



Ada Tutorial



# Ada Tutorial for Rational Rhapsody



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

This edition applies to IBM® Rational® Rhapsody® 7.4 and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1997, 2009.

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

# Contents

---

<b>Getting Started</b> .....	<b>1</b>
<b>Rational Rhapsody in Ada Tutorial Overview</b> .....	<b>1</b>
Audience for this Tutorial .....	1
Before You Begin .....	2
Tutorial Objectives .....	3
Documentation Conventions .....	4
About the Rational Rhapsody Product .....	4
<b>Setting up for the Tutorial</b> .....	<b>6</b>
Starting the Rational Rhapsody Product .....	6
Closing the Rational Rhapsody Product .....	6
Creating a Rational Rhapsody Project .....	7
Renaming the Default Package .....	9
Saving a Rational Rhapsody Project .....	10
Opening the Dishwasher Model .....	11
<b>About a Rational Rhapsody Project</b> .....	<b>12</b>
About Project Files and Folders .....	13
Using Naming Conventions and Project Guidelines .....	14
<b>Rational Rhapsody User Interface</b> .....	<b>15</b>
Toolbars .....	16
Browser .....	17
Drawing Area .....	18
Output Window .....	18
Drawing Toolbars .....	18
Features Dialog Box .....	19
<b>Summary</b> .....	<b>23</b>
<b>Lesson 1: Creating a Use Case Diagram</b> .....	<b>25</b>
<b>Goals for this Lesson</b> .....	<b>25</b>
<b>Exercise 1: Analyzing the Dishwasher System</b> .....	<b>25</b>
<b>Exercise 2: Creating the Dishwasher Use Case Diagram</b> .....	<b>27</b>
Task 2a: Creating the Dishwasher Use Case Diagram .....	28
Task 2b: Drawing the Boundary Box and Actors .....	30

## Table of Contents

---

Task 2c: Drawing the Use Cases . . . . .	31
Task 2d: Associating Actors with Use Cases . . . . .	33
Task 2e: Adding a Diagram Title . . . . .	34
<b>Summary . . . . .</b>	<b>35</b>
<b>Lesson 2: Creating an Object Model Diagram . . . . .</b>	<b>37</b>
<b>Goals for this Lesson . . . . .</b>	<b>37</b>
<b>Exercise 1: Creating the Dishwasher Object Model Diagram . . . . .</b>	<b>38</b>
Task 1a: Creating the Dishwasher Object Model Diagram . . . . .	39
Task 1b: Drawing Classes and Dependencies . . . . .	40
Task 1c: Creating a Singleton . . . . .	42
Task 1d: Adding Attributes . . . . .	43
Task 1e: Creating Operations . . . . .	44
Task 1f: Displaying Attributes and Operations in the OMD . . . . .	46
Task 1g: Adding the setup Operation . . . . .	48
Task 1h: Adding a main Operation to the Display Class . . . . .	50
Task 1i: Using the Entrypoint Stereotype . . . . .	53
<b>Exercise 2: Other Necessary Tasks . . . . .</b>	<b>55</b>
Task 2a: Saving Packages Separately . . . . .	55
Task 2b: Using Predefined Packages . . . . .	56
Task 2c: Establishing the Package Dependency . . . . .	59
Task 2d: Setting a Package Dependency Property . . . . .	60
Task 2e: Adding a default constructor . . . . .	62
<b>Summary . . . . .</b>	<b>63</b>
<b>Lesson 3: Generating Code and Building Your Model . . . . .</b>	<b>65</b>
<b>Goals for this Lesson . . . . .</b>	<b>65</b>
<b>Exercise 1: Preparing for Generating Code . . . . .</b>	<b>65</b>
Task 1a: Creating a Component . . . . .	66
Task 1b: Setting the Component Features . . . . .	67
Task 1c: Creating a Configuration . . . . .	69
<b>Exercise 2: Generating Code . . . . .</b>	<b>69</b>
Task 2a: Generating Code . . . . .	69
Task 2b: Fixing Code Generation Errors . . . . .	70
<b>Exercise 3: Building Your Model . . . . .</b>	<b>71</b>
Task 3a: Building your Model . . . . .	71
Task 3b: Fixing Build Errors . . . . .	72
Task 3c: Viewing Code . . . . .	73
<b>Summary . . . . .</b>	<b>73</b>

---

<b>Lesson 4: Creating a Statechart</b> .....	<b>75</b>
<b>Goals for this Lesson</b> .....	<b>75</b>
<b>Exercise 1: Creating the Dishwasher Statechart</b> .....	<b>76</b>
Task 1a: Creating a Statechart .....	77
Task 1b: Drawing the Dishwasher Statechart .....	77
Task 1c: Drawing History and Diagram Connectors .....	79
Task 1d: Drawing Default Connectors .....	81
Task 1e: Adding Ada Operations .....	82
Task 1f: Drawing the Transitions .....	83
Task 1g: Adding Actions to States .....	85
Task 1h: Changing Operation Synchronization .....	88
Task 1i: Adding a Diagram Title .....	90
<b>Exercise 2: Generating Code and Building Your Model</b> .....	<b>91</b>
Task 2a: Generating Code .....	91
Task 2b: Building the Model .....	92
<b>Summary</b> .....	<b>92</b>
<b>Lesson 5: Creating a Sequence Diagram</b> .....	<b>93</b>
<b>Goals for this Lesson</b> .....	<b>93</b>
<b>Exercise 1: Creating the KeyPress Event</b> .....	<b>94</b>
Task 1a: Creating an Event .....	94
<b>Exercise 2: Creating the Execution Sequence Diagram</b> .....	<b>95</b>
Task 2a: Creating the Sequence Diagram .....	97
Task 2b: Creating the Workflow for Your Sequence Diagram .....	98
<b>Summary</b> .....	<b>100</b>
<b>Lesson 6: Building and Running the Model</b> .....	<b>101</b>
<b>Goals for this Lesson</b> .....	<b>101</b>
<b>Exercise 1: Creating the Build Object Model Diagram</b> .....	<b>102</b>
Task 1a: Creating the Build Object Model Diagram .....	103
Task 1b: Creating a DishwasherBuilder Class .....	103
<b>Exercise 2: Generating Code and Building Your Model</b> .....	<b>105</b>
Task 2a: Creating Another Configuration .....	105
Task 2b: Generating Code .....	107
Task 2c: Troubleshooting the Build .....	110
Task 2d: Roundtripping .....	111
<b>Exercise 3: Running Your Model</b> .....	<b>111</b>
Task 3a: Running your Dishwasher Model .....	111
<b>Summary</b> .....	<b>113</b>

---

<b>Lesson 7: Animating Your Application</b> .....	<b>115</b>
<b>Goals for this Lesson</b> .....	<b>115</b>
<b>Exercise 1: Animating your Application</b> .....	<b>116</b>
Task 1a: Starting Animation .....	116
Task 1b: Viewing the Animated Statechart .....	117
Task 1c: Invoking Commands for your Program .....	118
Task 1d: Setting Breakpoints .....	121
Task 1e: Quitting Animation .....	122
<b>Summary</b> .....	<b>122</b>
<b>Index</b> .....	<b>123</b>



# Getting Started

---

Welcome to the *Ada Tutorial for IBM Rational Rhapsody*! IBM® Rational® Rhapsody® is the Model-Driven Development environment of choice for systems engineers and software developers of either embedded or real-time systems.

## Rational Rhapsody in Ada Tutorial Overview

This tutorial shows you how to use the Rational Rhapsody product to analyze, design, and build a model of a dishwasher using a file-based modeling approach. Before you begin creating this model, you need to consider the functions of the dishwasher.

A dishwasher has users who use it to wash dishes. Another user would be a service person who makes repairs and the like. When you use a dishwasher, you start functions that the dishwasher must perform, such as turning on and changing cycles.

For this tutorial, you are going to create a project called Dishwasher.

This tutorial helps you become familiar with the Rational Rhapsody product. You should consider it part of the Rational Rhapsody learning process, in addition, for example, to the *Rhapsody Essential Tool Training* class and the Rational Rhapsody eLearning courses, both of which are available at an additional cost.

The Rational Rhapsody product contains a number of sample models in Ada that you can review to help familiarize yourself with Rational Rhapsody in Ada. These Ada sample models are in the `<Rational Rhapsody installation>\Samples\AdaSamples` subfolder. If you make changes to one of the sample models, you might want to save your version in another folder, in case you want to refer to the original state of the provided sample model later. Choose **File > Save As** to save your version of a sample model.

## Audience for this Tutorial

The intended audience for this tutorial is system engineers and software engineers who are familiar with the Ada language. The tutorial assumes that you are familiar with UML™ (Unified Modeling Language™) and Object Oriented concepts.

## Before You Begin

Before you work through this tutorial, you might find it helpful to review the *Getting Started Guide* for the Rational Rhapsody product. It provides a functional overview for the Rational Rhapsody product for system designers, system engineers, and software developers with more functions (meaning how to do something), explanations, and details than this tutorial provides.

In addition, throughout the tutorial, references are made to other Rational Rhapsody documentation where appropriate. Note also that the *IBM Rational Rhapsody User Guide* has a Glossary section that you might find useful.

Note the following:

- ◆ This tutorial assumes that Rational Rhapsody (version 7.1 or greater) is installed on your system and that you have a valid permanent license. Contact the Rational Rhapsody Technical Support staff if you need assistance with installation or licensing.
- ◆ You must have installed the compiler necessary to generate code. You should have done this before you installed the Rational Rhapsody product because during the Rational Rhapsody installation process you identify the path to the compiler.
- ◆ Before you can work through any of the lessons in this tutorial, you must create the Dishwasher project, which is detailed in [Setting up for the Tutorial](#).
- ◆ You should work through the tutorial in the order of the lessons. During the course of working through this tutorial, you generate code as well as build your model at various stages. For example, in the lesson where you first learn how to generate code, you will get error messages because you have not yet created certain operations that you refer to in your code. Once you work through the next lesson, you will no longer get those error messages (though you might get others).

You might also find it useful to review documentation found within the Rational Rhapsody installation path (for example, <Rational Rhapsody installation>\Sodius\RiA\_CG\help).

## Tutorial Objectives

This tutorial develops an example of an embedded system for a dishwasher. This dishwasher example shows you how to use the Rational Rhapsody product to create the software that controls the simplified operation of a dishwasher. The operations of the dishwasher are simplified to minimize the complexity of the tutorial.


This tutorial includes Ada code examples within instructions where code entries are required.

When you have completed this tutorial, you will have performed the following standard tasks:

- ◆ Created a project
- ◆ Created use case diagrams, which show the main functions of the system (use cases) and the entities that are outside the system
- ◆ Created object model diagrams, which specify the structure of the classes, objects, and interfaces in the system and the static relationships that exist between them
- ◆ Created statecharts, which define the behavior of classifiers (actors, use cases, or classes), objects, including the states that they can enter over their lifetime and the messages, events, or operations that cause them to transition from state to state
- ◆ Created sequence diagrams, which show structural elements communicating with one another over time
- ◆ Generated code
- ◆ Built a model
- ◆ Run a model
- ◆ Animate a model

## Documentation Conventions

This document uses the following conventions:

- ◆ **Boldtype** for names of GUI objects and controls, including selection choices; and emphasis. Examples:
  - On the **General** tab, in the **Stereotype** box, select the **entrypoint in PredefinedTypesAda** check box from the drop-down menu.
  - Drag-and-drop the **Display** class onto the **Packages** object model diagram.
  - Click the **Create Boundary box** button  on the **Drawing** toolbar.
  - If the Rational Rhapsody browser does not display, select **View > Browser**.
  - A project file, called `<project_name>.rpy`.
- ◆ Courier font in 10 point for pathnames, system messages, and items that you have to type. Examples:
  - The Output window displays the message `Animation session terminated`.
  - In the **Project name** box, replace the default project name with `Dishwasher`.
  - Draw a transition from the **Running** to the **Open** state and label it `op_open`.
- ◆ *Italics* for the first mention of a concept with an explanation.

## About the Rational Rhapsody Product

Rational Rhapsody uses visual design to develop embedded software allowing you to perform these tasks:

- ◆ **Analyze**, during which you can define system requirements, identify necessary components, and define their structure and behavior using the (UML) *diagrams*.
- ◆ **Design**, during which you can specify and design the architecture, taking into account architectural, mechanistic, and detailed design considerations.
- ◆ **Implement**, during which you can automatically generate code from the analysis model and then build and run it within the Rational Rhapsody product.
- ◆ **Test**, during which you can animate the model on the local host or a remote target to perform design-level debugging within animated views.

## UML Diagrams

The following are the UML diagrams in Rational Rhapsody:

- ◆ **Use Case Diagrams** show the main functions of the system (use cases) and the entities (actors) outside the system.
- ◆ **Structure Diagrams** show the system structure and identify the organizational pieces of the system.
- ◆ **Object Model Diagrams** show the structure of the system in terms of classes, objects, and the relationships between these structural elements.
- ◆ **Sequence Diagrams** show sequences of steps and messages passed between structural elements when executing a particular instance of a use case.
- ◆ **Activity Diagrams** specify a flow for classifiers (classes, actors, use cases), objects, and operations.
- ◆ **Statecharts** show the behavior of a particular classifier (class, actor, use case) or object over its entire life cycle.
- ◆ **Collaboration Diagrams** provide the same information as sequence diagrams, emphasizing structure rather than time.
- ◆ **Component Diagrams** describe the organization of the software units and the dependencies among units.
- ◆ **Deployment Diagrams** show the nodes in the final system architecture and the connections between them.

## Setting up for the Tutorial

Before you can work through this tutorial, you must create and set up the Dishwasher project, which you do in this section. The following tasks show you how to:

- ◆ Create the Dishwasher project
- ◆ Save a project

## Starting the Rational Rhapsody Product

### Windows

To start the Rational Rhapsody product in **Windows**: Select **Start > Programs > IBM Rational > IBM Rational Rhapsody Version# > Rhapsody Development Edition > Rhapsody in Ada**; or, if available, click the **Rhapsody in Ada** icon on your the desktop.

### Linux

To start the Rational Rhapsody product in **Linux**, use these steps:


1. From the Terminal, browse to the Rational Rhapsody home directory.
2. Execute the `RhapsodyInAda` script. For example:

```
[RhapsodyUser@MyHostMachine]# cd /home/Rhapsody  
[RhapsodyUser@MyHostMachine Rhapsody]# ./RhapsodyInAda
```

In this example, “RhapsodyUser” is the username, “MyHostMachine” is the host machine and “/home/Rhapsody” is the installation directory.


## Closing the Rational Rhapsody Product

To close the Rational Rhapsody product, follow these steps:

1. Save your changes.
2. Choose **File > Exit** or click the **Close** button .

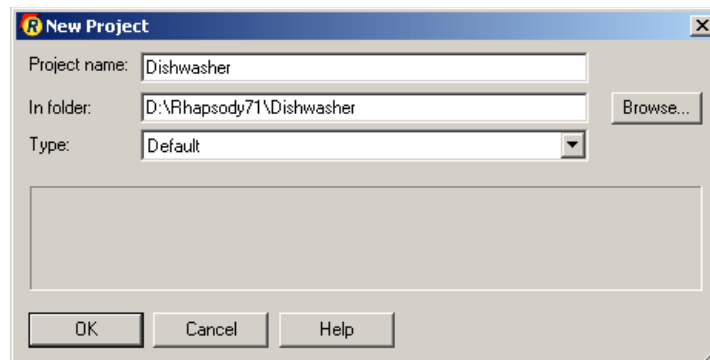
## Creating a Rational Rhapsody Project

To create a new Rational Rhapsody project, follow these steps:

1. Start the Rational Rhapsody product if it is not already running. If necessary, see [Starting the Rational Rhapsody Product](#).
2. Click the **New** button  on the main toolbar or select **File > New** to open the New Project dialog box.
3. In the **Project name** box, replace the default project name with `Dishwasher`.
4. In the **In folder** box, browse to find an existing folder or enter a new folder name.

**Note:** To avoid potentially long pathnames, do not create the project on the desktop.

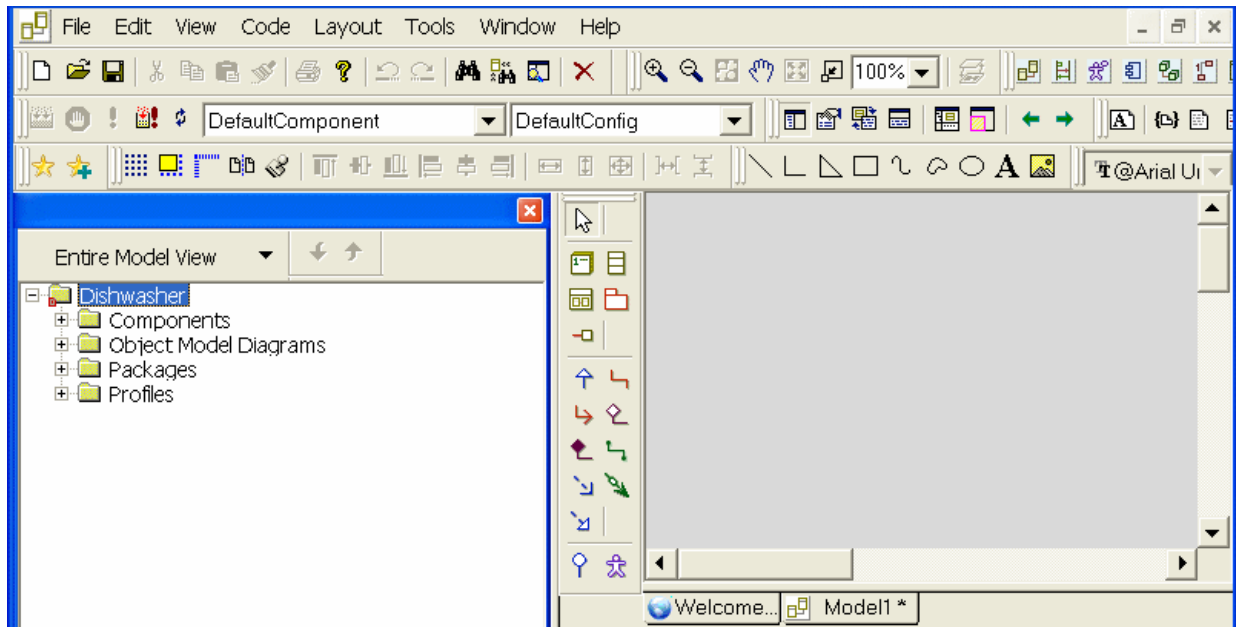
5. In the **Type** box, accept **Default**, which provides all of the basic UML structures. It is useful for most Rational Rhapsody projects. Your dialog box should resemble the following figure:



**Note:** The **Default** and **AdaCodeGeneration** choices are equivalent and provide standard UML features tailored for Ada code generation. The **SPARK** choice is the Rational Rhapsody product's Ada SPARK profile. For a description of the available project profile types that you can select from the **Type** drop-down list, refer to the *IBM Rational Rhapsody User Guide*. (Do a search of the user guide PDF file for “specialized profile.”)

- Click **OK**. The Rational Rhapsody product verifies that the specified location exists. If it does not exist, Rational Rhapsody asks whether you want to create it. Click **Yes**.

Rational Rhapsody creates your project in the new **Dishwasher** subfolder, opens the project, and displays the Rational Rhapsody browser in the left pane and the drawing area for an object model diagram (by default because of your **Type** [project profile] choice on the **New Project** dialog box), as shown in the following figure. For more information about a Rational Rhapsody project, see [About a Rational Rhapsody Project](#).



- Rename the default package; see [Renaming the Default Package](#).



## Renaming the Default Package

When you create a Rational Rhapsody project, the system creates a package for you called **Default**. In this task, you are going to rename it to something that better reflects this project.

*Packages* can be used to divide the model into functional domains or subsystems, which consist of objects, object types, functions, variables, and other logical artifacts. They can be organized into hierarchies to provide a high level of partitioning.

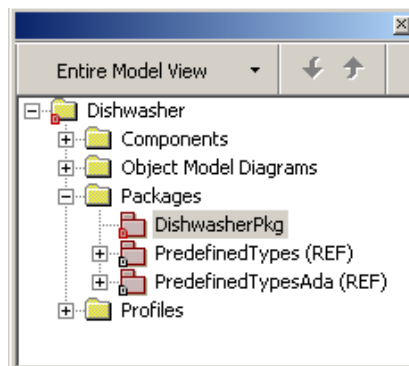
To rename the **Default** package, follow these steps:

1. In the Rational Rhapsody browser, expand the **Packages** category.
2. Double-click the **Default** package or right-click it and select **Features** to open the Features dialog box.
3. On the **General** tab, in the **Name** box, replace `Default` with `DishwasherPkg`, as shown in the following figure:




4. Click **OK**.

Your Rational Rhapsody browser should resemble the following figure:



## Saving a Rational Rhapsody Project

To save a Rational Rhapsody project in the current location, use one of the following methods:

- ◆ Click the **Save** button  on the main toolbar
- ◆ Select **File > Save**.

To save the project to a new location, select **File > Save As**.

Note that the **Save** command saves only the modified units, reducing the time required to save large projects.

A *unit* is a composite model element stored in its own file that you can check in and out of a Content Management system. A model element can be made into a unit as long as it can be saved as a separate file. Some elements that can be saved as units are the entire model, packages, classes, any type of Rational Rhapsody diagram, and components. The project, represented by the root node displayed in the browser, is always a unit. The primary purpose of units is to support collaboration with other developers.

### About Autosave

The Rational Rhapsody product automatically performs an autosave every ten minutes to back up changes made between saves. Modified units are saved in an autosave folder (`<project_name>_auto_rpy`), along with any units that have a time stamp older than the project file. Note that the autosave folder appears only when necessary (after ten minutes if a save has not been made) and disappears after you save.


### About Backups

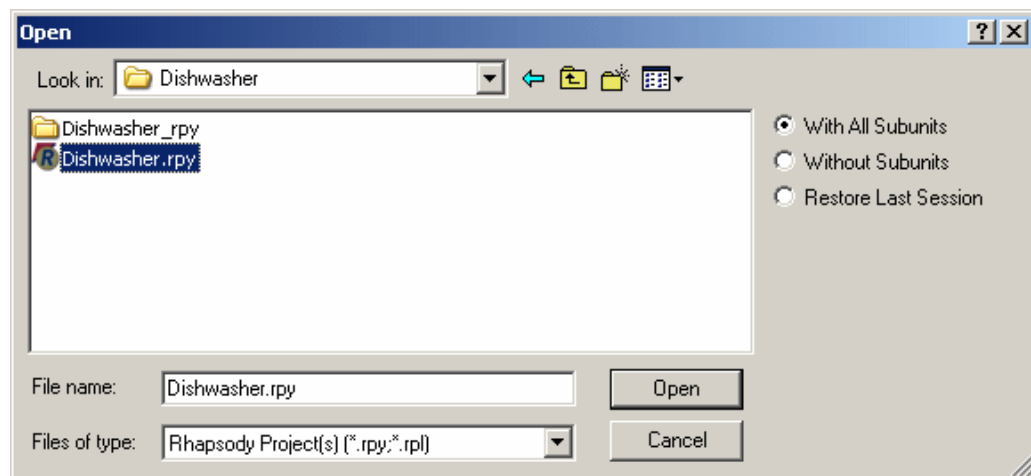
You can set a property (`General::Model::BackUps`) to create backups of your model every time you save your project. This gives you the opportunity to revert to a previously saved version if you encounter a problem. By default, Rational Rhapsody does not create backups. Refer to the *IBM Rational Rhapsody User Guide* for more information about creating backups. (Do a search of the user guide PDF file for “backups.”)

## Opening the Dishwasher Model

Once you have created, saved, and closed the Dishwasher model, you can open and work on it at any time.

To open the Dishwasher model, follow these steps:

1. Start Rational Rhapsody if it is not already running. If necessary, see [Starting the Rational Rhapsody Product](#).
2. Click the **Open** button  on the main toolbar or select **File > Open** to open the Open dialog box.
3. Navigate to the location in which you saved the Dishwasher project.
4. Select **Dishwasher.rpy**, or type the name of the project file in the **File name** box, as shown in the following figure:



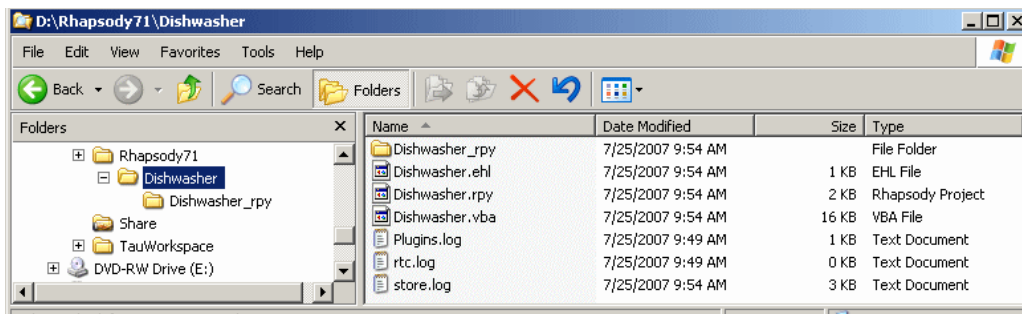
5. Accept the default **With All Subunits** option.

This choice means that the Rational Rhapsody product will load all units in the project. Refer to the *IBM Rational Rhapsody User Guide* for information about the options. (Do a search of the user guide PDF file by the option names.)

6. Click **Open**. Rational Rhapsody opens the Dishwasher model.

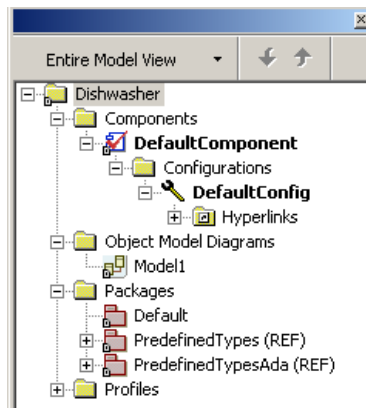
## About a Rational Rhapsody Project

A Rational Rhapsody project includes the UML diagrams, packages, and code generation configurations that define the model and the code generated from it. When you create a new project, Rational Rhapsody creates a project folder that contains the *project files* in the specified location. The name you choose for your new project is used to name project files and folders, as shown in the following figure.



For more information about the folders and files that are part of a Rational Rhapsody model, see [About Project Files and Folders](#).

