagrams 45, 60, 98
- ## V
- VB
 - post simplification sample 17
 - user-defined simplifier sample 18
 - VBA 12
 - sample for callback API 16
 - Video capture 117
 - Viewpoints
 - acquisition for MODAF 99
 - all views for DoDAF and MODAF 100
 - DoDAF 100
 - operational for DoDAF and MODAF 100
 - strategic for MODAF 99
 - systems for DoDAF and MODAF 100
 - technical for DoDAF and MODAF 100
- ## W
- Watchdog
 - pattern 11
 - Web site 119
 - access 119
 - Webinars 1
 - Welcome screen 2
 - accessing model samples 43
 - start ReporterPLUS from 106

Windows 5
Eclipse integrations 35
managing 7
open 7

output 6
Word
files in project 11
Word documents 110