In addition, the name appears at the top level of the project hierarchy in the Rational Rhapsody browser. Rational Rhapsody provides several default elements in the new project: a object model diagram, package, component, and configuration, as shown in the following figure:



An *element* is an atomic constituent of a model. In the Rational Rhapsody product, primary model elements within the browser are packages, classes, object model diagrams, associations, dependencies, operations, variables, events, event receptions, triggered operations, constructors, destructors, and types. Primary model elements in object model diagrams are packages, classes, associations (links), dependencies, and actors.

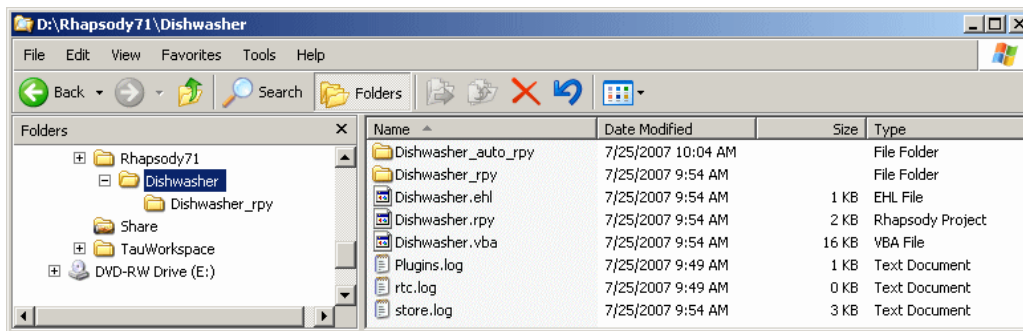
## About Project Files and Folders

The Rational Rhapsody product creates the following files and subfolders in the project folder. Some folders and files are created when you initially create a project, others only when applicable.

- ◆ A project folder, called **<project\_name>\_rpy**, which contains the unit files for the project, including UML diagrams, packages, and code generation configurations.
- ◆ A project file, called **<project\_name>.rpy**.
- ◆ A subfolder, called **<project\_name>\_auto\_rpy**, which appears only when necessary (after ten minutes if a save has not been made) and disappears after you save.
- ◆ An event history file, called **<project\_name>.ehl**, which contains a record of events injected during animation, and active and nonactive breakpoints. This file appears after your first save of a project.
- ◆ Log files, which record when projects were loaded and saved in the product; for example, **load.log** and **store.log**.
- ◆ A .vba file, called **<project\_name>.vba**, which contains macros or wizards.
- ◆ Backup project files and folders (**<project\_name>\_bak1\_rpy**, **<project\_name>\_bak2\_rpy**), which are optional, depending on project settings.
- ◆ An **\_RTC** subfolder, when applicable, which holds any tests created using the Rational Rhapsody TestConductor™ add-on.

The **<project\_name>.rpy** file and the **<project\_name>\_rpy** folder are necessary for the generation of source code.

The following figure shows the project folder for the Dishwasher project and some of its files and subfolders.



## Using Naming Conventions and Project 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 Rational 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.

### 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)

### Model Element Names

The names of the elements themselves should follow conventions such as these:

- ◆ File, block, and class names begin with an upper case letter, such as “System.”
- ◆ Functions and variables begin with lower case letters, such as “restartSystem.”
- ◆ Upper case letters to separate concatenated words, such as “checkStatus.”

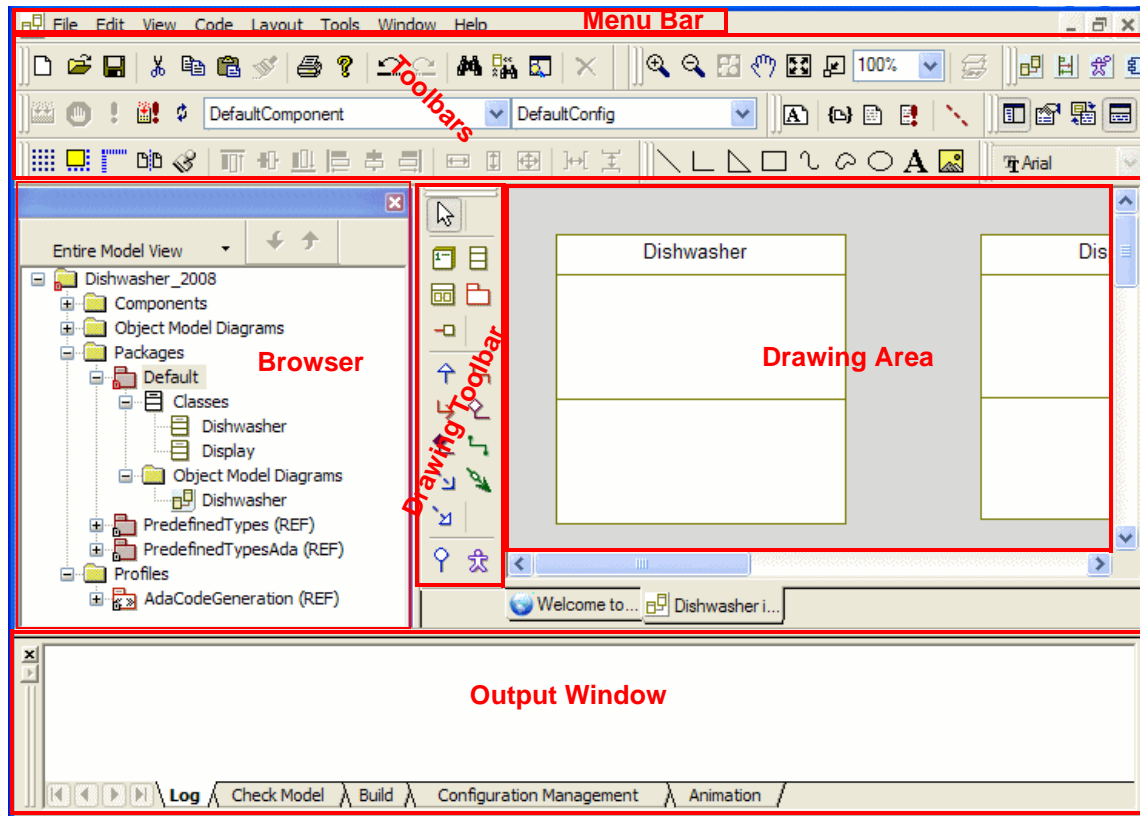
### Rational Rhapsody Project Guidelines

The following guidelines can help you design and structure your Rational Rhapsody project:

- ◆ Do not give the same name to several different elements of the model (for example, packages and classes). As shown in this tutorial, you could add a “Pkg” suffix to your packages (such as “DishwasherPkg”).
- ◆ Use an iterative approach for your model. Periodically save/generate/make/animate your model to see if it is working as expected. This way, you can catch errors earlier in the process when it should be easier to fix problems.
- ◆ Do not try to put everything into a single diagram. This can make reading the diagram difficult. Creating different diagrams to give you different views can really help you understand the model more.

## Rational Rhapsody User Interface

Before proceeding with this tutorial, you should become familiar with the main features of the Rational Rhapsody graphical user interface (GUI). The Rational Rhapsody GUI is made up of three key windows and different toolbars for each of the UML diagram types. The following figure shows the Rational Rhapsody GUI.



### Toolbars

The Rational Rhapsody toolbars provide quick access to the commonly used commands. These commands are also available from the menus. The Rational Rhapsody product has the following toolbars:

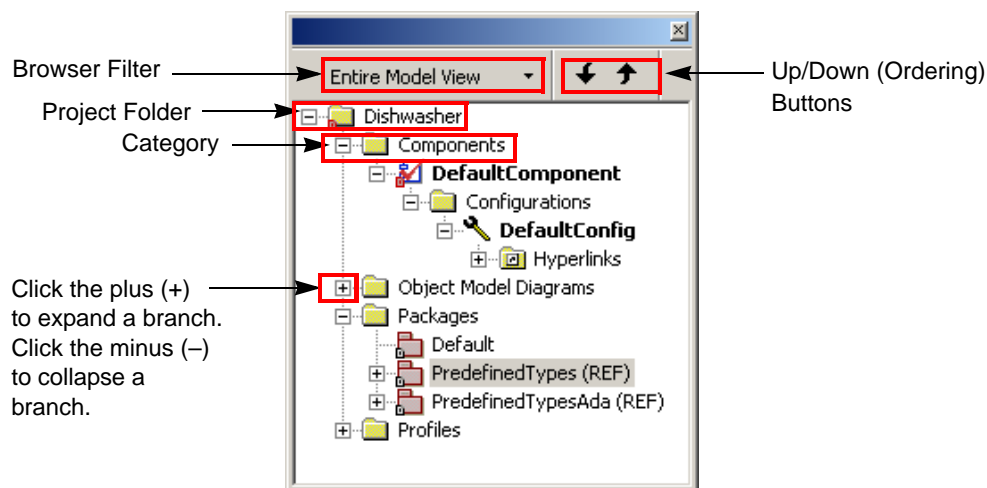
- ◆ **Standard** has buttons for the frequently used options on the File, Edit, and Help menus. Examples: New, Open, Save; Copy, Paste, Locate in Browser; About.
- ◆ **Code** has buttons for the frequently used options on the Code menu, such as Make, **Run** executable and GMR (for Generate/Make/Run).
- ◆ **Windows** has buttons for the frequently used options on the View menu, such as Show/Hide Browser and Show/Hide output window.
- ◆ **Diagrams** has buttons for the part of the Tools menu that give you quick access to the diagrams in the project, such as Sequence Diagrams and Open Statechart.
- ◆ **VBA** has buttons to provide access to the VBA options, such as VBA Editor and Show Macros Dialog. Note that VBA is for Windows only.
- ◆ **Animation** has buttons for the animation options during an animation session, such as Go, Animation Break, and Quit Animation.
- ◆ **Layout** has buttons that help you with the layout of elements in your diagram, such as Snap to Grid, Align Top, and Align Left.
- ◆ **Drawing** has buttons for the graphics editor used to create and edit diagrams. Each **Drawing** toolbar is unique to its particular diagram type. For example, the **Drawing** toolbar for a sequence diagram is different from that for a statechart.
- ◆ **Common Drawing** has buttons to add requirements, comments, and other annotations to any diagram, such as Note and Requirement.
- ◆ **Free Shapes** has buttons for basic drawing shapes, such as Polyline and Polycurve.
- ◆ **Zoom** has buttons to zoom options, such as Zoom In, Zoom Out, and Pan.
- ◆ **Format** has buttons for various text formatting options and line/fill options, such as Italic and Font Color.



## Browser

The Rational Rhapsody browser shows the contents of the project in an expandable tree structure. By default, it is the upper, left-hand part of the Rational Rhapsody GUI. 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*.

A project consists of at least one package in the **Packages** category. A package contains UML elements, such as classes, files, and diagrams. Rational Rhapsody automatically creates a default package called **Default**, which it uses to save model parts unless you specify a different package. The following figure shows an example of the browser.



### Filtering the Browser

The browser filter lets you display only the elements relevant to your current task.

To filter the Rational Rhapsody browser, click the drop-down arrow at the top of the browser window, and select the view you want to see from the menu. Refer to the *IBM Rational Rhapsody User Guide* for information on the view options.

### Repositioning the Browser

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

## Drawing Area

The drawing area displays the graphic editors and code editors, and it is the region for drawing diagrams. By default, it is the upper, right-hand section of the Rational Rhapsody GUI. Rational Rhapsody displays each diagram with a tab that includes the name of the diagram and an icon that denotes the diagram type. When you make changes to a diagram, Rational Rhapsody displays an asterisk after the name of the diagram in the title bar to indicate that you must save your changes.

## Output Window

The Output window displays Rational Rhapsody messages. By default, it is the lower section of the Rational Rhapsody GUI. It includes tabs that display the following types of messages:

- ◆ Log
- ◆ Check Model
- ◆ Build
- ◆ Configuration Management
- ◆ Animation
- ◆ Search Results

If the Output window does not appear, choose **View > Output Window**.

## Drawing Toolbars

The Rational Rhapsody product displays a separate **Drawing** toolbar for each UML diagram type. By default, it places the **Drawing** toolbar to the left of the diagram.

To move the toolbar, click and drag it to another location.

## Features Dialog Box

The Features dialog box lets you view and edit the features of an element in the Rational Rhapsody product.

To open the Features dialog box, do one of the following:

- ◆ Double-click an element (for example, **Out** [an interface])
- ◆ Right-click an element (for example, **Subsystem Architecture** [a diagram]) and then select **Features**
- ◆ Select an element and press **Alt + Enter**
- ◆ Select an element and select **View > Features**

You can resize the Features dialog box and hide the tabs on it if you want. For more information about the Features dialog box, refer to the section on it in the *IBM Rational Rhapsody User Guide*.

### Keeping Open the Features Dialog Box

Once you open the Features dialog box, you can leave it open and select other elements to view their features. This means that after you make changes to the Features dialog box for an element in your drawing or on the Rational Rhapsody browser, you can click **Apply**. Then, without closing the dialog box, you can select another element to view its features. Once you are done with the Features dialog box, you click **OK** to close it.

#### Note

---

Even though you clicked **Apply** or **OK** for your changes in the Features dialog box, you must still save your model to save all the changes you made. Clicking **Apply** or **OK** applies any changes to the currently opened model. However, to save the changes for the model so that they are in effect the next time you open it, you must save your model.

Note the following about the **Apply** and **OK** buttons on the Features dialog box:

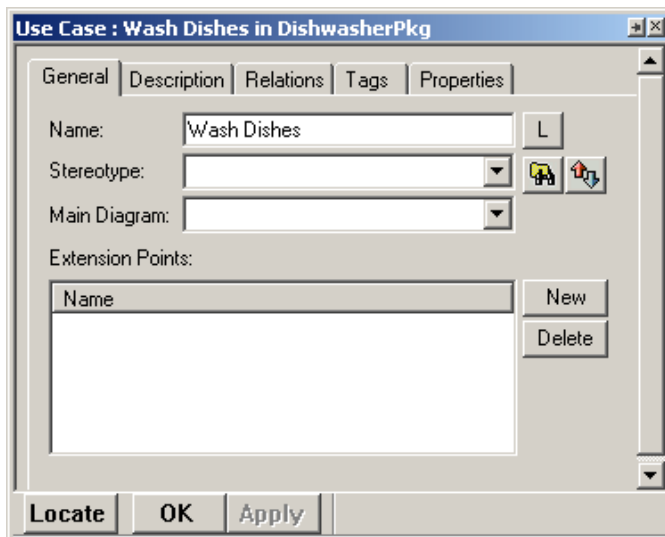
- ◆ Click **Apply** when you want to apply any changes you made to the Features dialog box but want keep it open. For example, you might need to apply a change before you can continue with using the Features dialog box, or you want to apply a change and see its effect before continuing making any more changes on the dialog box.
- ◆ Click **OK** when you want to apply your changes and close the Features dialog box at the same time.

## Tabs for the Features Dialog Box

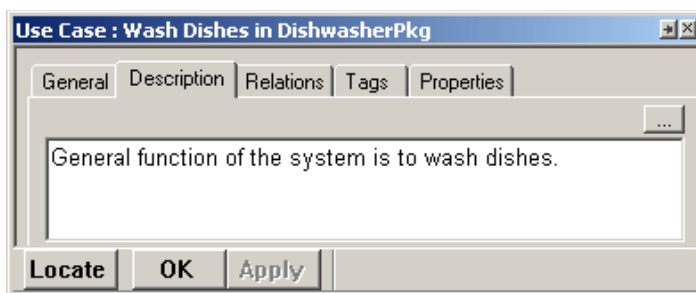
The Features dialog box has different tabs at the top of the dialog box and different boxes on the tabs depending on the element type.

The following tabs are common to all types of elements. For more information about these tabs, as well as the other tabs that you might see in the Features dialog box, refer to the section on it in the *IBM Rational Rhapsody User Guide*.

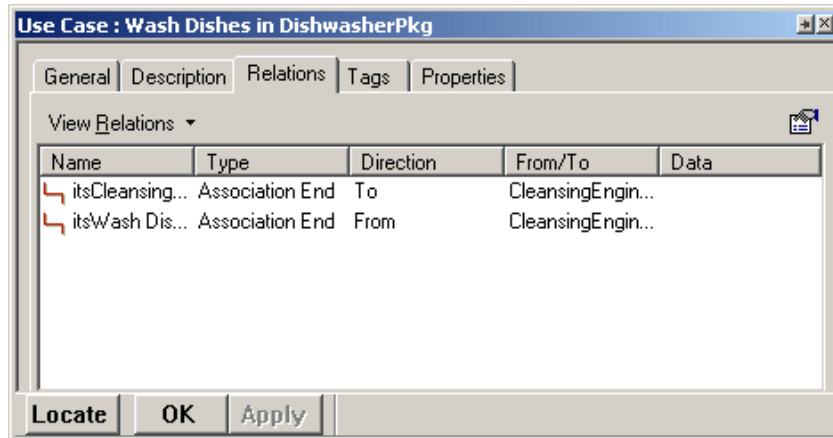
- ◆ **General** typically contains the name of the element and other general options, as shown in the following figure:



- ◆ **Description**, as its title implies and as shown in the following figure, contains the description of the element, if it has been included.

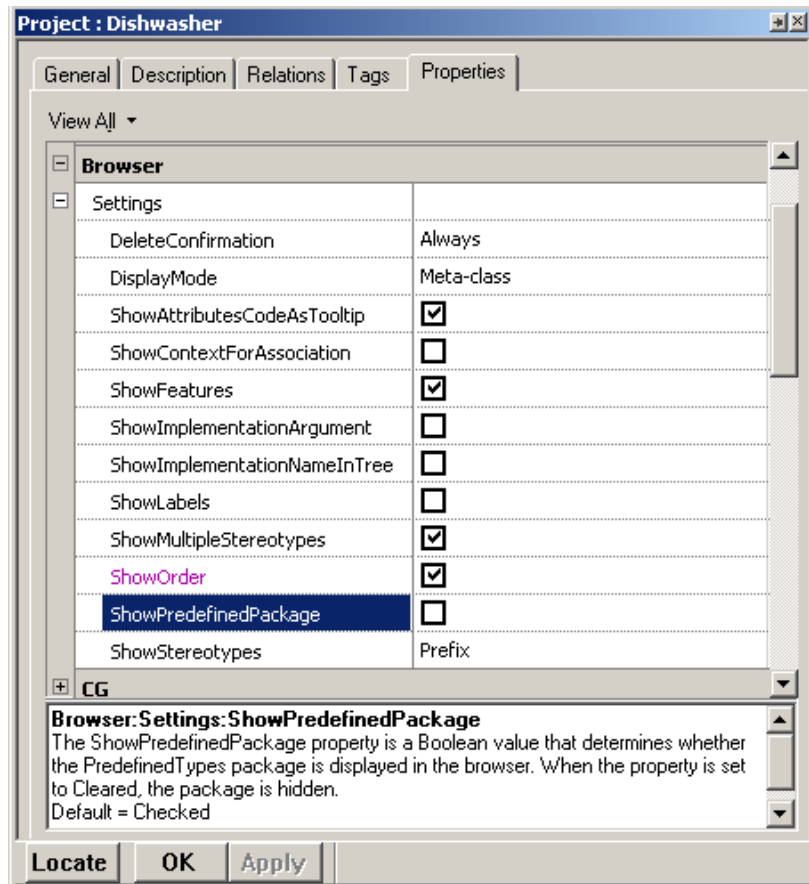


- ◆ **Relations** lists all the relationships (dependencies, associations, and so on) an element is engaged with, as shown in the following figure:



- ◆ **Tags** lists any tags available for an element. *Tags* enable you to add information to certain kinds of elements to reflect characteristics of the specific domain or platform for the modeled system. Refer to the *IBM Rational Rhapsody User Guide* for more information about tags.
- ◆ **Properties** lists the properties associated with the Rational Rhapsody element.
  - The top left column on this tab shows the metaclass and property (for example, **Settings** and **ShowPredefinedPackage**).
  - The top right column shows the default for the selected property, if there is one (for example, **Cleared**).

- The box at the bottom portion of the **Properties** tab shows the definition for the property selected in the upper left column of the tab. The definition display shows the names of the subject, metaclass, property, and the definition for the property, as shown in the following figure:



**Note:** Rational Rhapsody documentation uses a notation method with double colons to identify the location of a specific property. For example, for the property in the above figure, the location is `Browser::Settings::ShowPredefinedPackage` where `Browser` is the subject, `Settings` is the metaclass, and `ShowPredefinedPackage` is the property.

## Moving the Features Dialog Box

The Features dialog box is a floating window that can be positioned anywhere on the screen, or docked to the Rational Rhapsody GUI.

To dock the Features dialog box in the Rational Rhapsody window, do one of the following:

- ◆ Double-click the title bar. The dialog box docks. You can now drag it to another location if you want.
- ◆ Right-click the title bar and select **Docking by Drag**. Then drag the dialog box to another location.

To undock the Features dialog box, do one of the following:

- ◆ Double-click the title bar to undock it.
- ◆ Right-click the title bar and clear **Docking by Drag**, and then drag the dialog box to another location.

**Note:** An asterisk (\*) in a title bar for the Rational Rhapsody window and any dialog box means that data has been modified and a save has not been done yet.

If the Rational Rhapsody browser does not display, select **View > Browser**.

## Summary

In this section, you became familiar with the Rational Rhapsody product and its features. You performed the following:

- ◆ Created the Dishwasher project
- ◆ Saved the project

You are now ready to proceed to the next section where you are going to create the Dishwasher model. You are going to model the requirements of the dishwasher and the functions of using a dishwasher using use case diagrams.





# Lesson 1: Creating a Use Case Diagram

---

*Use case diagrams* (UCDs) show the main functions of the system (use cases) and the entities that are outside the system (actors). Use case diagrams allow you to specify the requirements for the system and show the interactions between the system and external actors.

## Note

You must complete all the tasks in [Setting up for the Tutorial](#) in the [Getting Started](#) section before you start this lesson.

## Goals for this Lesson

In this lesson, you are going to determine who are the users of the system and what are the requirements for the embedded system. Then you are going to create the Dishwasher use case diagram.

## Exercise 1: Analyzing the Dishwasher System

Before using Rational Rhapsody, you should determine the requirements for the embedded system. To analyze the dishwasher system used in this tutorial, answer these questions:

- ◆ Who might use the system?
- ◆ How they might use it?
- ◆ What are the major actions of the system?
- ◆ When do these actions occur?
- ◆ What are the relationships, similarities, or differences between the actions?
- ◆ What is standard behavior?
- ◆ What can go wrong?

Some simplified answers to these questions might be as follows:

- ◆ The system users or “actors” would include a “cleansing engineer” and a “service person.”
- ◆ The system washes, rinses, and then dries dishes.

## Lesson 1: Creating a Use Case Diagram

---

- ◆ The “cleansing engineer” loads the dishes into the dishwasher, starts the dishwasher, and removes dishes after they are washed.
- ◆ The system might fail to wash, rinse, or dry the dishes and require service.

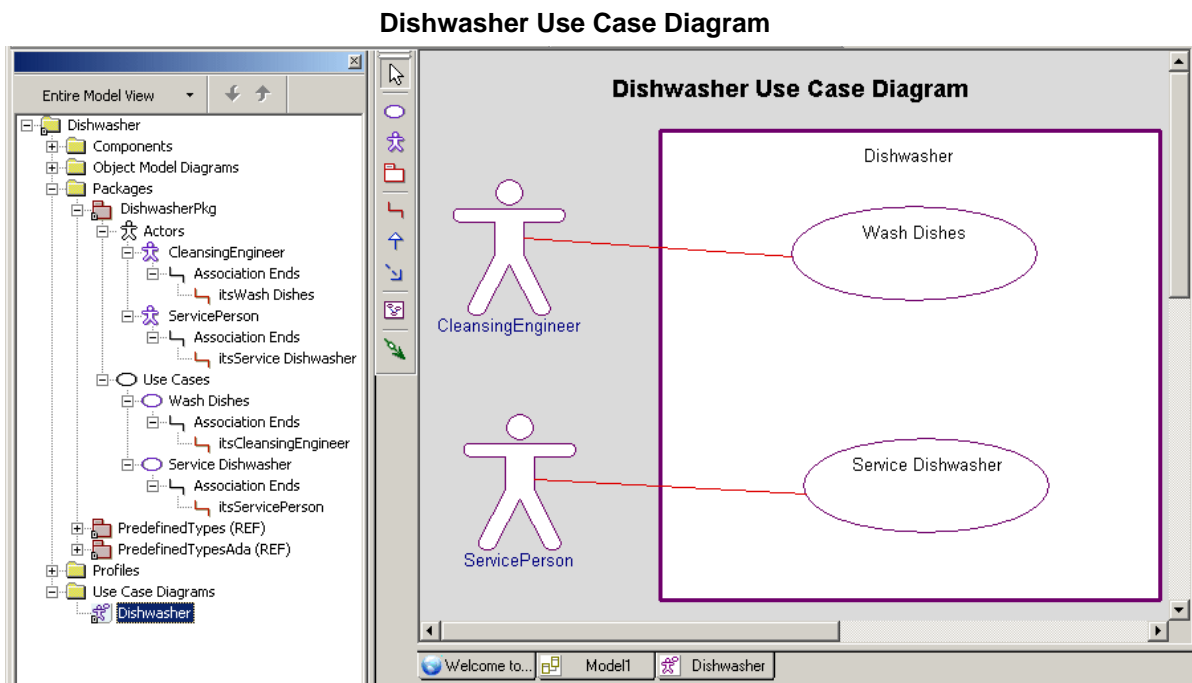
During this analysis phase, you identify actors for the system. The three types of actors are as follows:

- ◆ Users of the system
- ◆ External component providing information to the system
- ◆ External component receiving information from the system

## Exercise 2: Creating the Dishwasher Use Case Diagram

In this exercise you are going to create the Dishwasher use case diagram. A use case diagram shows typical interactions between the system being designed and the external actors who might interact with it.

The following figure shows the Dishwasher use case diagram that you are going to create in this exercise.

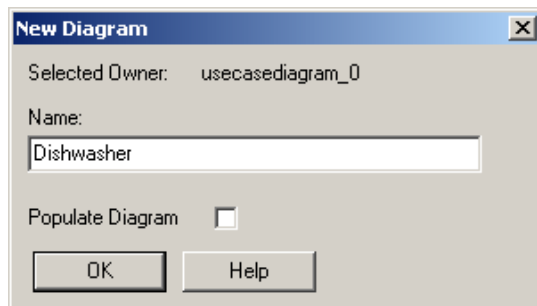


## Task 2a: Creating the Dishwasher Use Case Diagram

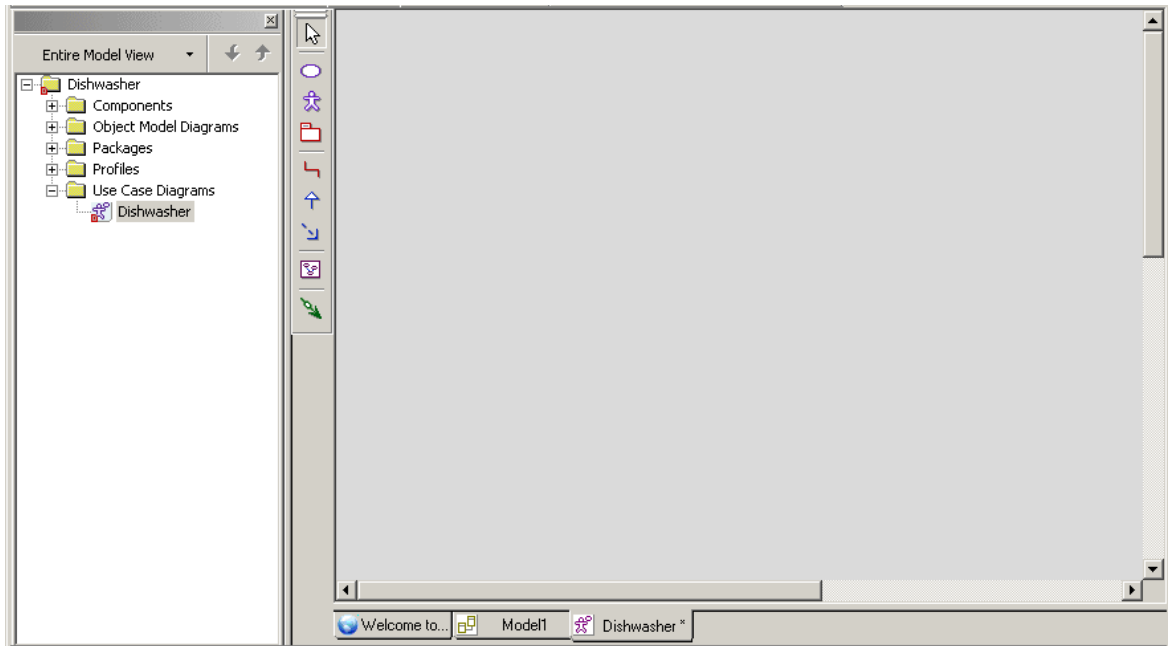
This task is the starting point for the design.

To create the Dishwasher use case diagram, follow these steps:

1. Start Rational Rhapsody in Ada and open the Dishwasher model if they are not already open.
2. Right-click the top-level **Dishwasher** in the Rational Rhapsody browser, and select **Add New > Use Case Diagram** to open the New Diagram dialog box.
3. Type `Dishwasher`, as shown in the following figure, and then click **OK**.



Rational Rhapsody automatically adds the **Use Case Diagrams** category and the name of the new diagram to the Rational Rhapsody browser and opens the new diagram in the drawing area, as shown in the following figure:





### Note

You can also create a diagram by using the Tools menu or the **Diagrams** toolbar. Also, once you create a diagram you can open it using the **Diagrams** toolbar. Refer to the *IBM Rational Rhapsody User Guide* for more information.


## Task 2b: Drawing the Boundary Box and Actors

The boundary box delineates the system under design from the external actors. Use cases are inside the boundary box; actors are outside the boundary box. In this task, you are going to draw the boundary box and actors using the [Dishwasher Use Case Diagram](#) figure as a reference.

To draw the boundary box and actors, follow these steps:

1. Click the **Create Boundary box** button  on the **Drawing** toolbar.
2. Click the drawing area and drag to create a boundary box. Rational Rhapsody creates a boundary box named `System Boundary Box`.
3. Rename the boundary box `Dishwasher` and then press **Enter**.
4. Click the **Create Actor** button  on the **Drawing** toolbar.
5. On the drawing area, click to the left side of the boundary box. Rational Rhapsody creates an actor with a default name of `actor_n`, where *n* is greater than or equal to 0.
6. Rename the actor `CleansingEngineer` and then press **Enter**.

**Note:** Because code can be generated using the specified names, do not include spaces in the names of actors.

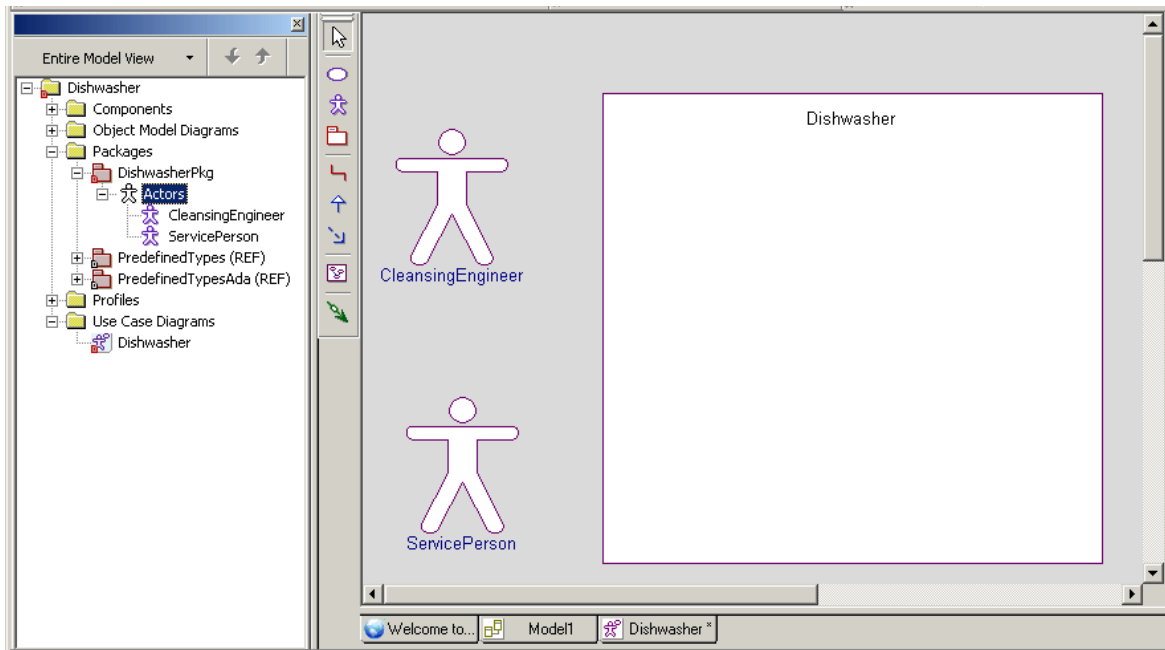
7. Draw another actor named `ServicePerson`.
8. Use the tools on the **Layout** toolbar to help you with the layout of selected elements (including labels) in your diagram. For example, to adjust the size of your actors, do the following:
  - a. Select an actor and then use the Sizing handles to adjust the size of the actor.
  - b. Select the other (original size) actor.
  - c. Use **Shift+Click** to select the size-adjusted actor and then click **Same Size**  on the **Layout** toolbar to resize the actors so that they are the same size. The last element selected is used as the default.

Refer to the *IBM Rational Rhapsody User Guide* for more information about the **Layout** toolbar.

**Note:** If you want to move a drawn element on a drawing more precisely than clicking and dragging it, click one or more elements, press the **Ctrl** key and use the standalone directional arrow keys to move your element(s). You can also use the directional arrows on the numeric keypad with NumLock not active.

9. In the browser, expand the **DishwasherPkg** package and the **Actors** category to view your newly created actors, as shown in the following figure:

**Note:** You created the **DishwasherPkg** package in [Renaming the Default Package](#).



**Note:** To quickly find the **Actors** category in the Rational Rhapsody browser, right-click an actor on the use case diagram and click **Locate** or press **Ctrl+L**. You can use this technique with other objects on a diagram.

### Task 2c: Drawing the Use Cases


During the analysis phase, you identified user-visible functions or important goals of the system. These are *use cases*. A *use case* represents a particular function of the system. In this task, you are going to draw the following use cases using the [Dishwasher Use Case Diagram](#) figure as a reference.

- ◆ Wash Dishes
- ◆ Service Dishwasher

## Lesson 1: Creating a Use Case Diagram

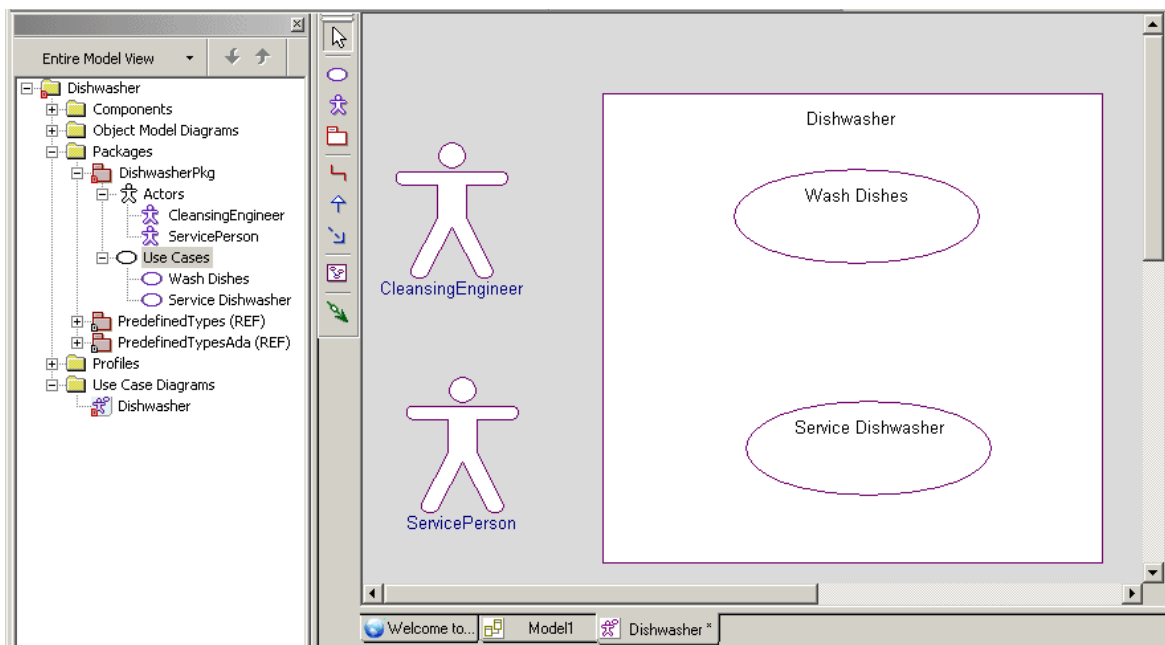
---

To draw the use cases, follow these steps:

1. Click the **Create Use Case** button  on the **Drawing** toolbar.
2. Click inside the top half of the boundary box. Rational Rhapsody creates a use case with a default name of `usecase_n`, where `n` is equal to or greater than 0.
3. Rename the use case `Wash Dishes` and then press **Enter**.

**Note:** For use case names, you can use spaces because use case names do not correspond to actual generated code. In the previous task where you drew actors, you did not use spaces in actor names because code can be generated using the specified actor names.

4. Create another use case inside the boundary box named `Service Dishwasher`.
5. In the browser, under the **DishwasherPkg** package, expand the new **Use Cases** category to view the use cases you created, as shown in the following figure:






## Task 2d: Associating Actors with Use Cases

The **CleansingEngineer** washes dishes and configures the washing mode, while the **ServicePerson** only services the dishwasher as needed.

To incorporate the relationships of the actors to the use cases into the design, you draw association lines between the actors and use cases. An *association* represents a connection between objects or users. In this task, you associate actors with use cases using the [Dishwasher Use Case Diagram](#) figure as a reference.

To draw association lines, follow these steps:

1. Click the **Create Association** button  on the **Drawing** toolbar.  
Notice that once you move your cursor over the drawing area the mouse pointer turns into a crosshairs pointer to signify that it is enabled and that it changes into a circled crosshairs pointer when drawing is possible.
2. Click the edge of the **CleansingEngineer** actor and then click the edge of the **Wash Dishes** use case. Rational Rhapsody creates an association line with the name label highlighted. You do not need to name this association, so click the mouse button again (this is the same as pressing **Enter**).

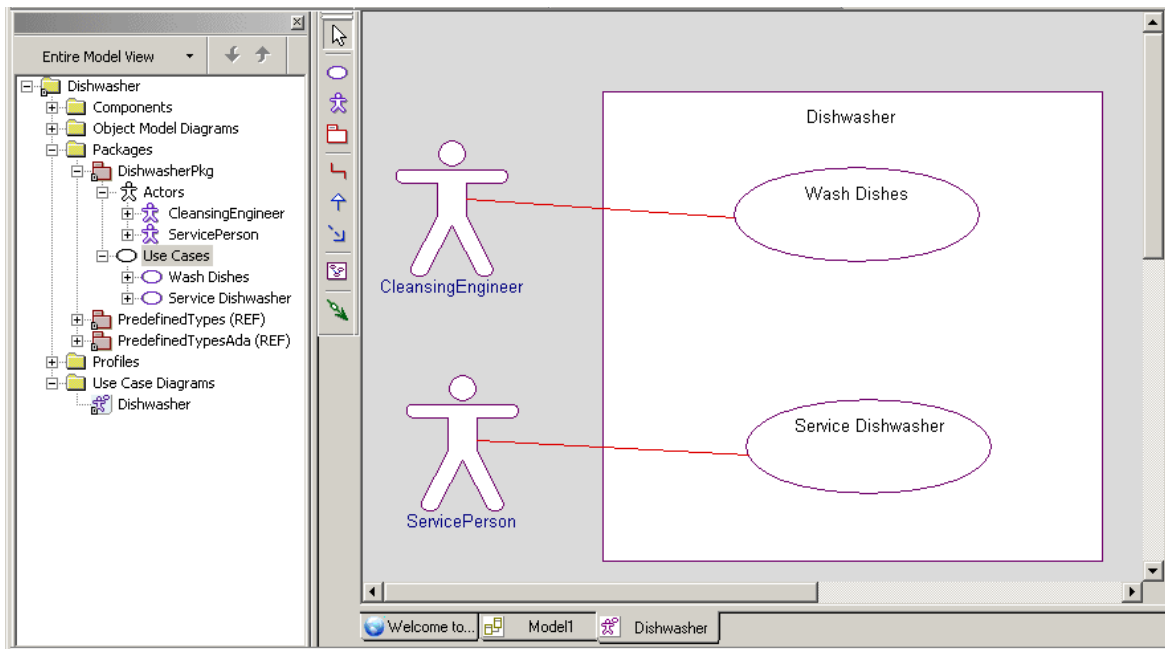
**Note:** To keep a line straight as you draw it, press the **Ctrl** key as you are drawing the line.

3. Create an association between the **ServicePerson** actor and the **Service Dishwasher** use case, and then click the mouse button again or press **Enter**.

## Lesson 1: Creating a Use Case Diagram

4. Click the **Save** button  to save your model.

Your use case diagram should resemble the following figure:




### Task 2e: Adding a Diagram Title

Each diagram has its name in the diagram table and in the title bar of the window that displays the diagram. However, it is also useful to add a title onto the diagram itself to help other members of your team understand the content and purpose of a diagram.

To add an optional title to your diagram, follow these steps:

1. With the diagram displayed in the drawing area, click **A** on the **Free Shapes** toolbar.
2. Click above the system boundary box in the diagram and type, for example, `Dishwasher Use Case Diagram`, and press **Ctrl+Enter**.

**Note:** If you press **Enter**, you move your cursor to a new line. In this case, to exit typing mode, you have to press **Ctrl+Enter** to end your action. Or you can click out of the typing area.

3. Make the following changes if you want:
  - a. Reposition the title by dragging it into another location.
  - b. Use the tools on the **Format** toolbar to change the font styles.
4. Click the **Save** button  to save your model.

For more information about the **Free Shapes** and **Format** toolbars, refer to the *IBM Rational Rhapsody User Guide*.

You have completed drawing the Dishwasher use case diagram. It should resemble the [Dishwasher Use Case Diagram](#) figure.

## Summary

In this lesson, you determined who are the users of the system and what are the requirements for the embedded system. Then you created a use case diagram that shows the functions and requirements of the dishwasher. You became familiar with the parts of a use case diagram and created the following:

- ◆ System boundary box
- ◆ Actors
- ◆ Use cases
- ◆ Association lines
- ◆ Title for your diagram

You are now ready to proceed to the next lesson, where you are going to define how the system components are interconnected using an object model diagram.



# Lesson 2: Creating an Object Model Diagram

---

*Object model diagrams* (OMDs) specify the types of objects in the system, the attributes and operations that belong to those objects, the static relationship that can exist between classes (types), and the constraints that might apply. The Rational Rhapsody code generator directly translates the elements and relationships modeled in OMDs into Ada source code.

## Goals for this Lesson

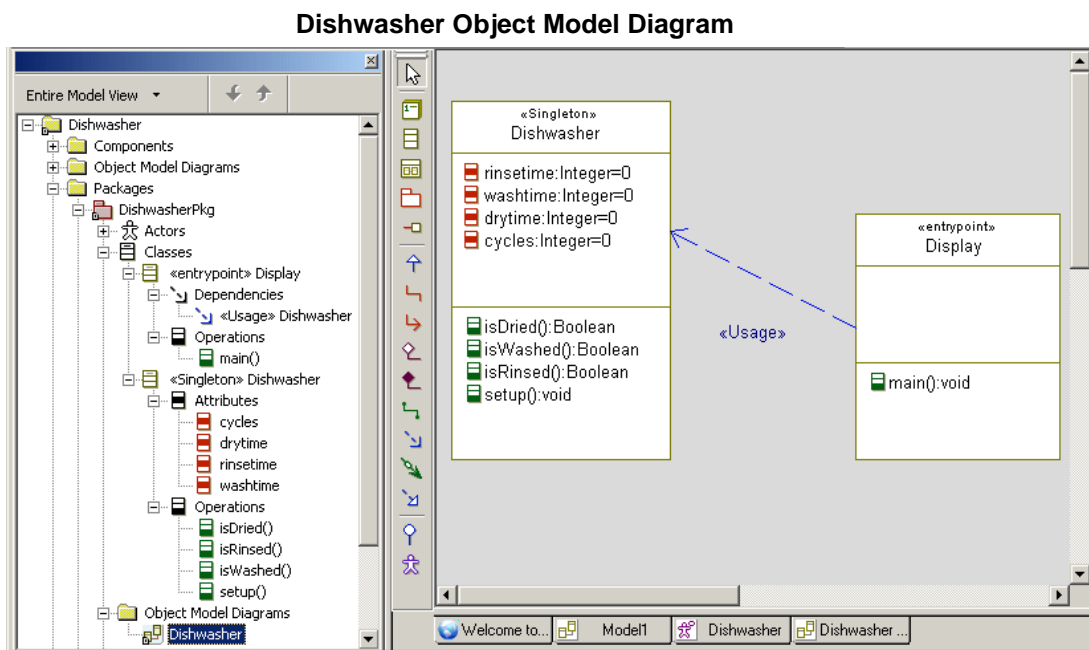
In this lesson, you are going to create the Dishwasher object model diagram. In this lesson, you are going to:

- ◆ Create an object model diagram
- ◆ Create classes in the object model diagram
- ◆ Draw dependencies
- ◆ Specify features of a class
- ◆ Set attributes of a class
- ◆ Add operations to a class

## Exercise 1: Creating the Dishwasher Object Model Diagram

Object model diagrams show the types of objects in the system, the attributes and operations that belong to those objects, and the static relationships that can exist between classes (types).

The following figure shows the Dishwasher object model diagram that you are going to create in this exercise.



## Task 1a: Creating the Dishwasher Object Model Diagram

You draw an object model diagram using the following general steps:

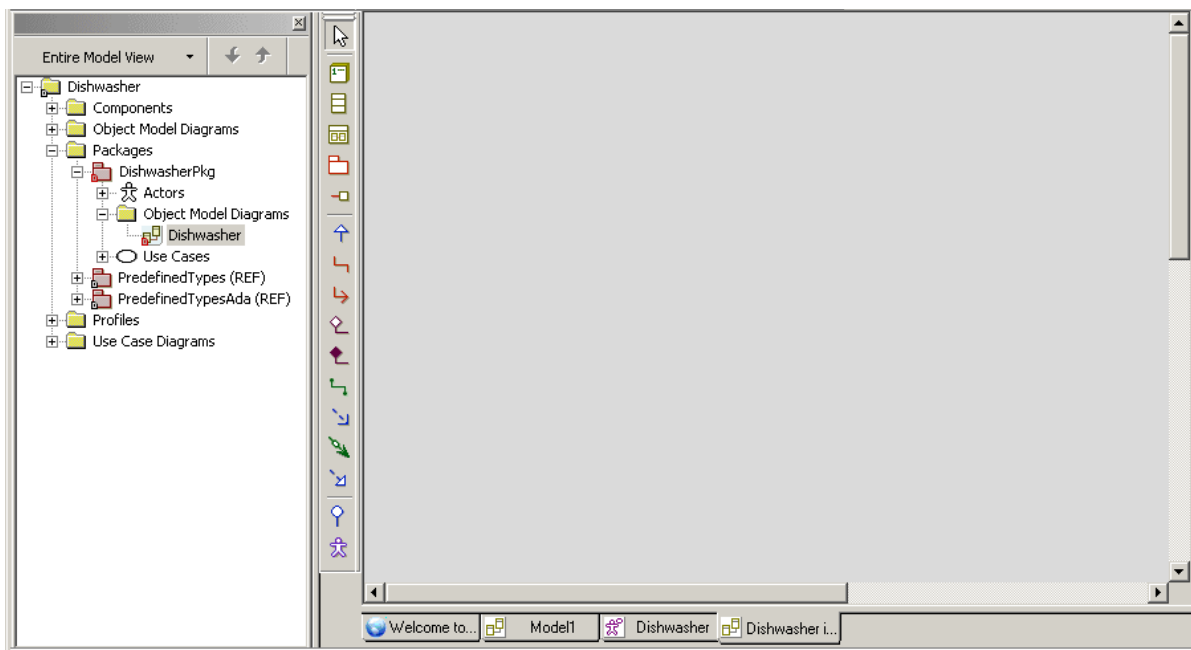
1. Draw objects.
2. Draw links.

The following tasks describe each of these steps in detail.

To create an object model diagram, follow these steps:

1. Start Rational Rhapsody and open the Dishwasher model if they are not already open.
2. In the browser, expand the **Packages** category, right-click the **DishwasherPkg** package, and then select **Add New > Object Model Diagram** to open the New Diagram dialog box.
3. Type `Dishwasher` and then click **OK**.

Rational Rhapsody adds the **Object Model Diagrams** category and the name of the new object model diagram to the browser. Rational Rhapsody also opens the new object model diagram in the drawing area, as shown in the following figure:



## Task 1b: Drawing Classes and Dependencies

In this task, you are going to draw classes for your Dishwasher object model diagram and then draw a dependency between the two classes. In addition, you are going to set a stereotype for that dependency. Use the [Dishwasher Object Model Diagram](#) figure as a reference.

Rational Rhapsody uses classes to represent the major elements of the object model. *Classes* are groupings of similar kinds of objects into types. There are two types of classes represented in the object model diagrams:



- ◆ **Simple (Specification) Class** shows only the class name, without any attributes or operations.
- ◆ **Composite (Structured) Class** contains other classes. The parts come into being and are destroyed with the creation and destruction of the composite class.

All instances of a class have the same attributes and operations, although their individual values can vary. The top compartment holds the name of the class, the middle compartment holds the attributes, and the bottom compartment holds the operations.

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

A *stereotype* is a modeling element that extends the semantics of the UML metamodel by typing UML entities. Rational Rhapsody includes predefined stereotypes, and you can also define your own stereotypes. Stereotypes are enclosed in angle quotes (or guillemets) on diagrams, for example, «Usage».

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

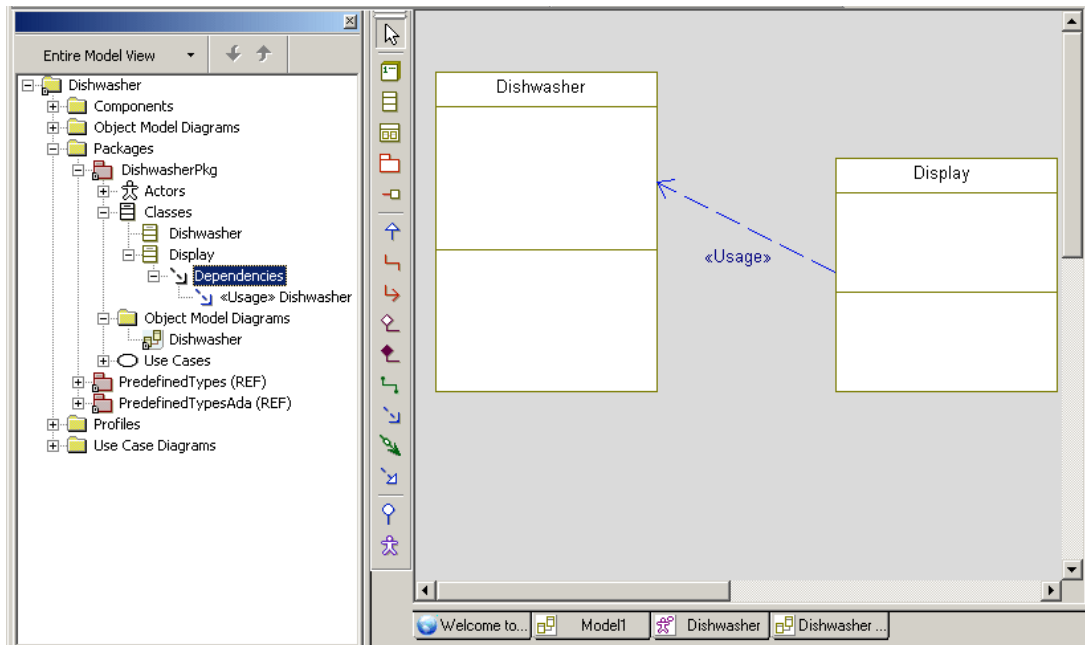
1. Click the **Class** button  on the **Drawing** toolbar.  
Notice that once you move your mouse pointer over the drawing area, a class icon appears along with it.
2. Click-and-drag on the drawing area and create a tall rectangular class.
3. Rename the class `Dishwasher` and then press **Enter**.
4. Create another class beside the **Dishwasher** class and name it `Display`.
5. Click the **Dependency** button  on the **Drawing** toolbar.
6. Click the left edge of the **Display** class and click the right edge of the **Dishwasher** class. This arrow shows the dependency relationship between the **Display** class and the **Dishwasher** class and, therefore, changes the definitions of both classes.
7. Double-click the dependency line or right-click it and select **Features** to open the Features dialog box.



## Exercise 1: Creating the Dishwasher Object Model Diagram

- On the **General** tab, in the **Stereotype** box, select the **Usage in PredefinedTypes** check box from the drop-down menu. (**Usage** appears in the **Stereotype** box after you do so.)
- Click **Apply** to apply your changes and then **OK** to close the dialog box.
- Save your project.

Your object model diagram should resemble the following figure:



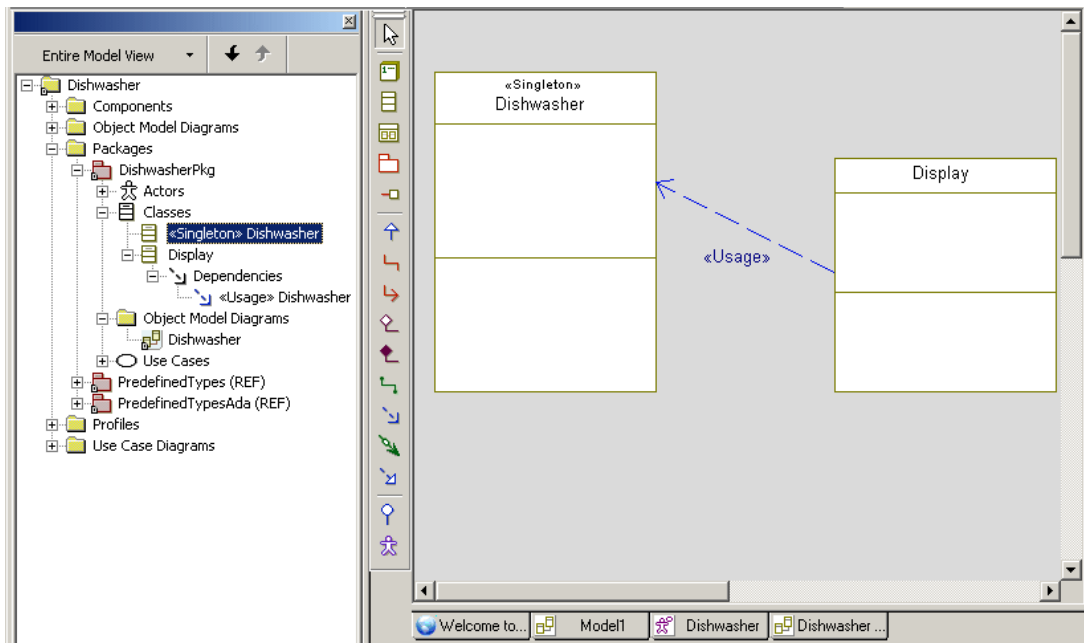
## Task 1c: Creating a Singleton

A *singleton* is a simple pattern or mechanism that creates a single, global instance of a class. In Rational Rhapsody, you can instruct the Ada code generator to create a singleton by creating a Singleton stereotype. Singleton classes are instantiated only once throughout the life of the system.

To create a single global instance of the Dishwasher class, follow these steps:

1. Double-click the **Dishwasher** class in the Rational Rhapsody browser or the diagram or right-click it and select **Features** to open the Features dialog box.
2. On the **General** tab, in the **Stereotype** box, select the **Singleton in PredefinedTypes** check box from the drop-down menu. (**Singleton** appears in the **Stereotype** box after you do so.)
3. Click **Apply** and then **OK**.
4. Save your model.

Your object model diagram should display the Singleton stereotype for your **Dishwasher** class, as shown in the following figure:

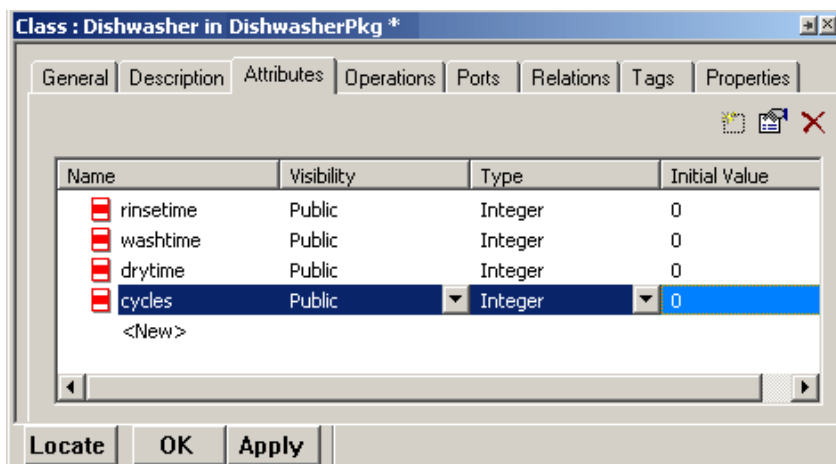


## Task 1d: Adding Attributes

In this task, you add attributes to your **Dishwasher** class. *Attributes* are the data members of a class.

To add attributes describing the dishwasher's behavior, follow these steps:

1. Double-click the **Dishwasher** class in the Rational Rhapsody browser or diagram to open the Features dialog box.
2. On the **Attributes** tab, click <New> to create a blank row for an attribute.
3. To define the `rinsetime` attribute, fill in the row with these values:
  - a. For **Name**, type `rinsetime`.
  - b. For **Visibility**, select **Public** from the drop-down list if necessary. It should be set by default.
  - c. For **Type**, select **Integer** if necessary. It should be set by default.
  - d. For **Initial Value**, type 0 (zero).
4. Repeat steps 2 – 3 to create the remaining attributes, as shown in the following figure. Give them the same **Visibility**, **Type**, and **Initial Value** as you did the first attribute.
  - ◆ `washtime`
  - ◆ `drytime`
  - ◆ `cycles`



5. Click **Apply** to apply your changes.
6. Do not close the Features dialog box. Continue with [Task 1e: Creating Operations](#).

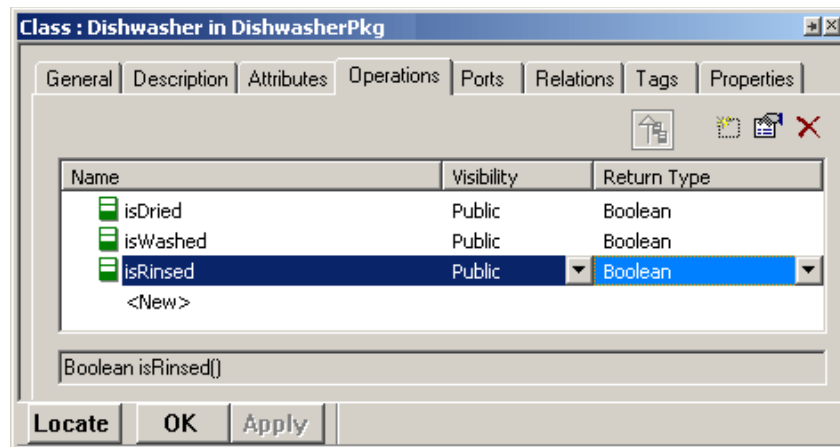
## Task 1e: Creating Operations

In this task, you are going to create operations that signal when a dishwasher action is complete. An *operation* is a service that can be requested from an object to affect behavior. An operation has a signature, which might restrict the actual parameters that are possible.


This task includes typing code for the operation.

To create operations that signal when a dishwasher action is complete, follow these steps:

1. Continuing from the previous task, with the Features dialog box still opened for the **Dishwasher** class, on the **Operations** tab, click **<New>** and select **Primitive Operation** to add a blank row for an operation.
2. To define the `isDried` operation, fill in the row with these values:
  - a. For **Name**, type `isDried`.
  - b. For **Visibility**, select **Public** from the drop-down list if necessary. It should be set by default.
  - c. For **Return Type**, select **Boolean**.
3. Repeat steps 1 – 2 to create the remaining operations, as shown in the following figure. Give them the same **Visibility**, **Type**, and **Return Type** as you did for the first operation.
  - ◆ `isWashed`
  - ◆ `isRinsed`

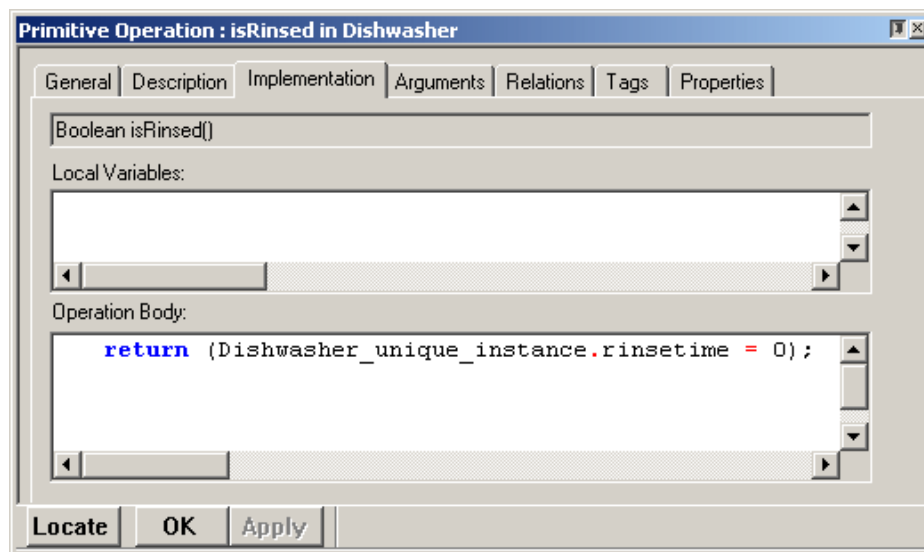


4. Click **Apply** to apply your changes.
5. Highlight the **isRinsed** operation, as shown above.


6. Click  on the upper right corner of the tab (this is the Invoke Features Dialog button) to open the Features dialog box for the operation.
7. On the **Implementation** tab, type the following code in the **Operation Body** box, which is also shown in the following figure:

```
return (Dishwasher_unique_instance.rinsetime = 0);
```

**Note:** The `unique_instance` naming convention is used because we made the Dishwasher class a singleton. The Rational Rhapsody code generator generates the attributes based on this naming convention.



**Note:** You can copy the code you just entered so that you can use it again for the other operations in this task. Just be sure to change the operation name after each time you paste it.

8. Click **OK** to save your changes and return to the list of operations you entered previously.
9. Double-click the Operation icon  to the left of the **isDried** operation to open the Features dialog box for this operation.
10. On the **Implementation** tab, type the following code in the **Operation Body** box:
 

```
return (Dishwasher_unique_instance.drytime = 0);
```
11. Click **OK**.
12. Open the Features dialog box for the **isWashed** operation.

13. On the **Implementation** tab, type the following code in the **Operation Body** box:

```
return (Dishwasher_unique_instance.washtime = 0);
```

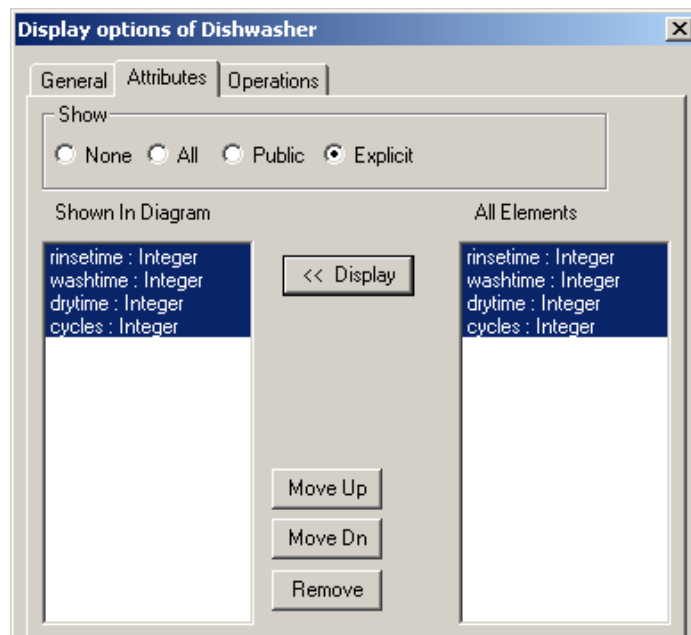
14. Click **OK** to apply your changes and close the Features dialog box for the operation.
15. Click **OK** to close the Features dialog box for the class.

## Task 1f: Displaying Attributes and Operations in the OMD

To display the attributes and operations defined for the **Dishwasher** class on the Dishwasher object model diagram, follow these steps:

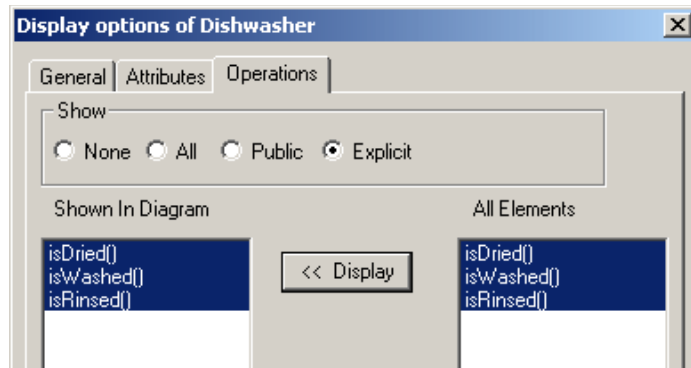
1. On the **Dishwasher** object model diagram, right-click the **Dishwasher** class and then select **Display Options** to open the Display Options dialog box.
2. On the **Attributes** tab, select the **Explicit** option button if it is not already selected.
3. Highlight all of the attributes listed in the **All Elements** box and then click the **Display** button.

This moves all of the attributes into the **Shown in Diagram** box, as shown in the following figure:



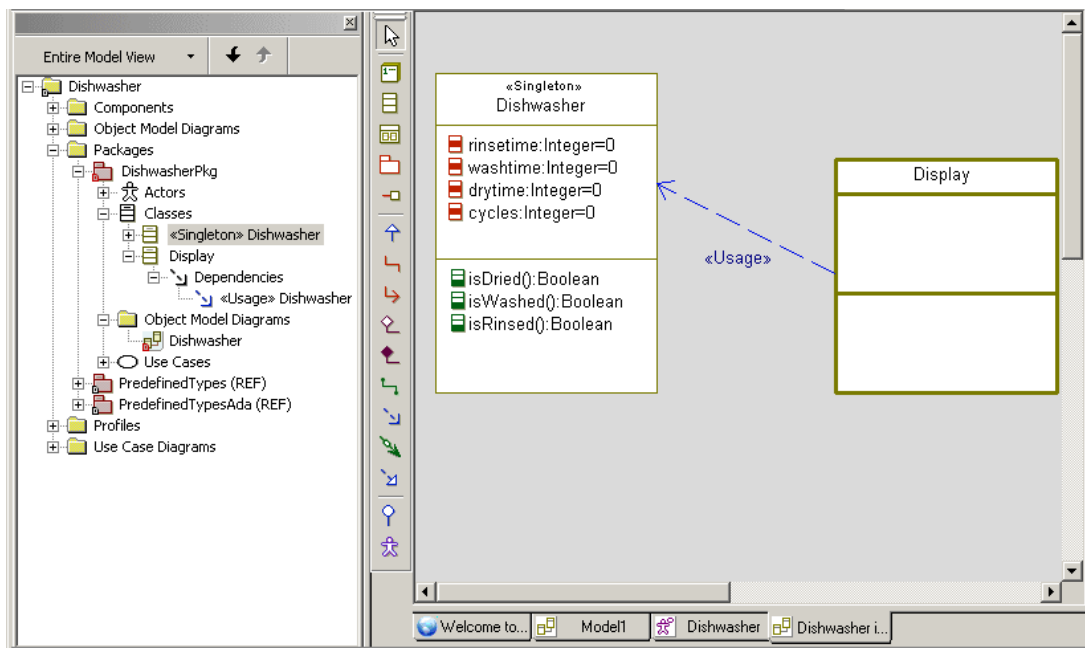
**Note:** If you select the **All** option button, the notations would be included in the attributes and operations display.

4. On the **Operations** tab, use the same method as described above to move all the operations from the **All Elements** box to the **Shown in Diagram** box, as shown in the following figure:



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

Your object model diagram should resemble the following figure:

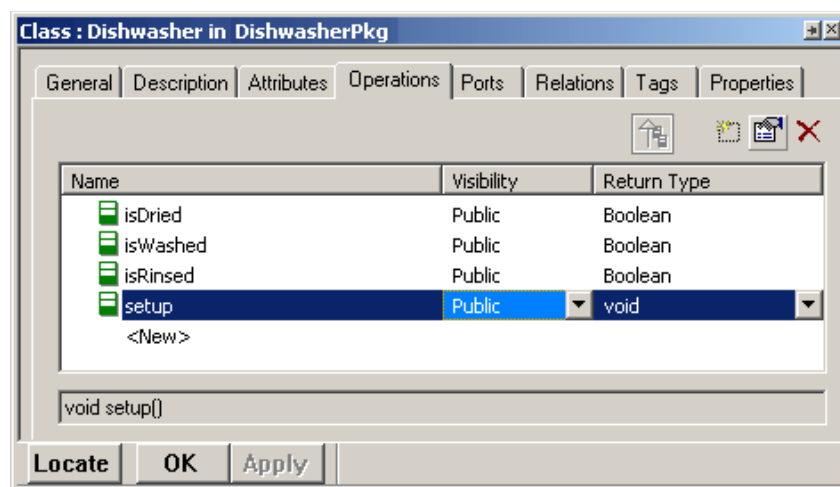




6. Save your model.

## Task 1g: Adding the setup Operation

To add a `setup` operation for use during system installation, follow these steps:

1. Double-click the **Dishwasher** class in the Rational Rhapsody browser or the diagram to open the Features dialog box for the class.
2. On the **Operations** tab, click **<New>** and select **Primitive Operation** to add a blank row for an operation.
3. Type `setup` as the new operation name and accept all of the default settings, as shown in the following figure:

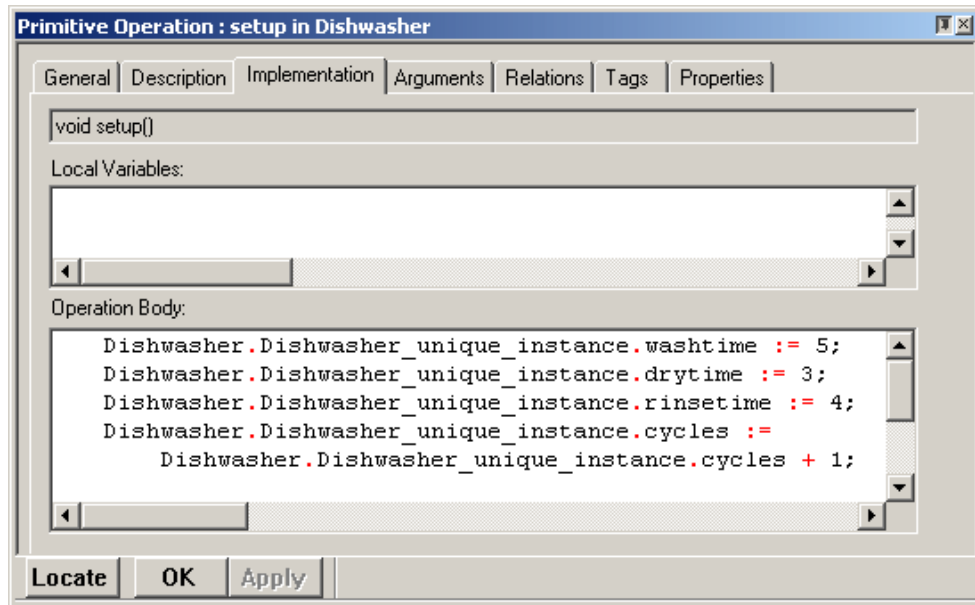


4. Click **Apply**.
5. Open the Features dialog box for the **setup** operation:
  - Double-click the icon  to the left of the operation, or
  - Highlight the operation and click the Invoke Features Dialog button .



6. On the **Implementation** tab, type the following code in the **Operation Body** box:

```
Dishwasher.Dishwasher_unique_instance.washtime := 5;  
Dishwasher.Dishwasher_unique_instance.drytime := 3;  
Dishwasher.Dishwasher_unique_instance.rinsetime := 4;  
Dishwasher.Dishwasher_unique_instance.cycles :=  
    Dishwasher.Dishwasher_unique_instance.cycles + 1;
```

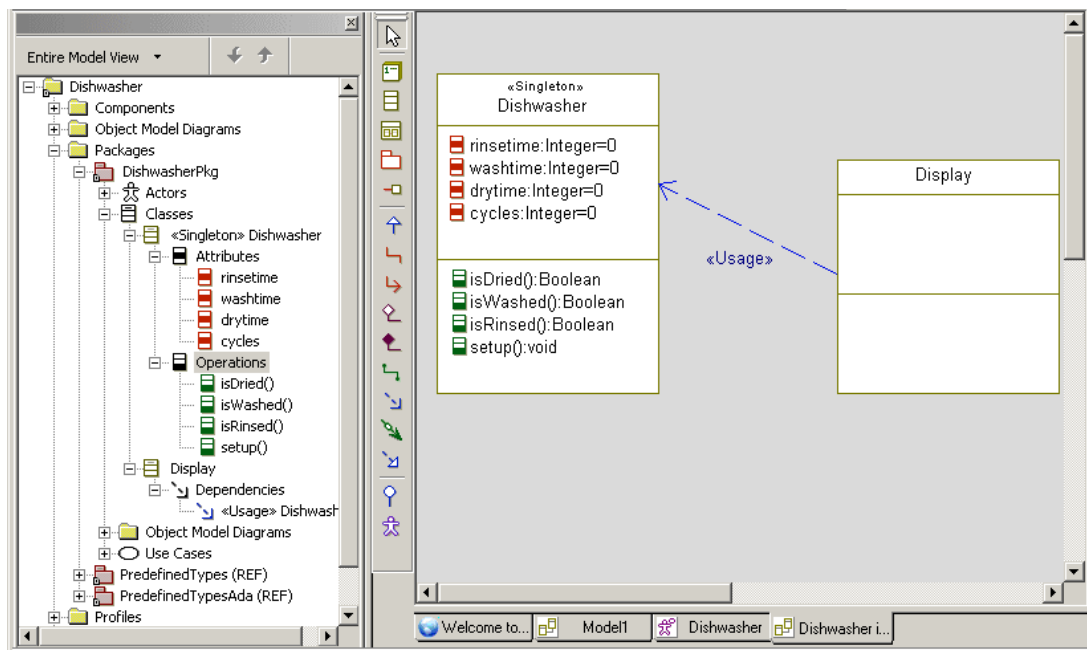


7. Click **OK** to close Features dialog box for the operation.
8. Click **OK** to apply your changes and close the Features dialog box for the class.

## Lesson 2: Creating an Object Model Diagram

- Use the method described in [Task 1f: Displaying Attributes and Operations in the OMD](#) to display the **setup** operation for the **Dishwasher** class.
- Save your model.

You Dishwasher object model diagram should resemble the following figure:



### Task 1h: Adding a main Operation to the Display Class

In this task you are going to create the `main` operation that will serve as the entry point procedure in the Ada executable.

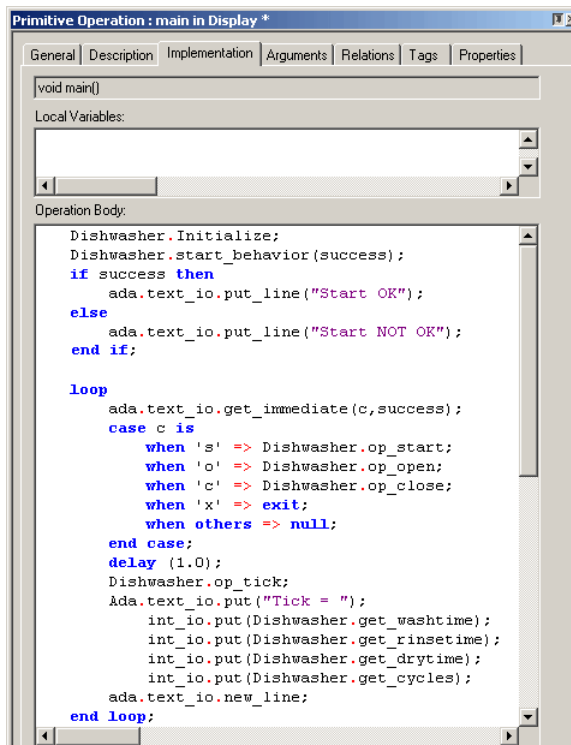
To add the `main` operation, follow these steps:

- Double-click the **Display** class in the Rational Rhapsody browser or the diagram to open the Features dialog box.
- On the **Operations** tab, click **<New>** and select **Primitive Operation** to add a blank row for an operation.
- Type `main` as the new operation name and accept all of the default settings.
- Open the Features dialog box for the operation.

5. On the **Implementation** tab, type the following code in the **Operation Body** area:

```
Dishwasher.Initialize;
Dishwasher.start_behavior(success);
if success then
  ada.text_io.put_line("Start OK");
else
  ada.text_io.put_line("Start NOT OK");
end if;

loop
  ada.text_io.get_immediate(c,success);
  case c is
    when 's' => Dishwasher.op_start;
    when 'o' => Dishwasher.op_open;
    when 'c' => Dishwasher.op_close;
    when 'x' => exit;
    when others => null;
  end case;
  delay (1.0);
  Dishwasher.op_tick;
  Ada.text_io.put("Tick = ");
  int_io.put(Dishwasher.get_washtime);
  int_io.put(Dishwasher.get_rinsetime);
  int_io.put(Dishwasher.get_drytime);
  int_io.put(Dishwasher.get_cycles);
  ada.text_io.new_line;
end loop;
```



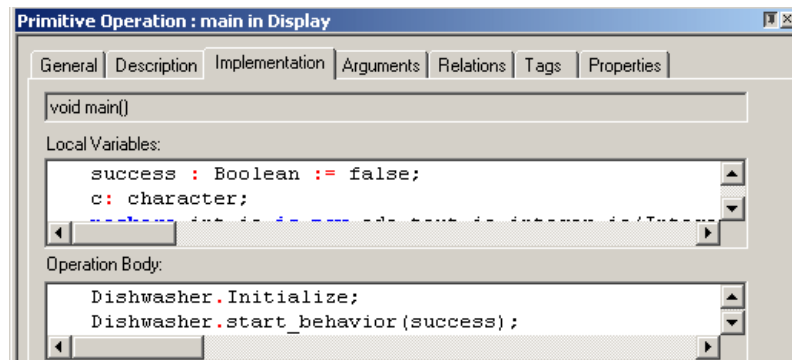
6. Click **Apply**.

## Lesson 2: Creating an Object Model Diagram

---

- Specify variables that you want to appear in the declaration of the endpoint or operation. Type the following code in the **Local Variables** box on the **Implementation** tab:

```
success : Boolean := false;
c: character;
package int_io is new ada.text_io.integer_io(Integer);
use int_io;
```



- Click **OK** to close the Features dialog box.
- Use the method described in [Task 1f: Displaying Attributes and Operations in the OMD](#) to display the **main** operation for the **Display** class.
- Save your model.

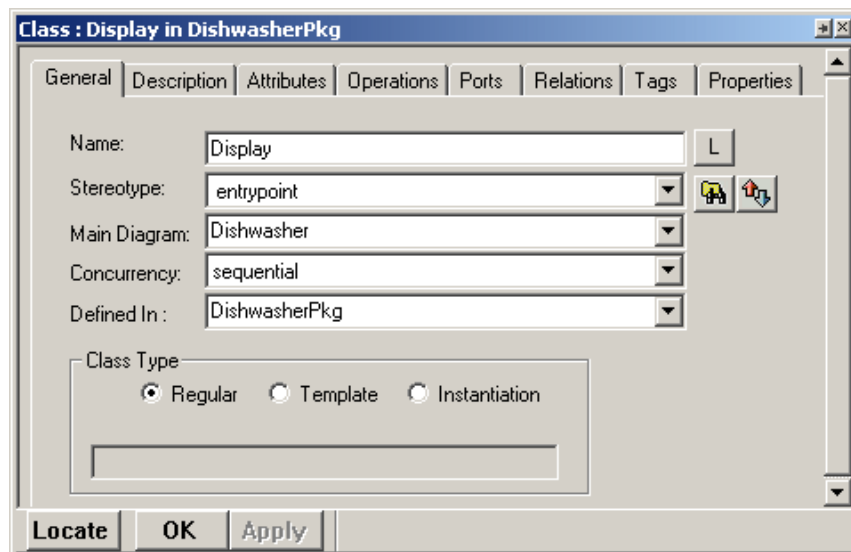
## Task 1i: Using the Entrypoint Stereotype

For the **Dishwasher** model, you want to use the «**entrypoint**» stereotype to direct the code generator and make the `main` Ada operation into the application entry point.

To use the «**entrypoint**» stereotype, follow these steps:

1. Double-click the **Display** class in the Rational Rhapsody browser or on your object model diagram to open the Features dialog box.
2. On the **General** tab, in the **Stereotype** box, select the **entrypoint** in **PredefinedTypesAda** check box from the drop-down menu. (**entrypoint** appears in the **Stereotype** box after you do so.)
3. Click **Apply**.

Your **General** tab should resemble the following figure:

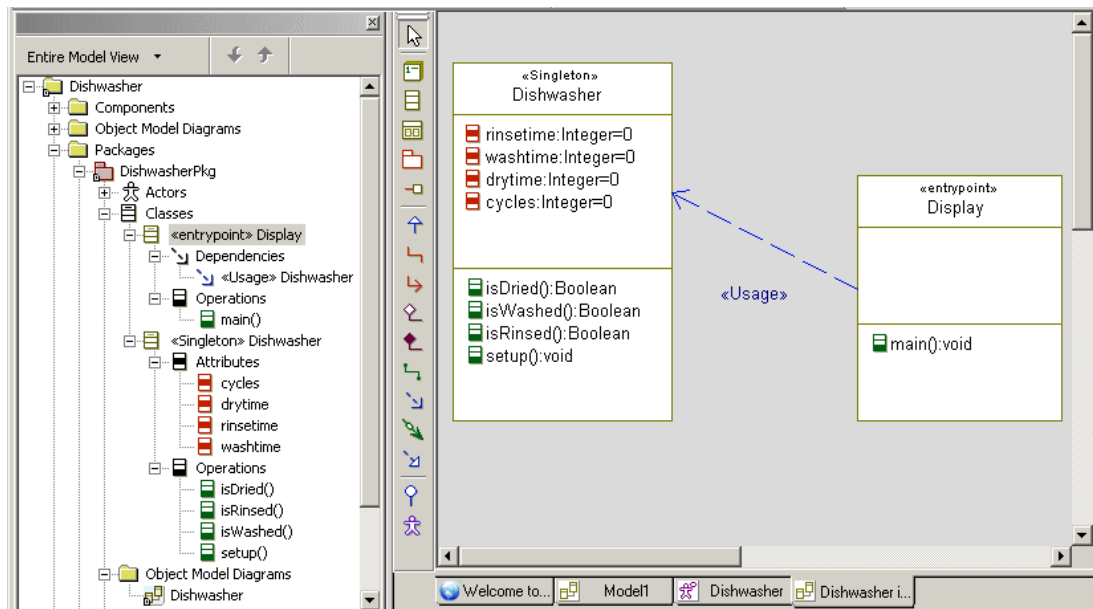


4. Click **OK**.

## Lesson 2: Creating an Object Model Diagram

### 5. Save your model.

Your Dishwasher object model diagram should resemble the following figure:



## Exercise 2: Other Necessary Tasks

In this exercise you perform some other tasks that are necessary for the Dishwasher model before you proceed further.

### Task 2a: Saving Packages Separately

To assist with configuration management and improve project organization, you might want to store packages in separate subfolders within a parent folder. Rational Rhapsody has two directory schemes: flat and hierarchical.

- ◆ In *flat* mode, all package files are stored in the project directory, regardless of their location in the project hierarchy.
- ◆ In *hierarchical* mode, a package is stored in a subdirectory one level below its parent. It is possible to have a hybrid project, where some packages are stored in flat mode, and others are organized in a hierarchy of folders.

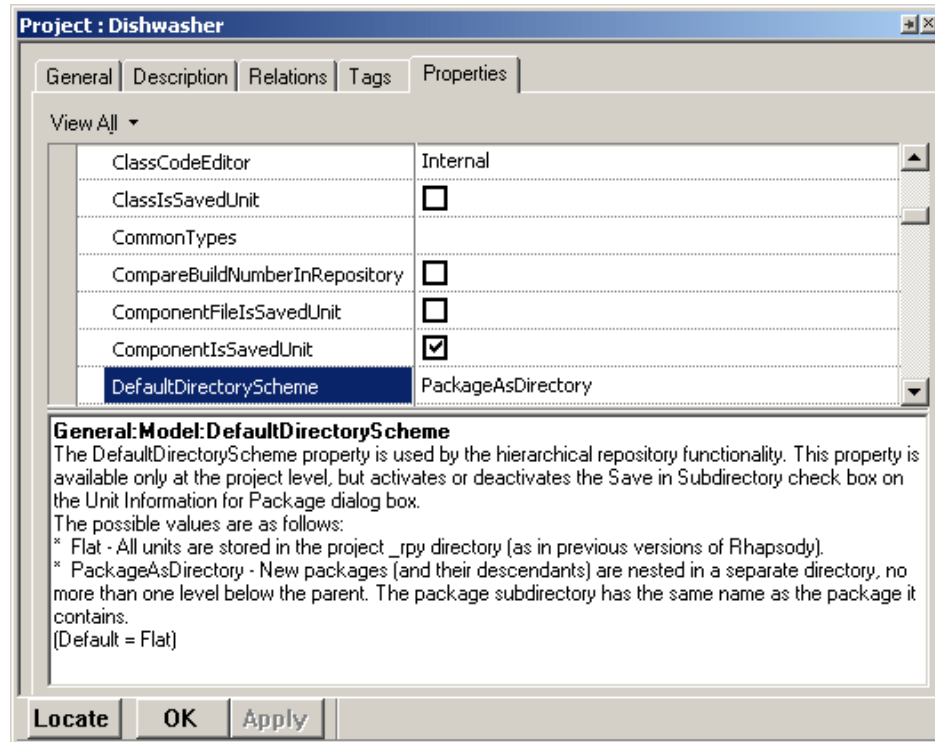
To change the directory scheme so new packages are stored in separate folders by default, follow these steps:

1. Double-click the top-level **Dishwasher** in the Rational Rhapsody browser hierarchy to open the Features dialog box.
2. On the **Properties** tab, select **All** from the drop-down list in the upper-left corner of the dialog box. (The label appears as **View All** after you make the selection.)
3. Expand the **General** subject and the **Model** metaclass, and then highlight the **DefaultDirectoryScheme** property.

**Note:** Rational Rhapsody documentation uses a notation method with double colons to identify the location of a specific property, for example, `General::Model::DefaultDirectoryScheme`. In this example, **General** is the name of the subject, **Model** is the name of the metaclass, and **DefaultDirectoryScheme** is the name of the property.

Refer to the *IBM Rational Rhapsody User Guide* for more information on setting properties. (Do a search of the user guide PDF file for “properties tab.”)

4. Click the box next to **DefaultDirectoryScheme** and use the drop-down list to change the value to **PackageAsDirectory**, as shown in the following figure:



5. Click **OK**.
6. Save your model.

### Task 2b: Using Predefined Packages

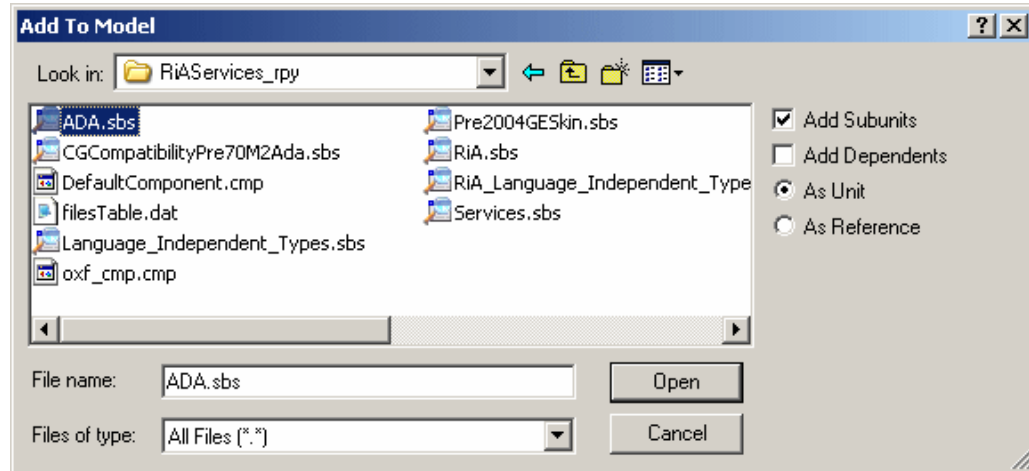
Ada developers use a standard library to input/output text. A stub model of this library is available from Ada.sbs package in the behavioral services model of Rational Rhapsody in Ada. To refine this stub library, the following instructions add the standard `Text_IO` package.

To add this set of predefined standard packages to your model, follow these steps:

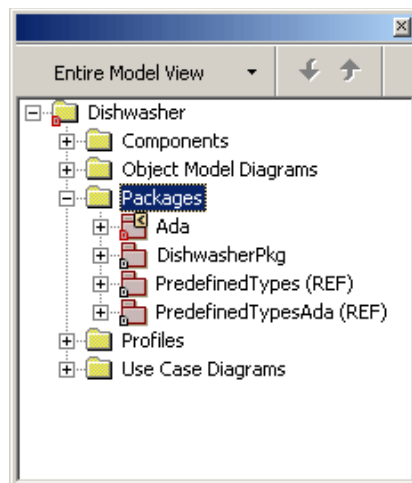
1. Select **File > Add to Model** to open the Add To Model dialog box.
2. Navigate to this path:  
`<Rational Rhapsody installation>\Share\LangAda83\model\RIAServices_rpy.`



3. In the **Files of type** box, select **All Files (\*.\*)** and then select the **ADA.sbs** file, as shown in the following figure:



4. Accept the default settings and click **Open**. The ADA package is added to the browser, as shown in the following figure:



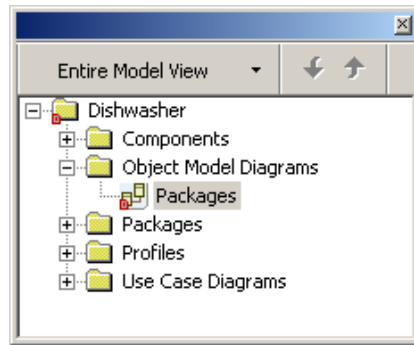
5. In the Rational Rhapsody browser, expand the **Object Model Diagrams** category (directly below the **Components** category on the Rational Rhapsody browser).
6. Right-click **Modell1** (the default object model diagram) and select **Features** to open the Features dialog box.

## Lesson 2: Creating an Object Model Diagram

---

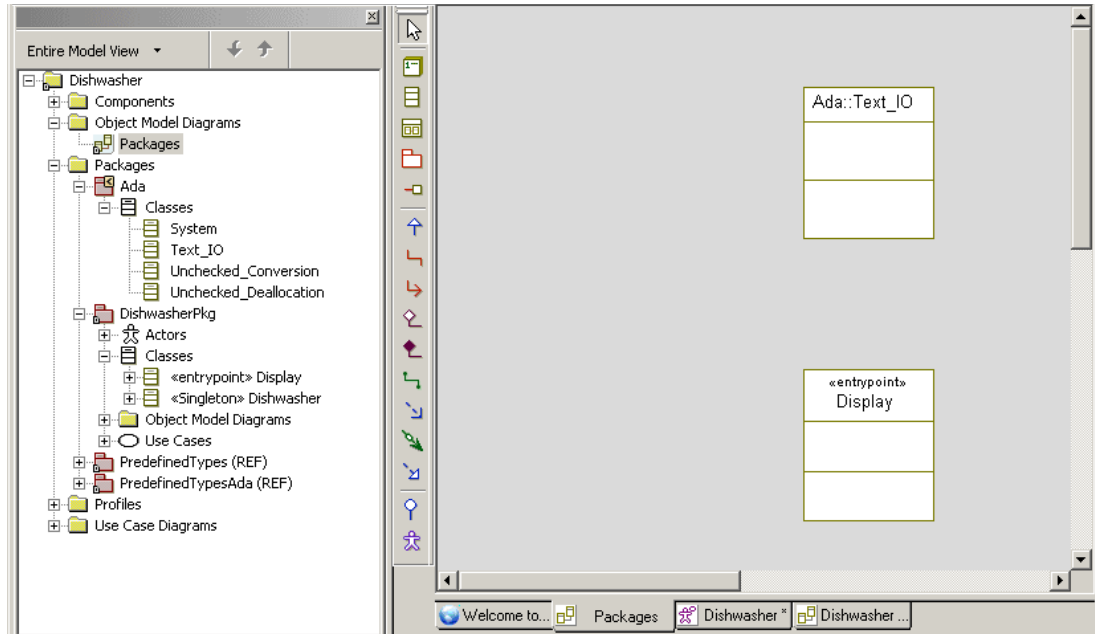
7. On the **General** tab, rename this model as `Packages`.
8. Click **OK**.

Your Rational Rhapsody browser should resemble the following figure:



9. Double-click `Packages` object model diagram to bring its drawing area forward.
10. Drag-and-drop the **Display** class onto the `Packages` object model diagram you just created.
11. Right-click the **Ada** package you added to the model earlier, and then select **Add New > Class**.
12. Name the new class `Text_IO` and press **Enter**.
13. Drag-and-drop the **Text\_IO** class onto the `Packages` object model diagram.


14. Save your model. Your Packages object model diagram should resemble the following figure:



15. Continue with [Task 2c: Establishing the Package Dependency](#).

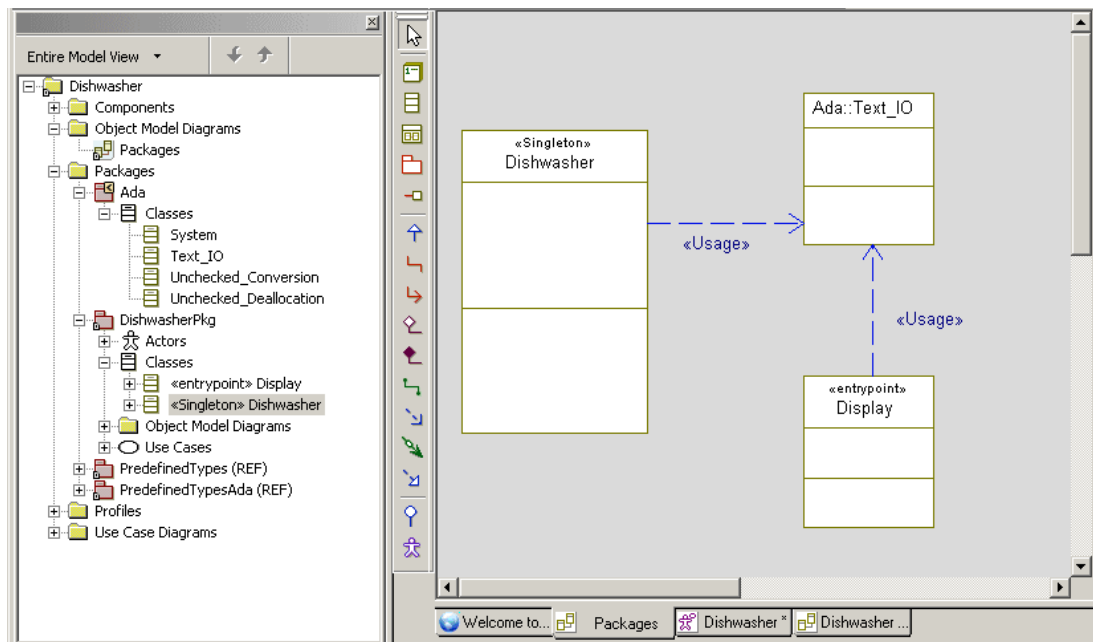
## Task 2c: Establishing the Package Dependency

To establish the package dependency, follow these steps:

1. Continuing from the previous task, for your Packages object model diagram, click the **Dependency** button  and draw a dependency arrow from the **Display** class to the **Text\_IO** class.
2. Double-click the **Dependency** line to open the Features dialog box.
3. In the **Stereotype** box, select the **Usage In PredefinedTypes** check box from the drop-down menu and click **Apply**.
4. Leave the Features dialog box open. Move it aside if necessary.
5. Drag-and-drop the **Dishwasher** class onto the object model diagram.
6. Create a dependency from the **Dishwasher** class to the **Text\_IO** class and set the line to the **Usage** stereotype.
7. Click **Apply** and then **OK**.

### 8. Save your model.

Now the **Display** and **Dishwasher** classes can use the **Text\_IO** package. Your Packages object model diagram should resemble the following figure:



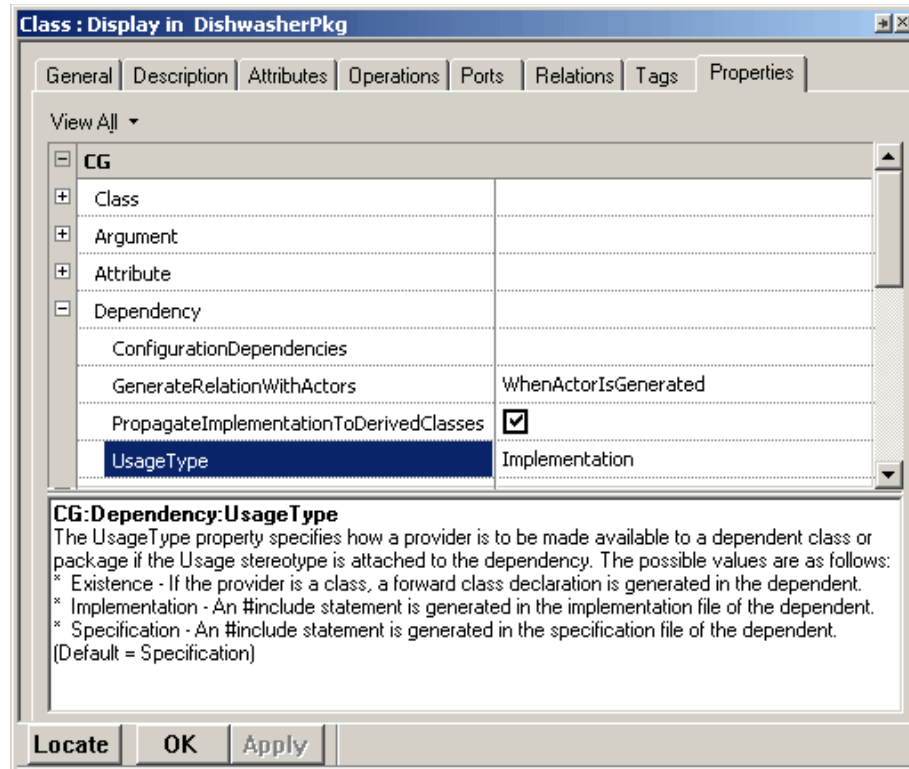
## Task 2d: Setting a Package Dependency Property

Rational Rhapsody enables you to specify whether the dependency should generate the context clause in the unit specification or the unit body. Because the **Display** class has only a body, you must set the generation location of the context clause to *implementation body*.

To set the property, follow these steps:

1. Double-click the **Display** class in the Rational Rhapsody browser or on the diagram to open the Features dialog box.
2. On the **Properties** tab, make sure that all the subjects are available.
3. Locate the `CG::Dependency::UsageType` property.

- Click the box next to this property and select **Implementation** from the drop-down list, as shown in the following figure:



- Click **OK** to apply your changes and close the dialog box.

**Note:** Because you set this property for the class and not on an individual dependency, it applies to *all* the dependencies in this class.

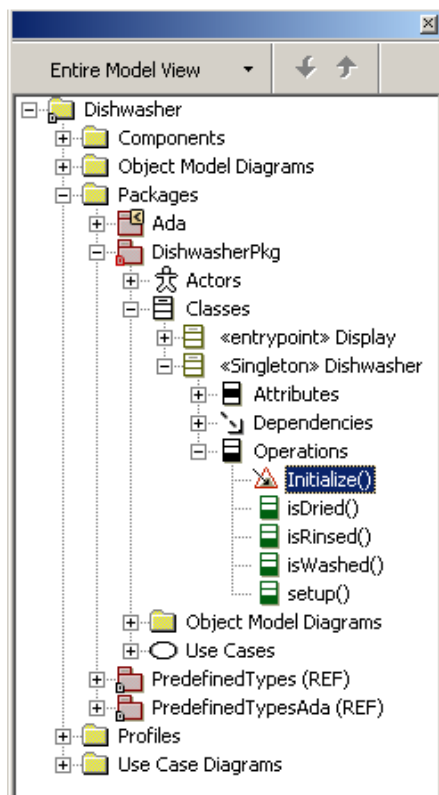
- Save your model.

## Task 2e: Adding a default constructor

In preparation for the next lesson where you generate code and try to build your model for the first time, you want to add a *constructor* to the Dishwasher class. A constructor is called when an object is instantiated. An object can use a constructor to explicitly initialize object members or dynamically allocate space for member pointers. For our model, the constructor makes it possible for the code to compile correctly when not using animation, which you will not get to until later in this tutorial.

To add a default constructor, follow these steps:

1. On the Rational Rhapsody browser, right-click the Dishwasher class and select **Add New > Constructor** to open the Constructor Arguments dialog box.
2. Click **OK**.
3. Expand the **Operations** category for the **Dishwasher** class and notice that Rational Rhapsody adds an Initialize operation for the **Dishwasher** class, as shown in the following figure:



## Summary

In this lesson, you created use object model diagrams that specified the types of objects in the system, and the attributes and operations that belong to those objects. You became familiar with the parts of an object model diagram and created the following:

- ◆ Classes
- ◆ Dependencies
- ◆ Attributes
- ◆ Operations
- ◆ Stereotypes
- ◆ Packages
- ◆ Constructors

You are now ready to proceed to the next lesson, where you are going to generate code and try to build your model in its current state. This lets you determine whether the model meets the requirements and identify defects early on in the design process.





# Lesson 3: Generating Code and Building Your Model

---

It is good practice to test the model incrementally using model execution. You can animate pieces of the model as it is developed. This gives you the opportunity to determine whether the model meets the requirements and find defects early on. Then you can test the entire model. In this way, you iteratively build the model, and then with each iteration perform an entire model validation.

## Goals for this Lesson

In this lesson, you are going to prepare for generating code, generate code, and try to build your model.

## Exercise 1: Preparing for Generating Code

Before you generate code, you must do the following general steps:

1. Create a component and set its features.
2. Create a configuration.

The following tasks describe these steps in detail.

## Task 1a: Creating a Component

A *component* is a physical subsystem in the form of a library or executable program. It plays an important role in the modeling of large systems that contain several libraries and executables. Each component contains configuration and file specification categories, which are used to generate, build, and run the executable model.

The name of the component becomes the name of the executable application to build. This component defines classes for which to generate code and options to apply to the generated code.

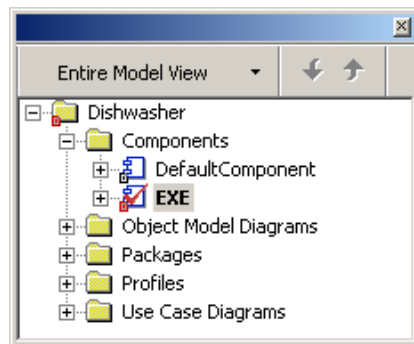
Each project contains a default component, named `DefaultComponent`. You can use the default component or create a new component. In this task, you are going to create a new component called `EXE`. Later you will use the **EXE** component to animate the model.

To use create a component, follow these steps:

1. In the Rational Rhapsody browser, expand the top-level **Dishwasher** category.
2. Right-click **Components** and select **Add New Component**.

Rational Rhapsody creates a new component called `component_n`, where  $n$  is greater than or equal to 0.

3. Rename the component `EXE` and press **Enter**. Rational Rhapsody displays the renamed component in the browser, as shown in the following figure:



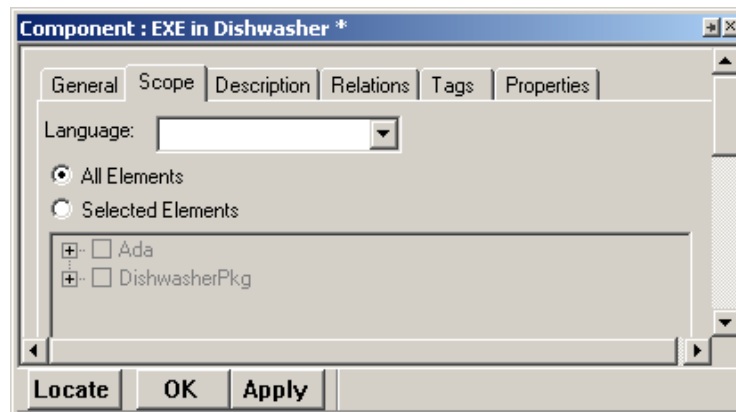
4. Because the **DefaultComponent** is not used in this project and to unclutter the browser, right-click **DefaultComponent** and select **Delete from Model**, and then click **Yes** to confirm the requested action.
5. Continue with [Task 1b: Setting the Component Features](#).

## Task 1b: Setting the Component Features

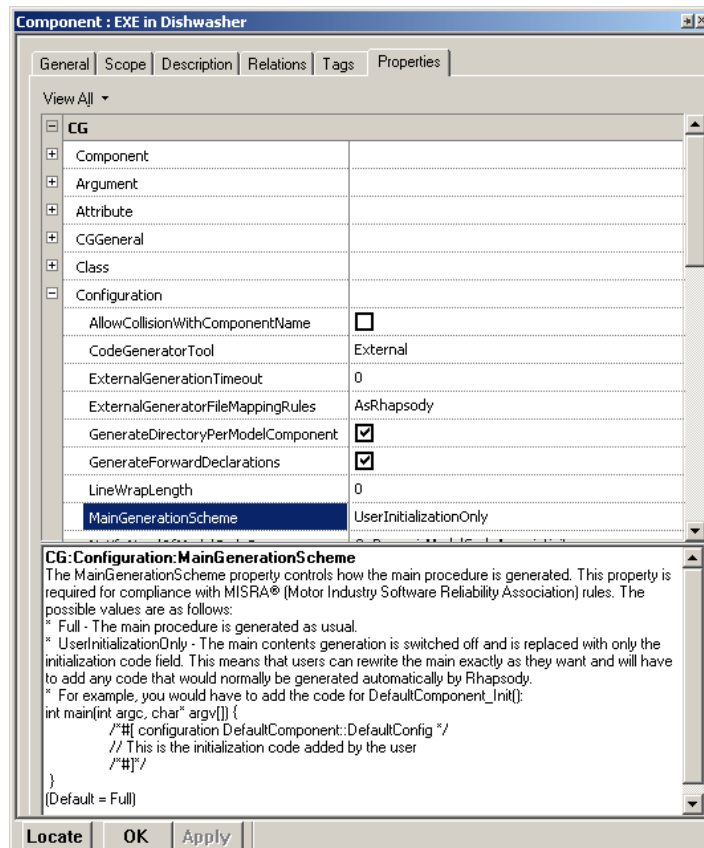
Once you have created the component, you must set its features.

To set the features for your component, follow these steps:

1. Continuing from [Task 1a: Creating a Component](#), double-click the **EXE** component or right-click and select **Features** to open the Features dialog box.
2. On the **General** tab, in the **Type** group, select the **Executable** option button if it is not already selected.
3. On the **Scope** tab, select the **All Elements** option button, as shown in the following figure:



4. Do the following to deactivate the automatically generated `main` so that only the user-defined one is used:
  - a. On the **Properties** tab, make sure all the subjects are available. This should be set from the last time you used this tab.
  - b. Locate the `CG::Configuration::MainGenerationScheme` property.
  - c. Click the value next to the property name and use the drop-down list to change the property value to **UserInitializationOnly**, as shown in the following figure:



5. Click **OK**.

## Task 1c: Creating a Configuration

A component can contain many configurations. A *configuration* specifies how the component is to be produced. Each component contains a default configuration, named `DefaultConfig`. In this task, you are going to rename the default configuration to `Host`.

To rename the default configuration, follow these steps:

1. In the Rational Rhapsody browser, expand the **EXE** component and the **Configurations** category.
2. Double-click **DefaultConfig** or right-click and select **Features** to open the Features dialog box.
3. On the **General** tab, in the **Name** box, replace `DefaultConfig` with `Host`.
4. Click **Apply** and **OK**.

## Exercise 2: Generating Code

In this exercise, you generate code for the first time for your application.

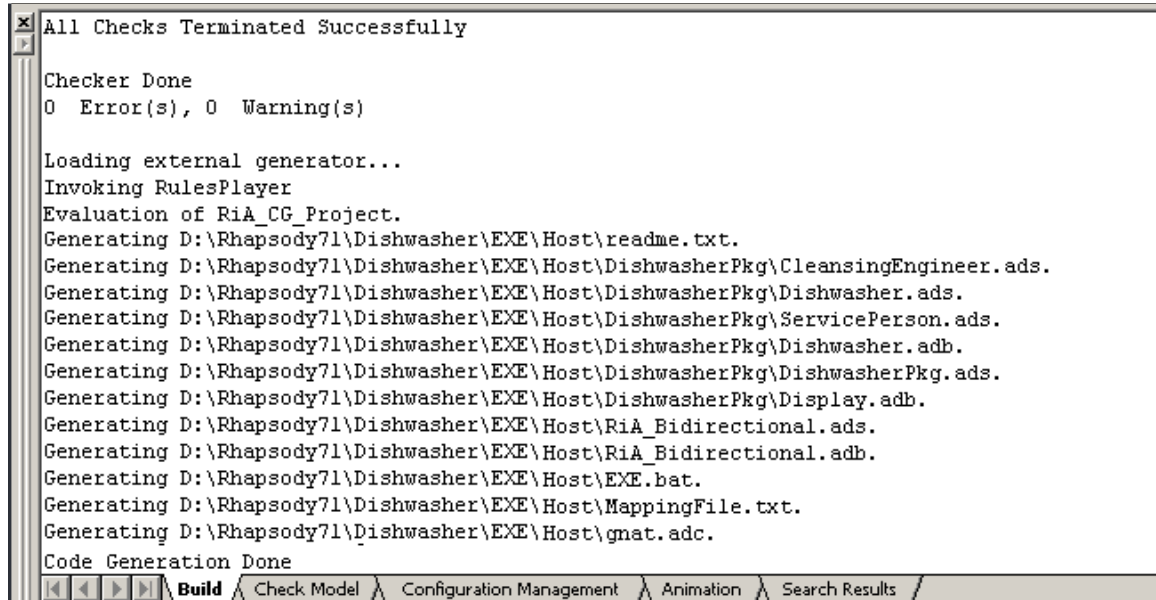
### Task 2a: Generating Code

In this task you generate code in Rational Rhapsody. Before you generate code, you must first set the *active configuration*. The active configuration is the configuration for which you generate code. The active configuration appears in the drop-down list on the **Code** toolbar.

To set the active configuration and generate code for the **Host** configuration, follow these steps:

1. In the Rational Rhapsody browser, right-click the **Host** configuration and then select **Set as Active Configuration**.  
**Note:** You can also select the active configuration from the drop-down list on the **Code** toolbar.
2. Select **Code > Generate > Host**. Rational Rhapsody displays a message that the **Host** directory does not yet exist and asks you to confirm its creation.
3. Click **Yes**. Rational Rhapsody places the files generated for the active configuration in the new `Host` directory.

Rational Rhapsody generates the code and displays output messages in the **Build** tab of the Output window, as shown in the following figure:



```
All Checks Terminated Successfully

Checker Done
0 Error(s), 0 Warning(s)

Loading external generator...
Invoking RulesPlayer
Evaluation of RiA_CG_Project.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\readme.txt.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\DishwasherPkg\CleansingEngineer.ads.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\DishwasherPkg\Dishwasher.ads.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\DishwasherPkg\ServicePerson.ads.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\DishwasherPkg\Dishwasher.adb.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\DishwasherPkg\DishwasherPkg.ads.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\DishwasherPkg\Display.adb.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\RIa_Bidirectional.ads.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\RIa_Bidirectional.adb.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\EXE.bat.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\MappingFile.txt.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\gmat.adc.

Code Generation Done
```

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.

### Task 2b: Fixing Code Generation Errors

If you receive code generation errors, double-click the error in the Output window to go to the source of the error. The source of the error appears as a highlighted element. Once you fix the problem, regenerate the code (choose **Code > Re Generate > Host**) until there are no error messages.

## About Code Generation Warnings

If you receive code generation warnings, double-click the warning in the Output window to go to the source of the warning. The source of the warning appears as a highlighted element. You might be able to fix the warning. Or you might leave the warning as is because your model is not yet fully formed.

Keep in mind that you might receive warnings because your model is not yet fully formed, so that, for example, all your port connections might not yet in place.

In other cases, if you do have warnings that are valid for the current state of your mode, fix them, regenerate the code, and rebuild the application until those warnings are no longer appearing.

## Examining Generated Source Files

To examine any of the generated source files, go to the `Host` subfolder of the Dishwasher project.

## Using External Elements

The Rational Rhapsody product enables you to visualize frozen legacy code or edit external code as external elements. This external code is code that is developed and maintained outside of the Rational Rhapsody product. This code will not be regenerated by the Rational Rhapsody product, but will participate in the code generation and build process of Rational Rhapsody models that interact or interface with this external code. You can create external elements by reverse engineering the files or by modeling. Refer to the *IBM Rational Rhapsody User Guide* for more information on using external elements.

# Exercise 3: Building Your Model

In this exercise, you try to build your model for the first time.

## Task 3a: Building your Model

Once you generate code without any errors, you are ready to build the model.

To build the model, do one of the following:

- ◆ Select **Code > Build EXE.exe**, or
- ◆ Click the **Make** button  on the **Code** toolbar.

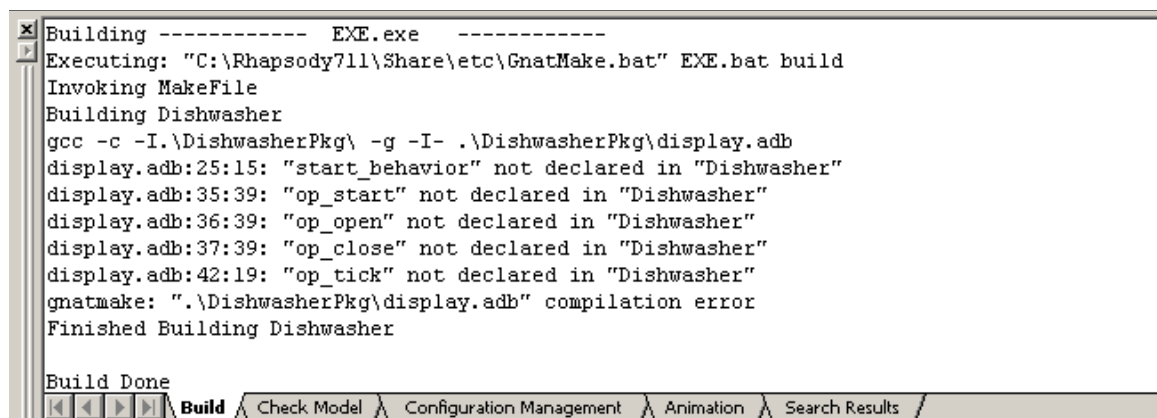
Rational Rhapsody builds the model by performing the following tasks:

- ◆ Executes the makefile that it generated for the configuration.
- ◆ Sets up the environment for the compiler.
- ◆ Starts the compiler and linker, which run on the generated code. Once the code is compiled and linked, the Rational Rhapsody product displays the message `Build Done` in the Output window.

As you can see from the following figure, when you try to build your model code you receive error messages about not declared **op\_** operations for the dishwasher. This is because you have not defined these operations yet. You will do so in the next lesson. You can also ignore the message about **start\_behavior**. So for now, you can ignore these message.

If you see other error messages, you should correct them; continue with [Task 3b: Fixing Build Errors](#).

If you have no error messages, continue with [Task 3c: Viewing Code](#).



```
Building ----- EXE.exe -----
Executing: "C:\Rhapsody711\Share\etc\GnatMake.bat" EXE.bat build
Invoking MakeFile
Building Dishwasher
gcc -c -I.\DishwasherPkg\ -g -I- .\DishwasherPkg\display.adb
display.adb:25:15: "start_behavior" not declared in "Dishwasher"
display.adb:35:39: "op_start" not declared in "Dishwasher"
display.adb:36:39: "op_open" not declared in "Dishwasher"
display.adb:37:39: "op_close" not declared in "Dishwasher"
display.adb:42:19: "op_tick" not declared in "Dishwasher"
gnatmake: ".\DishwasherPkg\display.adb" compilation error
Finished Building Dishwasher

Build Done
```

The screenshot shows the Rational Rhapsody Output window. The title bar reads "Building ----- EXE.exe -----". The main text area displays the following output: "Executing: 'C:\Rhapsody711\Share\etc\GnatMake.bat' EXE.bat build", "Invoking MakeFile", "Building Dishwasher", "gcc -c -I.\DishwasherPkg\ -g -I- .\DishwasherPkg\display.adb", "display.adb:25:15: 'start\_behavior' not declared in 'Dishwasher'", "display.adb:35:39: 'op\_start' not declared in 'Dishwasher'", "display.adb:36:39: 'op\_open' not declared in 'Dishwasher'", "display.adb:37:39: 'op\_close' not declared in 'Dishwasher'", "display.adb:42:19: 'op\_tick' not declared in 'Dishwasher'", "gnatmake: '.\DishwasherPkg\display.adb' compilation error", and "Finished Building Dishwasher". At the bottom, the status bar shows "Build Done" and a menu with options: Build, Check Model, Configuration Management, Animation, Search Results.

## Task 3b: Fixing Build Errors


If you receive build errors, double-click the error in the Output window to go to the source of the error. The source of the error appears as a highlighted element. Once you fix the problem, regenerate the code and rebuild the application until there are no error messages.

Any time you make changes to the model, you need to regenerate the code (choose **Code > Re Generate > Host**, in this case) and rebuild the model (choose **Code > Rebuilt Exe.exe**, in this case). For more information about full code generation and an incremental code generation, refer to the *IBM Rational Rhapsody User Guide*. (Do a search of the user guide PDF for “incremental code generation.”) You might also find it useful to use the Clean function. Do a search of the user guide PDF for “deleting old objects.”



## Task 3c: Viewing Code

To view the generated code, perform these steps:

1. For example, select the **Dishwasher** class on the Dishwasher object model diagram and choose **View > Active Code View**.
2. Review the code on the **Dishwasher.abs** tab.
3. If you want to see line numbers on the Active Code View window, do the following:
  - a. Right-click in the window and select **Properties** to open the Windows Properties dialog box.
  - b. On the **Misc** tab, in the **Line Numbering** area, select a numbering style from the drop-down list (for example, **Decimal**).
  - c. Click **OK**.
4. To close the Active Code View window, click  (the Hide Docked Window button) for that window.

## Summary

In this lesson, you prepared for code generation, generated code, and tried to build your model at its current point. You performed the following:

- ◆ Created a component and set its features
- ◆ Created a configuration and set it as the active configuration
- ◆ Generated code in Rational Rhapsody
- ◆ Tried to build the Dishwasher model at its current point
- ◆ Viewed code

You are now ready to proceed to the next lesson, where you continue to create your Dishwasher model. You are going to define the behavior of objects, including the various states that an object can enter into over its lifetime and the messages or events that cause it to transition from one another by drawing statecharts. You also get to regenerate code and try to build your model again.



# Lesson 4: Creating a Statechart

---

*Statecharts* (SCs) define the behavior of objects, including the various states that an object can enter into over its lifetime and the messages or events that cause it to transition from one state to another. Each statechart defines the life cycle behavior of a single reactive class. Therefore, a single reactive class can be associated with only one statechart.

## Goals for this Lesson

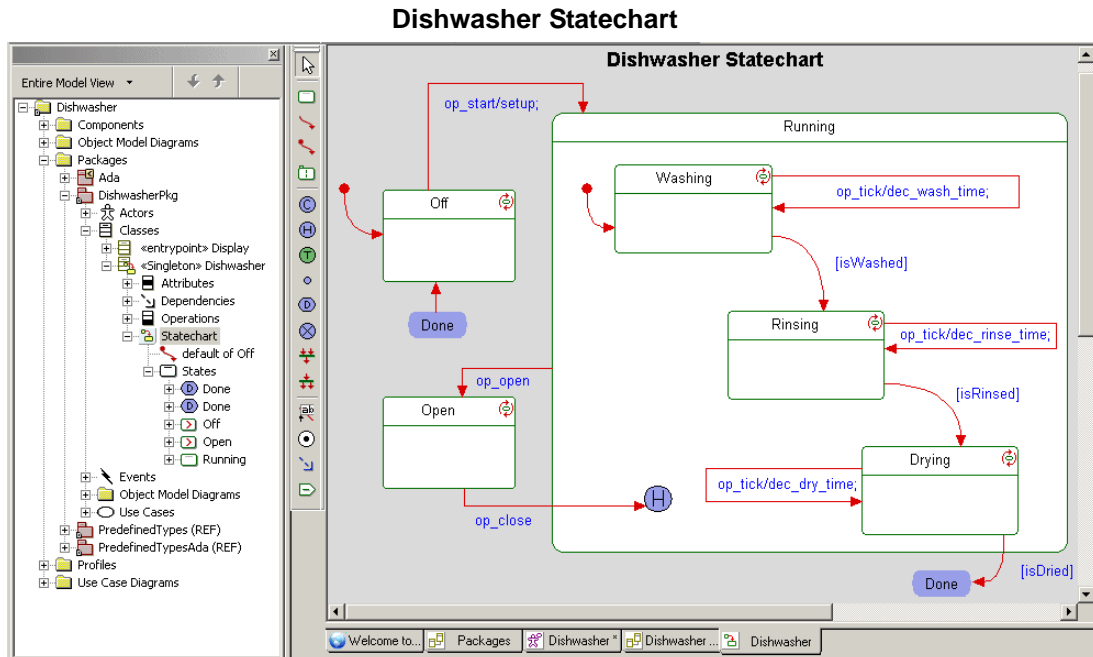
In this lesson you will perform the following tasks:

- ◆ Draw a statechart
- ◆ Draw states, transitions, and nested states
- ◆ Put Ada timeouts on transitions
- ◆ Put actions on transitions
- ◆ Specify entry and exit actions
- ◆ Draw history connectors
- ◆ Change operations synchronization

## Exercise 1: Creating the Dishwasher Statechart

Statecharts define the behavior of objects, including the various states that an object can enter into over its lifetime and the messages or events that cause it to transition from one state to another.

The following figure shows the Dishwasher statechart that you are going to create in this exercise.



## Task 1a: Creating a Statechart

You can create either a *statechart* or an *activity diagram* to model the behavior of your system. For example, objects in a model can have different states, such as On or Off. An activity diagram is used to incorporate these states into a model if no transitions are needed between the states. In this case, the `Dishwasher` needs to transition from state to state in reaction to events, so a statechart is used instead of an activity diagram.

You can use the Rational Rhapsody browser, the **Tools** menu, or the **Open Statechart** button on the **Diagrams** toolbar to create a new statechart. This task describes how to create a statechart through use of the browser.

To create a statechart, follow these steps:

1. Start Rational Rhapsody and the Dishwasher model if they are not already open.
2. In the Rational Rhapsody browser, right-click the **Dishwasher** class.
3. Select **Add New > Statechart**.

Rational Rhapsody automatically adds the new statechart under the **Dishwasher** class in the browser. In addition, Rational Rhapsody opens the new statechart in the drawing area.

## Task 1b: Drawing the Dishwasher Statechart

The general steps for drawing the Dishwasher statechart are as follows:


1. Draw states.
2. Draw history and diagram connectors.
3. Draw default connectors.
4. Add Ada operations.
5. Draw transitions.
6. Add actions to states.

The following sections describe these steps in detail.

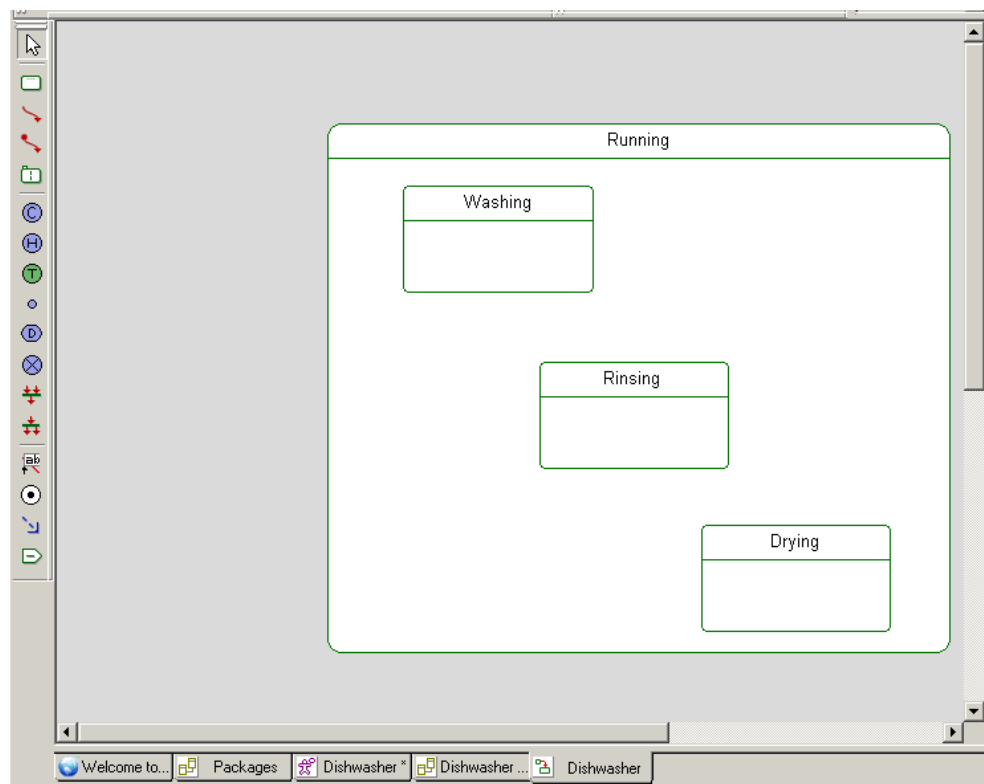
## Lesson 4: Creating a Statechart

---

To draw a state, follow these steps:

1. Click the **State** button  in the **Drawing** toolbar.
2. Click-and-drag on the drawing area to create a large state with a default name of `state_n`, where  $n$  is greater than or equal to 0.
3. Rename the state `Running`.
4. Using the [Dishwasher Statechart](#) as a reference, draw three more states inside the **Running** state, as shown in the following figure:

- Washing
- Rinsing
- Drying





5. To the left and outside the **Running** state, draw two more states and name them `off` and `Open`.
6. Continue with the next task.

## Task 1c: Drawing History and Diagram Connectors

If you open and close the door during operation, the dishwasher must start up again where it left off in the wash cycle. In other words, you want the dishwasher to save its history so it can continue where it left off after an interruption. *History connectors* store the most recent active configuration of a state. A transition to a history connector restores this configuration.

When the dishwasher is done drying, the cycle should start over again at the beginning, to handle future loads. To define the cycle restart, use *diagram connectors* to connect the end of one part of a statechart to the beginning of another part. These connectors physically join distant transition segments. Diagram connectors have the same name to indicate they are a pair of connectors. This tells the system to jump from one to the other even if they are located on different statecharts.

To draw the connectors, follow these steps:

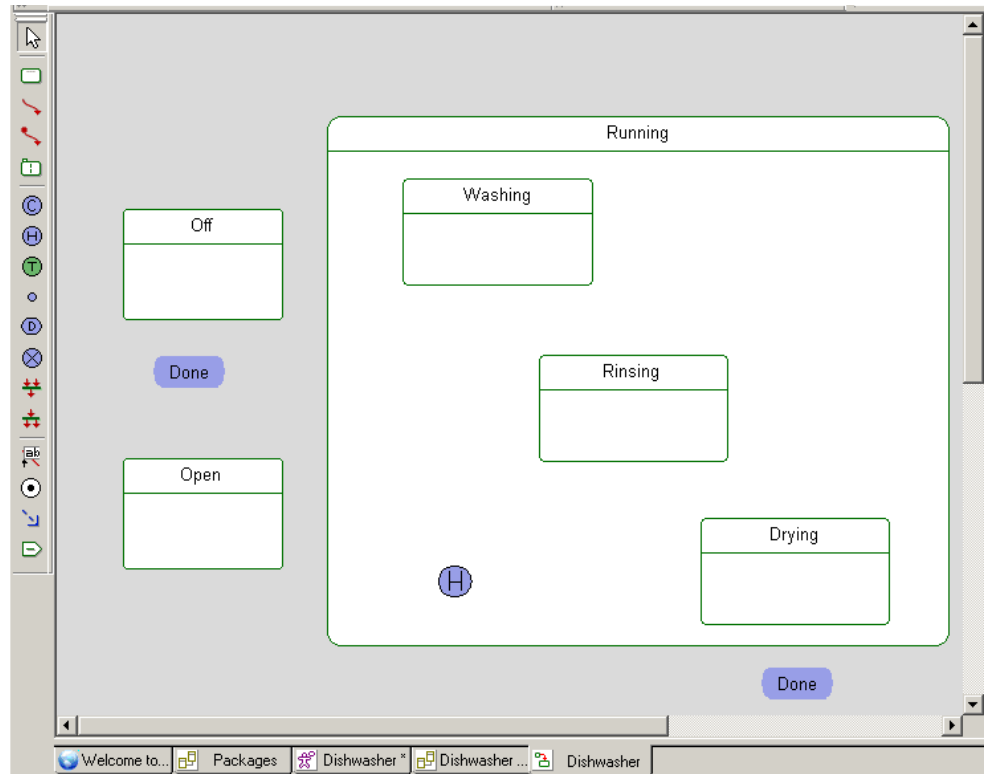
1. Click the **History connector** button  on the **Drawing** toolbar and then click in the lower left corner inside the **Running** state.
2. Click the **Diagram connector** button  on the **Drawing** toolbar and create the following diagram connectors and label them `Done` in the following locations:
  - Below the **Off** state. This is the source diagram connector.
  - Outside the **Running** state below the **Drying** state. This is the target diagram connector.

## Lesson 4: Creating a Statechart

---

3. Save your model.

Your statechart should resemble the following figure:






## Task 1d: Drawing Default Connectors

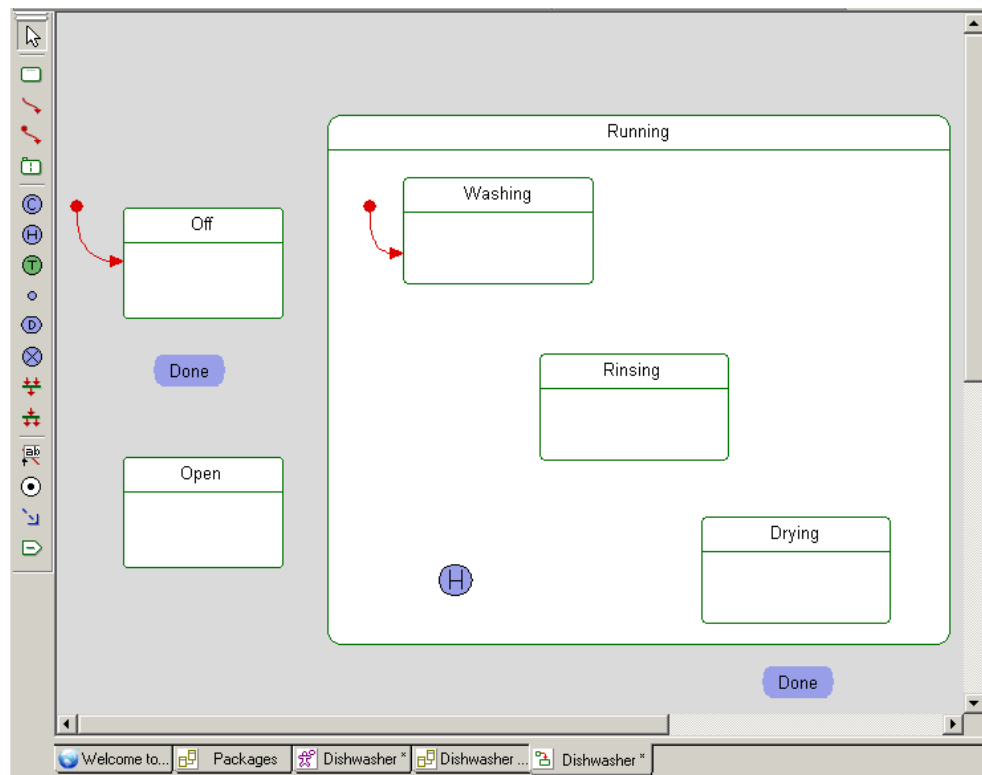
One object must be assigned the *default* state. In the default state, the object knows to start the system. When the dishwasher first starts, it is in the **Off** state.

Note that once you have drawn a default connector in a statechart, Rational Rhapsody does not allow you to draw another one in the same chart. Each object can have only one default state.

To assign the default states for classes in the statechart, follow these steps:

1. Click the **Default connector** button  in the **Drawing** toolbar, click in the drawing area to above the **Off** state, and then move your cursor to the **Off** state and click the edge of the state.
2. Click away from the label box to skip naming the connector.
3. Use the same method to draw a default connector to the **Washing** state, keeping the connector inside the **Running** state.
4. Click away from the label box to skip naming the connector.


At this point your statechart should resemble the following figure:



## Task 1e: Adding Ada Operations

You need to create additional operations to determine the duration for the previously created operations.

To define these operations, follow these steps:

1. Expand the **Packages** category in the Rational Rhapsody browser.
2. Double-click the **Dishwasher** class or right-click and select **Features** to open the Features dialog box for the class.
3. On the **Operations** tab, click <New> and select **PrimitiveOperation** to create a blank line for a new operation.
4. Name this operation `dec_dry_time` and accept the other default settings.
5. Click **Apply**.
6. Double-click the icon to the left of **dec\_dry\_time** or highlight it and click  to open the Features dialog box for the operation.
7. On the **Implementation** tab, type the following text in the **Operation Body** area:

```
Dishwasher_unique_instance.drytime :=  
Dishwasher_unique_instance.drytime - 1;
```

8. Click **OK** to apply your changes and close the Features dialog box for the operation and return to the Features dialog box for the class.
9. Repeat Steps 3 – 8 to create the following operations and enter implementations for each.

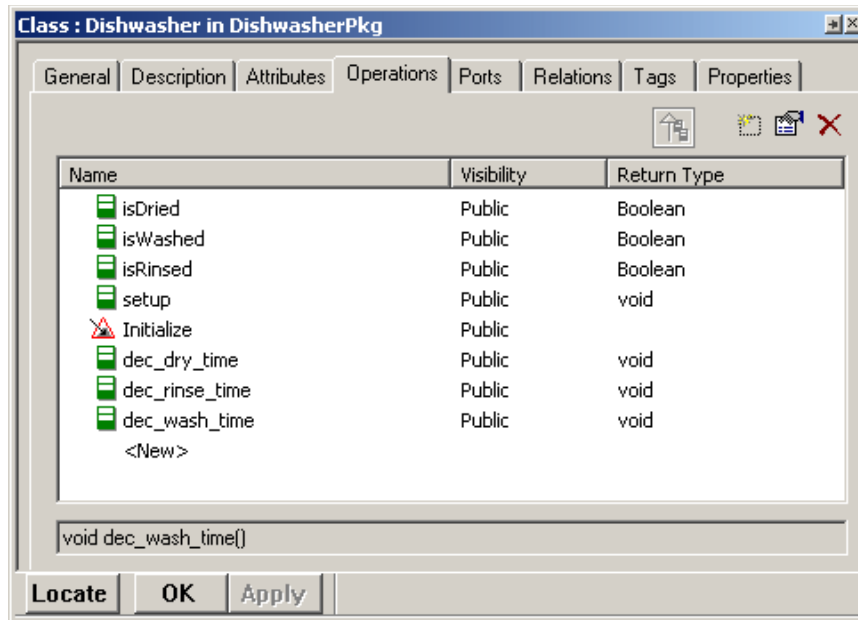
– `dec_rinse_time` with the following implementation code.

```
Dishwasher_unique_instance.rinsetime :=  
Dishwasher_unique_instance.rinsetime - 1;
```

– `dec_wash_time` with the following implementation code:

```
Dishwasher_unique_instance.washtime :=  
Dishwasher_unique_instance.washtime - 1;
```

10. Your Operations tab should resemble the following figure:




11. When done, click **OK** to close all Feature dialog boxes.

## Task 1f: Drawing the Transitions

A *transition* represents a message or event that causes an object to switch from one state to another. Use the [Dishwasher Statechart](#) as a reference to do this task.

To add transitions, follow these steps:

1. Click the **Transition** button  in the **Drawing** toolbar.
2. Click the edge of the **Off** state to anchor the start of the transition.
3. Move the cursor to the **Running** state and click its to anchor the transition line.
4. In the label box, type `op_start/setup`; and press **Ctrl+Enter**. (Pressing only **Enter** inserts a new line.)

This creates an event and an action with the same name.

5. For purposes of illustrating the possible line shapes, this task uses two line shapes. By default, Rational Rhapsody uses the Spline line shape. To change the line shape for a transition, right-click the line in the drawing area, select **Line Shape**, and then one of the following options:
  - **Straight** to change the line to a straight line.
  - **Spline** to change the line to a curved line.
  - **Rectilinear** to change the line to a group of line segments connected at right angles.
  - **Re-Route** to remove excess control points to make the line more fluid.
6. Draw a transition from the **Running** to the **Open** state and label it `op_open`.
7. Draw a transition from the **Open** state to the history connector and label it `op_close`.
8. Inside the **Running** state, draw a transition from the **Washing** state to the **Rinsing** state and label it `[isWashed]`.

**Note:** The square brackets must be included because they denote a guard condition. A *guard* is a Boolean condition that, if specified, must be true for the transition to be taken. In this case, when the dishes are washed (`isWashed` is `true`), the dishwasher transitions from the **Washing** to the **Rinsing** state. Previously in this tutorial, you specified transitions only with event triggers. However, *transition labels* can have up to three parts, all of which are optional: trigger, guard, action. You can specify the trigger, guard, and action textually in the transition, or enter them in the features dialog box for the transition. Refer to the *IBM Rational Rhapsody User Guide* to learn more about transition labels.

9. Draw a transition from the **Rinsing** state to the **Drying** state and label it `[isRinsed]`.
10. Draw a transition from **Drying** state to the **Done** connector and label it `[isDried]`.
11. Draw a self-directed transition (shown by an arrow that bends back to the sending state) on the **Washing** state and label it:

```
op_tick/dec_wash_time;
```

**Note:** This transition keeps the **Dishwasher** in washing mode for `dec_wash_time`; when the specified time is up and the guard `[isWashed]` is true, the **Dishwasher** transitions from **Washing** to **Rinsing** mode.

12. Add a self-directed message on the **Rinsing** state and label it:

```
op_tick/dec_rinse_time;
```

13. Add a self-directed message on the **Drying** state and label it:

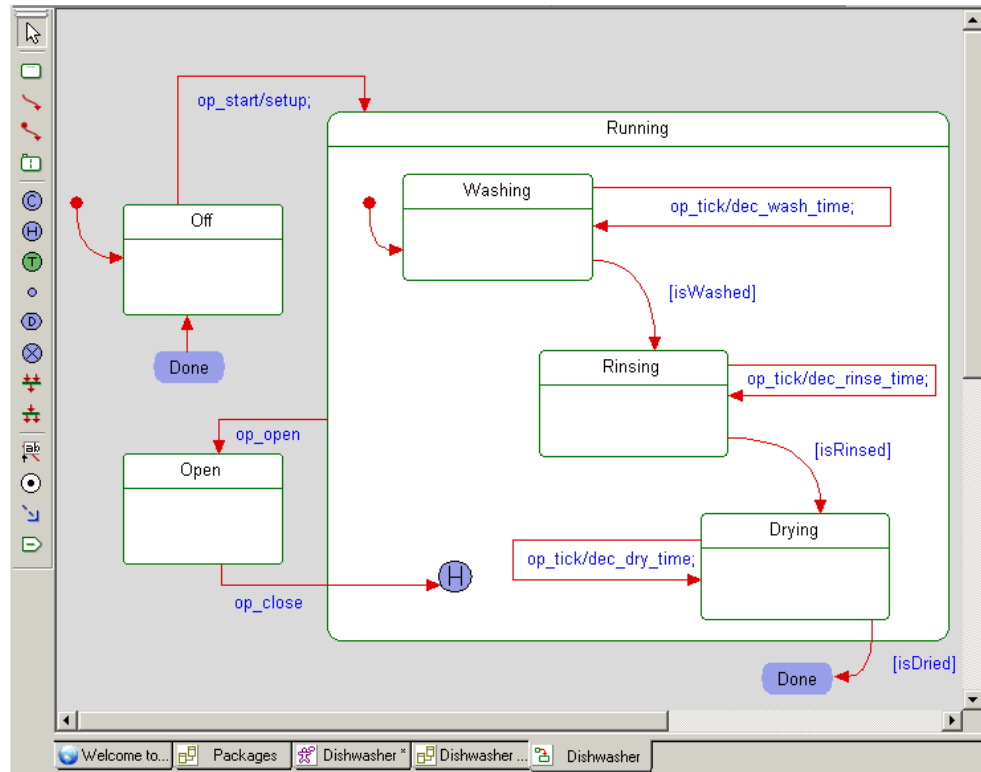
```
op_tick/dec_dry_time;
```

- Draw an unlabeled transition from the target diagram connector (**Done** under the **Off** state) to the **Off** state.

At this point your statechart includes the time descriptions for all of the major operations.

- Save your model.

Your statechart and should resemble the following figure:



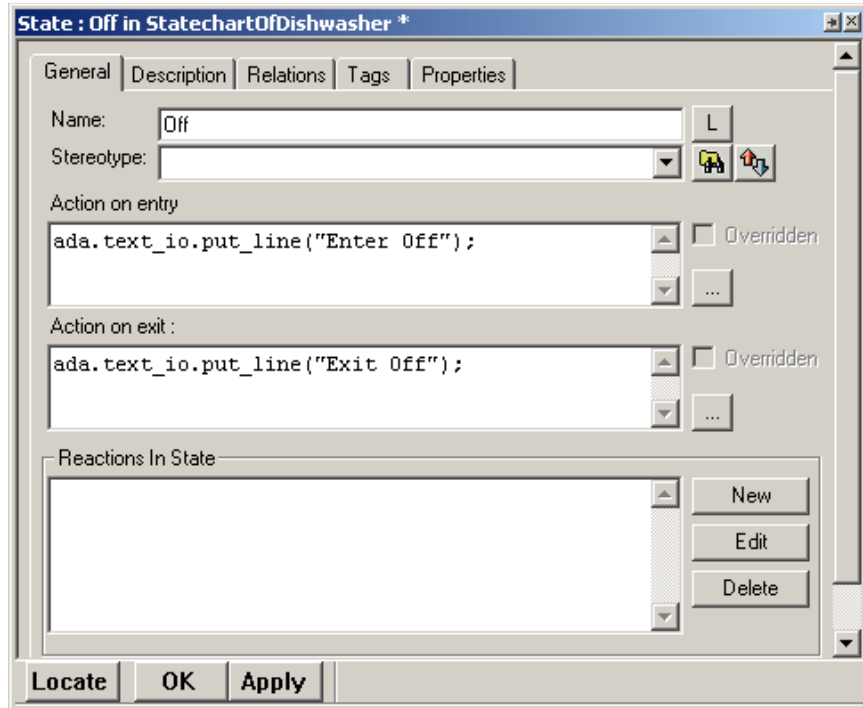
## Task 1g: Adding Actions to States


*Actions on entry* are actions specified for a state that an object performs when it enters that state. Similarly, *actions on exit* are actions that an object performs when it exits the state. In this example, the `Dishwasher` sends an enter or exit action to each of the states.

To specify actions on entry and exit, follow these steps:

- Double-click the **Off** state or right-click and select **Features** to open the Features dialog box.
- On the **General** tab, in the following boxes, type the following code, which is also shown in the following figure:

- **Action on entry:**  
`ada.text_io.put_line("Enter Off");`
- **Action on exit:**  
`ada.text_io.put_line("Exit Off");`



3. Click **Apply** to apply your changes and keep the Features dialog box open. On your statechart, notice that the **Off** state has a symbol  that indicates that the **Off** state now has underlying actions.
4. Double-click the **Washing** state and in the following boxes, type the corresponding code:
  - **Action on entry:**  
`ada.text_io.put_line("Enter Washing");`
  - **Action on exit:**  
`ada.text_io.put_line("Exit Washing");`

5. Click **Apply**.
6. Double-click the **Rinsing** state and in the following boxes, type the corresponding code:
  - **Action on entry:**  

```
ada.text_io.put_line("Enter Rinsing");
```
  - **Action on exit:**  

```
ada.text_io.put_line("Exit Rinsing");
```
7. Click **Apply**.
8. Double-click the **Drying** state and in the following boxes, type the corresponding code:
  - **Action on entry:**  

```
ada.text_io.put_line("Enter Drying");
```
  - **Action on exit:**  

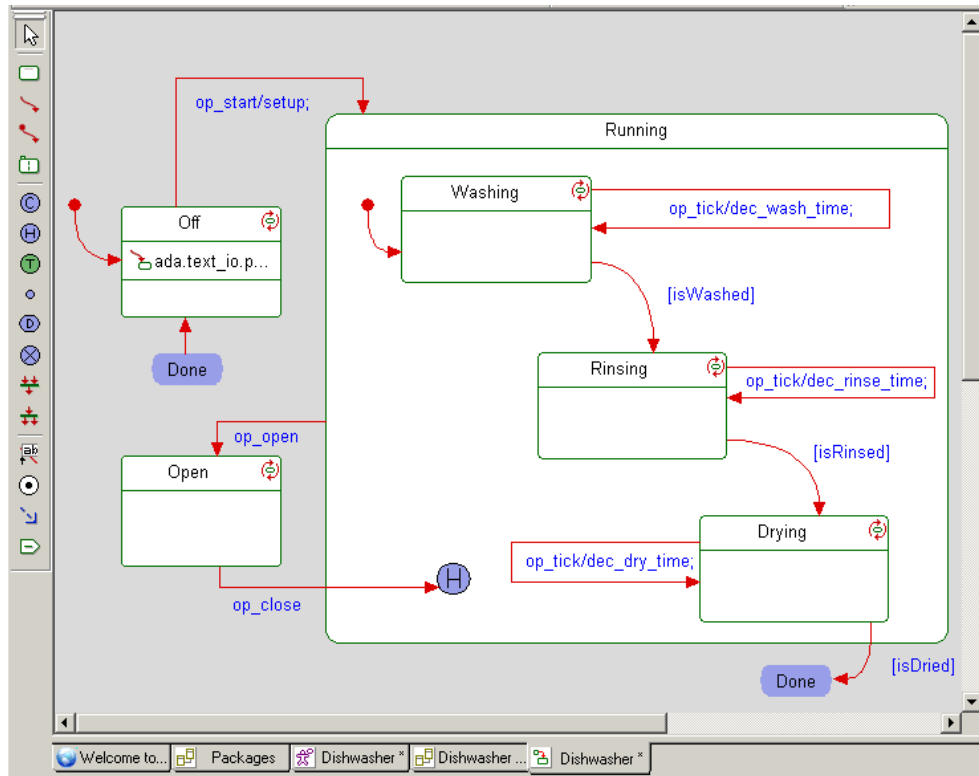
```
ada.text_io.put_line("Exit Drying");
```
9. Click **Apply**.
10. Double-click the **Open** state and in the following boxes, type the corresponding code:
  - **Action on entry:**  

```
ada.text_io.put_line("DOOR OPEN!!!");
```
  - **Action on exit:**  

```
ada.text_io.put_line("Door Closed");
```
11. Click **Apply** and then **OK** to close the Features dialog box.

12. Save your model.

Your statechart should resemble the following figure:




### Task 1h: Changing Operation Synchronization

Events and operations relate statecharts to the rest of the model by triggering transitions. Operations specified by a statechart are called triggered operations (as opposed to operations specified in object model diagrams, called primitive operations).

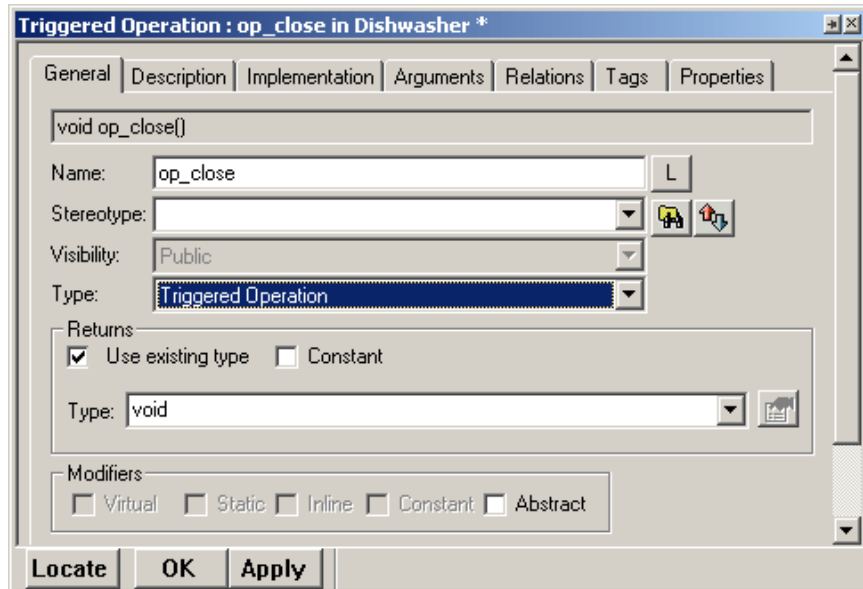
Events facilitate asynchronous collaborations and operations facilitate synchronous collaborations. Triggered operations have a return type and reply. Triggered operations have a higher priority than events.


To change each *asynchronous* operation to be *synchronous*, follow these steps:

1. In the browser, expand the **Dishwasher** class (in the **DishwasherPkg** package) and then expand **Operations** so that you see a list of all the operations.
2. Notice the **op\_** elements (for example, **op\_close**), especially notice the icon  for them as it will change once you change the type operation.



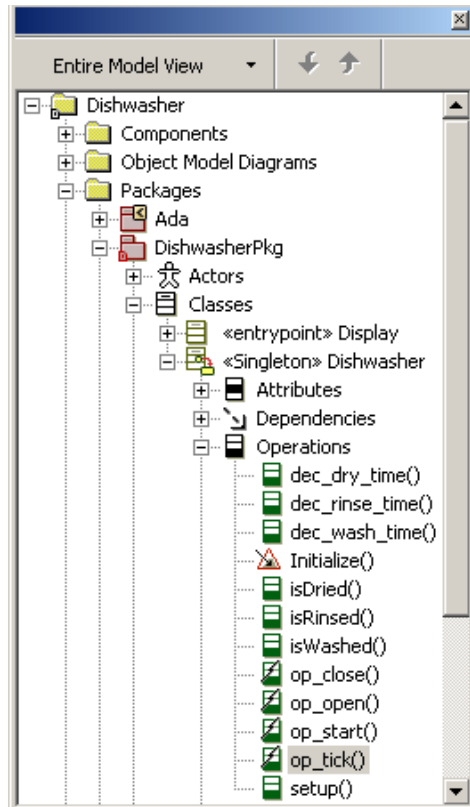
3. Double-click one of the **op\_** elements or right-click it (for example, **op\_close**) and select **Features** to open the Features dialog box.
4. On the **General** tab, in the **Type** box, use the drop-down list to change **Reception** to **Triggered Operation**, as shown in the following figure:



5. Click **Apply** to apply your change. On the Rational Rhapsody browser, notice that the icon  for the operation has changed.
6. With the Features dialog box still opened, select another **op\_** operation and repeat this change for each **op\_** operation in the browser list (**op\_close**, **op\_open**, **op\_start**, and **op\_tick**). Click **Apply** after each change.
7. Click **OK** to close the Features dialog box.

### 8. Save your project.

Your Rational Rhapsody browser should resemble the following figure.



### Task 1i: Adding a Diagram Title

To add a title to your statechart diagram, see [Task 2e: Adding a Diagram Title](#).

If you do this task, remember to save your project.

## Exercise 2: Generating Code and Building Your Model

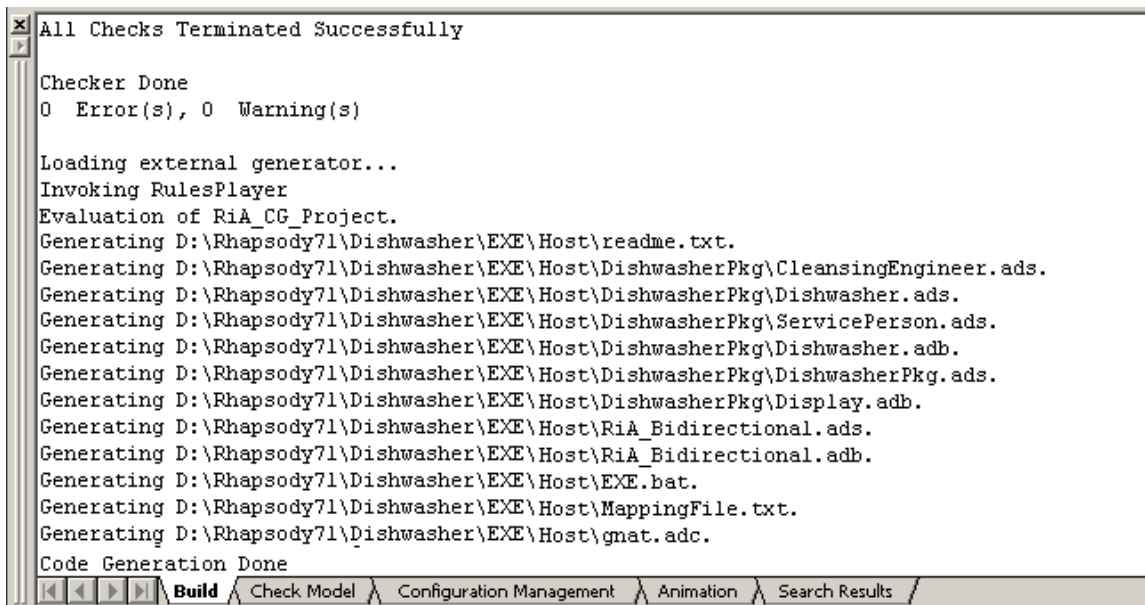
In the previous lesson you generated code and tried to build your Dishwasher model for the first time. In this exercise, you will generate code and try to build your model again now that you have added a statechart to your model.

### Task 2a: Generating Code

To generate code, follow these steps:

1. If necessary, in the Rational Rhapsody browser, right-click the **Host** configuration and then select **Set as Active Configuration**.
2. Select **Code > Re Generate > Host**.

Rational Rhapsody generates the code and displays output messages in the **Build** tab of the Output window, as shown in the following figure. If you have any error messages, see [Task 2b: Fixing Code Generation Errors](#).



```
All Checks Terminated Successfully


Checker Done
0 Error(s), 0 Warning(s)

Loading external generator...
Invoking RulesPlayer
Evaluation of RiA_CG_Project.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\readme.txt.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\DishwasherPkg\CleansingEngineer.ads.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\DishwasherPkg\Dishwasher.ads.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\DishwasherPkg\ServicePerson.ads.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\DishwasherPkg\Dishwasher.adb.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\DishwasherPkg\DishwasherPkg.ads.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\DishwasherPkg\Display.adb.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\RiA_Bidirectional.ads.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\RiA_Bidirectional.adb.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\EXE.bat.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\MappingFile.txt.
Generating D:\Rhapsody71\Dishwasher\EXE\Host\gmat.adc.
Code Generation Done
```

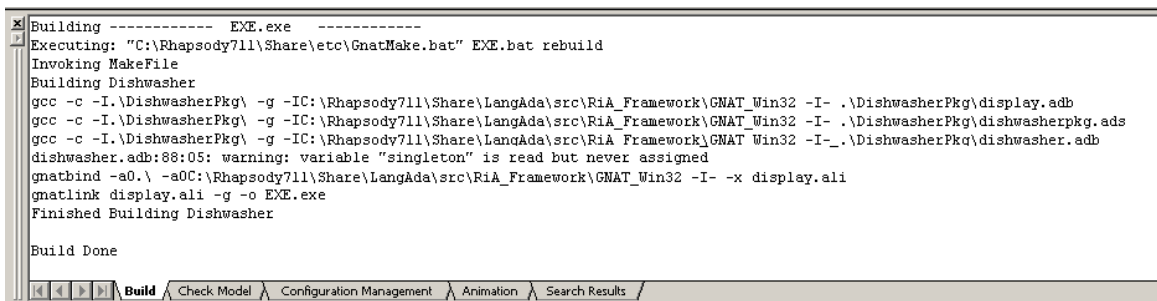
## Task 2b: Building the Model

Once you regenerate code without any errors, you are ready to rebuild the model.

To rebuild the model, do one of the following:

- ◆ Select **Code > Rebuild EXE.exe**, or
- ◆ Click the **Make** button  on the **Code** toolbar.

As you can see from the following figure, you should no longer get messages about **op\_** elements because you created them in this lesson. You can ignore the warning message. If you do have error messages, you should correct them; see [Task 3b: Fixing Build Errors](#).



```
Building ----- EXE.exe -----
Executing: "C:\Rhapsody711\Share\etc\GnatMake.bat" EXE.bat rebuild
Invoking MakeFile
Building Dishwasher
gcc -c -I.\DishwasherPkg\ -g -IC:\Rhapsody711\Share\LangAda\src\RiA_Framework\GNAT_Win32 -I-.\DishwasherPkg\display.adb
gcc -c -I.\DishwasherPkg\ -g -IC:\Rhapsody711\Share\LangAda\src\RiA_Framework\GNAT_Win32 -I-.\DishwasherPkg\dishwasherpkg.ads
gcc -c -I.\DishwasherPkg\ -g -IC:\Rhapsody711\Share\LangAda\src\RiA_Framework\GNAT_Win32 -I-.\DishwasherPkg\dishwasher.adb
dishwasher.adb:88:05: warning: variable "singleton" is read but never assigned
gnatbind -a0.\ -a0C:\Rhapsody711\Share\LangAda\src\RiA_Framework\GNAT_Win32 -I- -x display.ali
gnatlink display.ali -g -o EXE.exe
Finished Building Dishwasher

Build Done
```

## Summary

In this lesson, you created a statechart, which identifies the state-based behavior for your dishwasher model. You became familiar with the parts of a statechart and created the following:

- ◆ States and nested states
- ◆ Default connectors
- ◆ Transitions
- ◆ Actions
- ◆ Operation synchronization

You also regenerated code and built your model.

You are now ready to proceed to the next lesson, where you are going to define the message exchange between subsystems and subsystem modules using a sequence diagram.

# Lesson 5: Creating a Sequence Diagram

---

*Sequence diagrams* show structural elements communicating with one another over time. They also identify required relationships and messages. A high-level sequence diagram shows the interactions between actors, use cases, and blocks. Lower-level sequence diagrams show communication between classes and objects.

Sequence diagrams have an executable aspect and are a key *application animation* tool. When you animate the model to see the application's operations, Rational Rhapsody dynamically builds sequence diagrams that record the object-to-object or block-to-block messaging.

## Goals for this Lesson

In this lesson you will learn to perform the following tasks:

- ◆ Create an event in preparation for its use in the sequence diagram
- ◆ Draw a sequence diagram

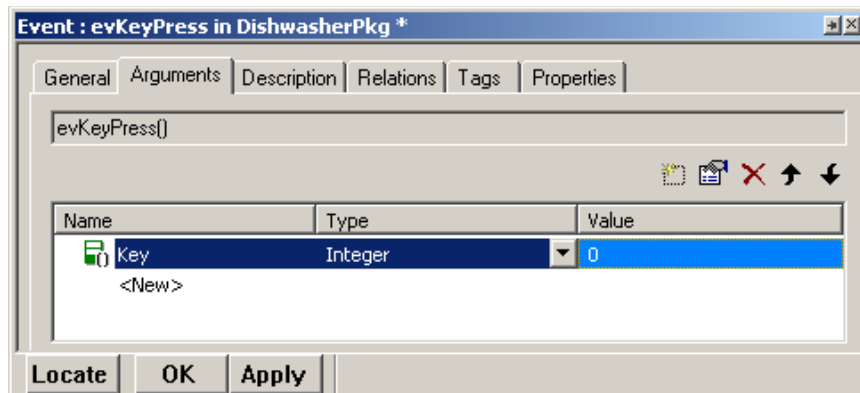
## Exercise 1: Creating the KeyPress Event

To create the “starter” for the dishwasher application, you need to define the KeyPress event, which will be called by the sequence diagram.

### Task 1a: Creating an Event

To create the KeyPress event, perform these steps:

1. Start Rational Rhapsody and open the Dishwasher project if they are not already open.
2. In the Rational Rhapsody browser, expand the **DishwasherPkg** package, right-click **Events**, and select **Add New Event**.
3. Rename the event `evKeyPress` and press **Enter**.
4. Double-click **evKeyPress** to open the Features dialog box.
5. On the **Arguments** tab, click **<New>** to create a blank row for a new argument.
6. To define the `key` argument, fill in the row with the following values, as shown in the following figure:
  - a. For **Name**, type `key`.
  - b. For **Type**, select **Integer** from the drop-down list if necessary. It should be set by default.
  - c. For **Value**, enter 0 (zero).



7. Click **OK**.

## Exercise 2: Creating the Execution Sequence Diagram

Before you start you want to define the system workflow. The dishwasher application needs this high-level workflow:

1. The `main()` loop in `Display` checks for a character every second.
2. The one-second timer message, **op\_tick**, is delivered in every iteration through the loop.
3. Each character relates to a triggered event in the `Dishwasher`, as follows:
  - `s` Dispatches the **op\_start** message to start the processing.
  - `o` Dispatches the **op\_open** message to open the door.
  - `c` Dispatches the **op\_close** message to close the door.

The Execution sequence diagram shows how subsystems interact during the scenario of successfully turning on and off a dishwasher.

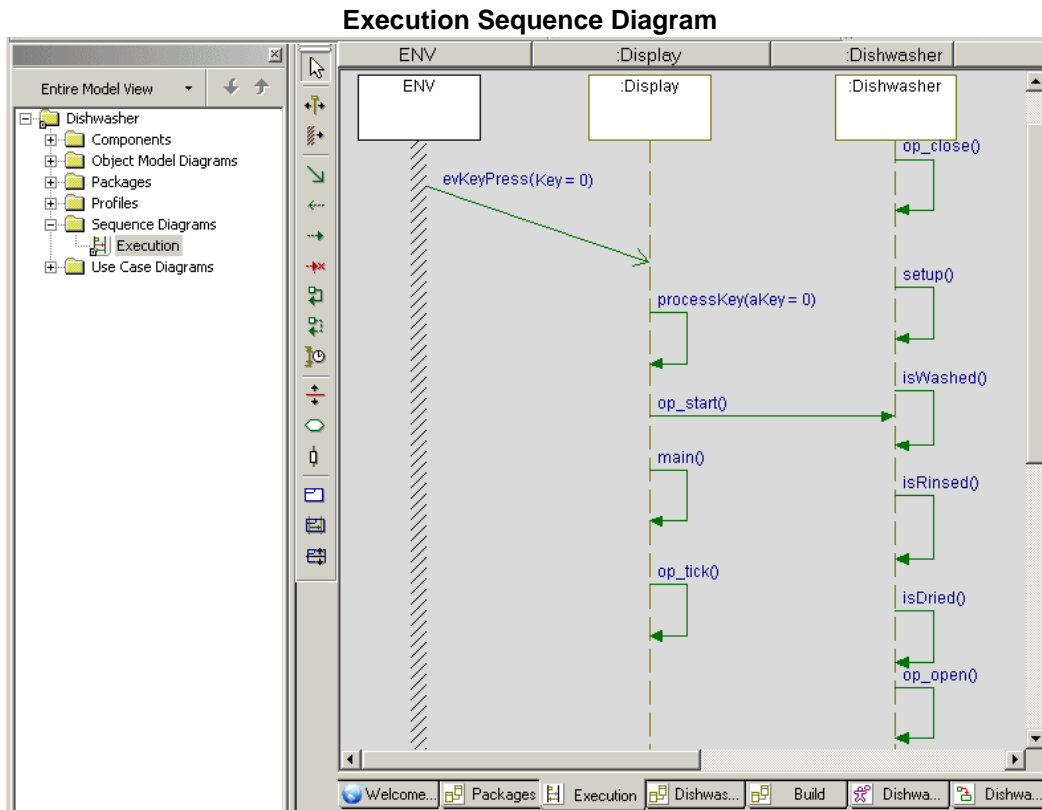
You draw a sequence diagram using the following general steps:

1. Draw the actor lines.
2. Draw classifier roles.
3. Draw messages.
4. Draw interaction occurrences.

This exercise describes each of these steps in detail.

## Lesson 5: Creating a Sequence Diagram

The following figure shows the Execution sequence diagram that you are going to create in this exercise.




Rational Rhapsody separates sequence diagrams into a Names pane and a Message pane. The Names pane contains the name of each instance line or classifier role. The Message pane contains the elements that make up the interaction.

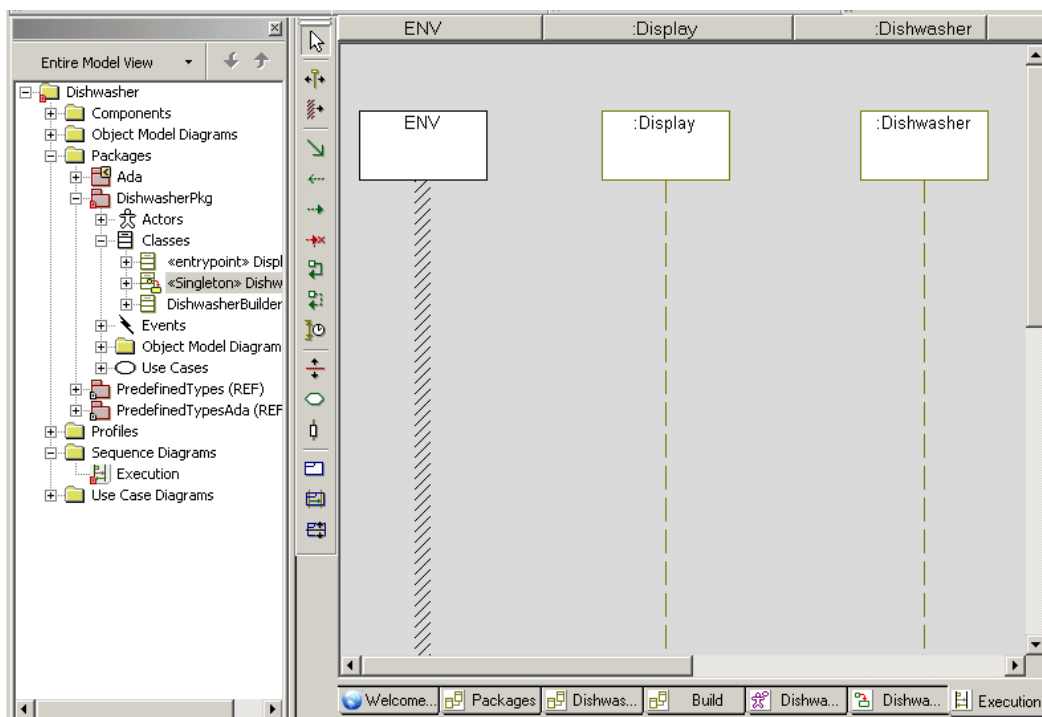


## Task 2a: Creating the Sequence Diagram

To create a sequence diagram, follow these steps:

1. In the Rational Rhapsody browser, right-click the top-level **Dishwasher**, and select **Add New > Sequence Diagram**.
2. Rename the diagram `Execution` and click **OK**.
3. Click the **System Border** button  on the **Drawing** toolbar and click on your sequence diagram. Rational Rhapsody creates an item named **ENV** (for environment) that represents the system border.
4. Drag the **Display** class and the **Dishwasher** class from the Rational Rhapsody browser onto the sequence diagram.



Your diagram should resemble the following figure:



## Task 2b: Creating the Workflow for Your Sequence Diagram

To define the workflow for your sequence diagram, use the events and operations you created in the earlier exercises.

To add the flow instructions for your sequence diagram (use the [Execution Sequence Diagram](#) as a reference), follow these steps:

1. Click the **Message** button  on the **Drawing** toolbar.
2. Click the line for the **ENV** column and then click the line for the **Display** column to create a downward-slanted line, and rename the message `evKeyPress(key = 0)`.
3. Select the **Message** button  and draw a message-to-self line on the **Display** line and rename the message `processKey(aKey = 0)`.

**Note:** For a reactive class, the “message to self” is interpreted as an event. If it is a non-reactive class, it would be viewed as a primitive operation.

4. Use the **Message** button to draw a straight-line message from **Display** to **Dishwasher** and rename it `op_start()`.

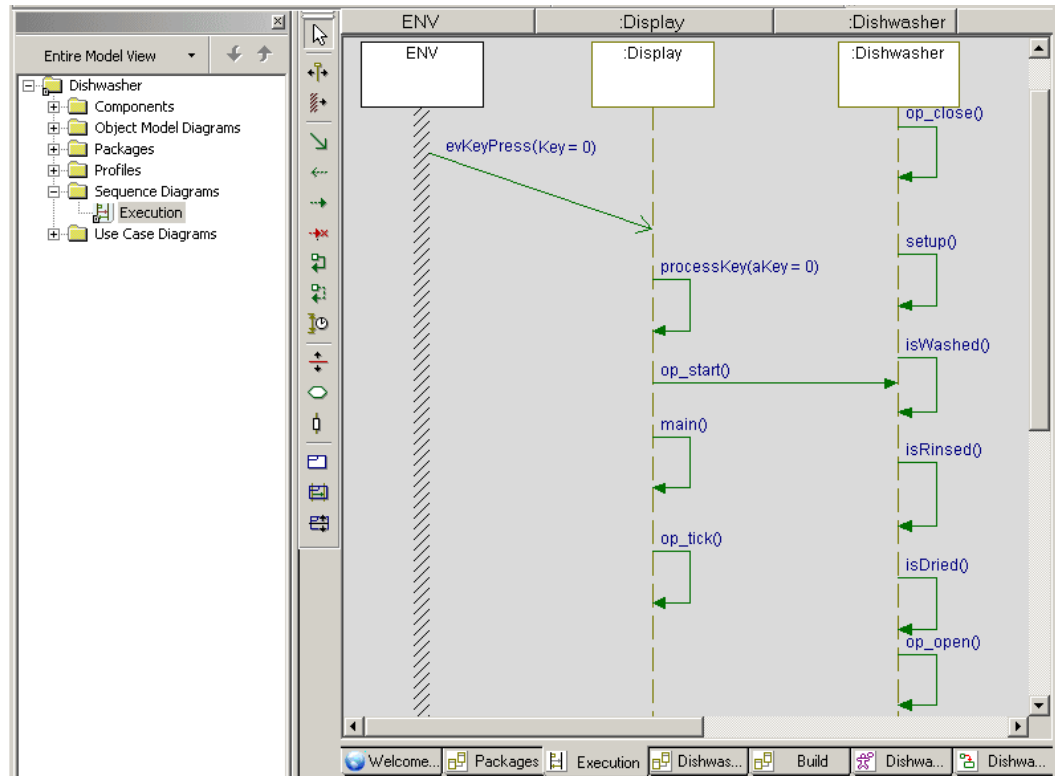
**Note:** The `()` should automatically be entered when you press **Enter**.

5. Use the **Message** button and draw a message-to-self lines of `main()` and `op_tick()` on the **Display** line.
6. Use the **Message** button to add the following to the **Dishwasher** line:

- `op_close()`
- `setup()`
- `isWashed()`
- `isRinsed()`
- `isDried()`
- `op_open()`

7. Save your model.

Your sequence diagram should resemble the following figure:



## Summary

In this lesson, you created a sequence diagram, which show structural elements communicating with one another over time for your dishwasher model. You became familiar with the parts of a sequence diagram and created the following:

- ◆ System border
- ◆ Classifier roles
- ◆ Workflow with messages, events, and time intervals

You are now ready to proceed to the next lesson, where you are going to generate code and build your model.

# Lesson 6: Building and Running the Model

---

Rational Rhapsody uses the following sources to generate code for the model:

- ◆ Project Type or profile selected when you created the project, as described in [Creating a Rational Rhapsody Project](#).
- ◆ Component definition, as described in [Task 1a: Creating a Component](#).
- ◆ Code you entered for operations, as in [Task 1e: Creating Operations](#) and [Task 1e: Adding Ada Operations](#).
- ◆ Communication you defined in a sequence diagram, as described in [Lesson 5: Creating a Sequence Diagram](#).
- ◆ State actions you defined, as described in [Task 1g: Adding Actions to States](#).
- ◆ Compiler and instrumentation mode selections made when defining the configuration, as described in [Task 1c: Creating a Configuration](#).
- ◆ Ada code that Rational Rhapsody automatically generates to support the design you created in the diagrams and from any predefined packages you selected, as described in [Task 2b: Using Predefined Packages](#).

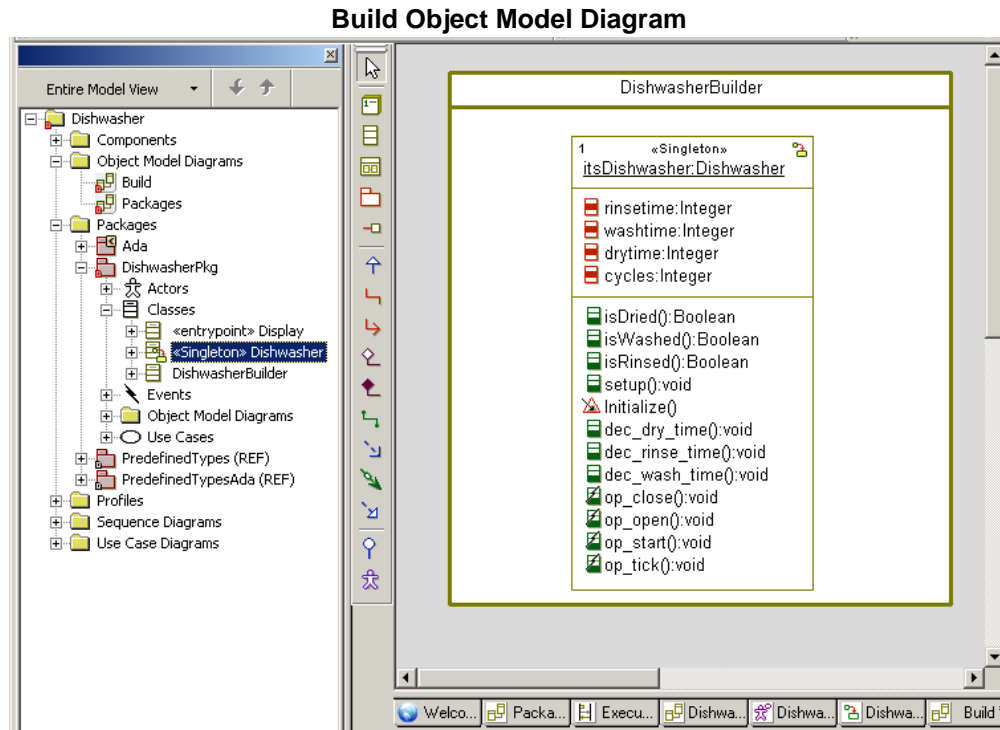
## Goals for this Lesson

In this lesson you are going to create a Build object model diagram and then you are going to generate code and run your model.

## Exercise 1: Creating the Build Object Model Diagram

In this exercise you are going to create a new object model diagram called `Build` and in it a composite class called `DishwasherBuilder`.

The following figure shows the `Build` object model diagram that you are going to create in this exercise.



## Task 1a: Creating the Build Object Model Diagram

To create a new object model diagram, follow these steps:


1. Start Rational Rhapsody and the Dishwasher model if they are not already open.
2. Right-click the **Object Model Diagrams** category in the Rational Rhapsody browser and then select **Add New Object Model Diagram** to open the New Diagram dialog box.
3. Type `Build` and click **OK**.

The new object model diagram displays in the drawing area.

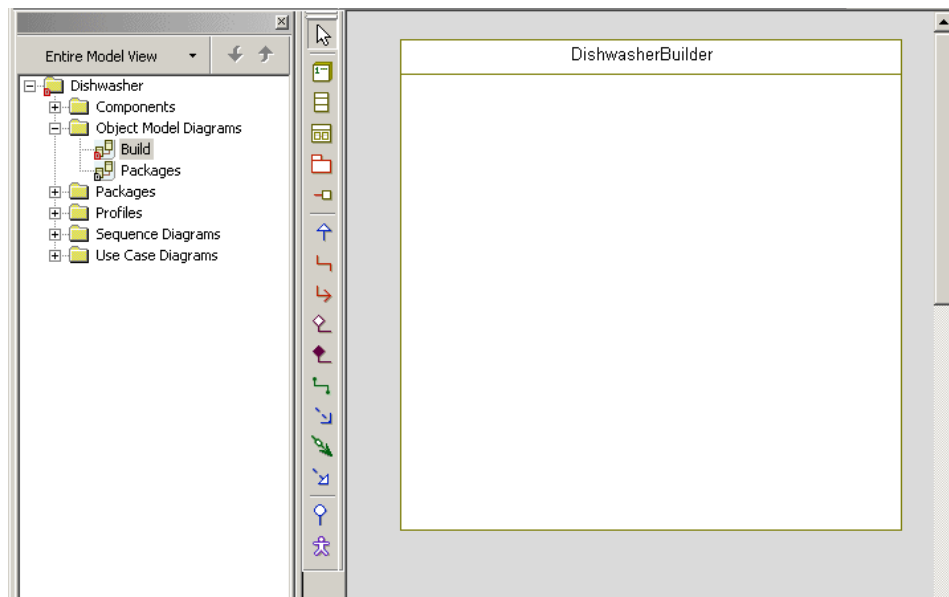
## Task 1b: Creating a DishwasherBuilder Class

To draw the DishwasherBuilder class, use the [Build Object Model Diagram](#) as a reference.

To create a composite class in the Build object model diagram, follow these steps:

1. Select the Composite Class button  from the **Drawing** toolbar and draw the composite class in the Build object model diagram.
2. Rename the composite class `DishwasherBuilder` and then press **Enter**.

Your Build object model diagram should resemble the following figure:

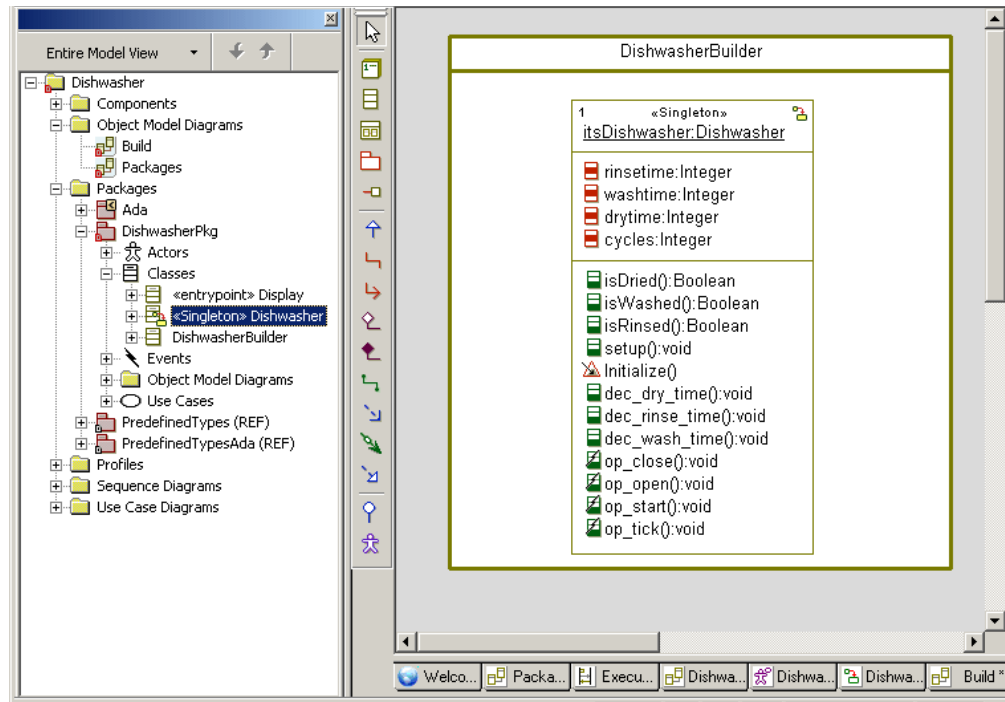


3. Drag the **Dishwasher** class into the **DishwasherBuilder** composite class.

## Lesson 6: Building and Running the Model

4. Right-click the **DishwasherBuilder** class and select **Make an Object**.
5. Resize the object on the diagram so that you can see all of its attributes and operations.

Your Build object model diagram should resemble the following figure:



6. Save your project.



## Exercise 2: Generating Code and Building Your Model

In this exercise, you generate code and build your model.

As part of [Lesson 4: Creating a Statechart](#), you generated code and successfully built your Dishwasher model at that point in time. In this exercise, you will create another configuration for your **EXE** component, which you will use later to animate your model.

### Task 2a: Creating Another Configuration

As you learned earlier, a component can contain many configurations. In this task, you are going to create a configuration called **HostAnimated** by copying the **Host** configuration you already have in the **EXE** component.

To create another configuration by copying a current one, follow these steps:

1. In the Rational Rhapsody browser, expand the **EXE** component and the **Configurations** category.
2. Hold down the **Ctrl** key while you use the mouse to drag the **Host** configuration onto the **EXE** component.
3. Double-click the **Host\_copy** configuration to open the Features dialog box.

**Note:** You might find it useful to arrange it so that you can see the Rational Rhapsody browser while you have the Features dialog box opened.

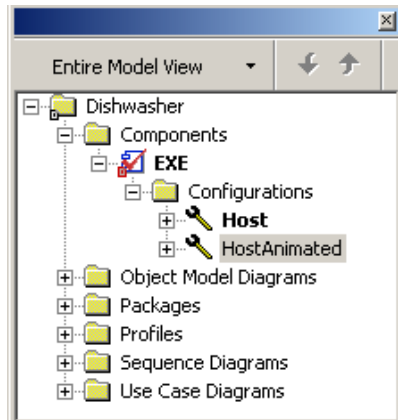
4. On the **General** tab, in the **Name** box, replace `Host_copy` with `HostAnimated`.

## Lesson 6: Building and Running the Model

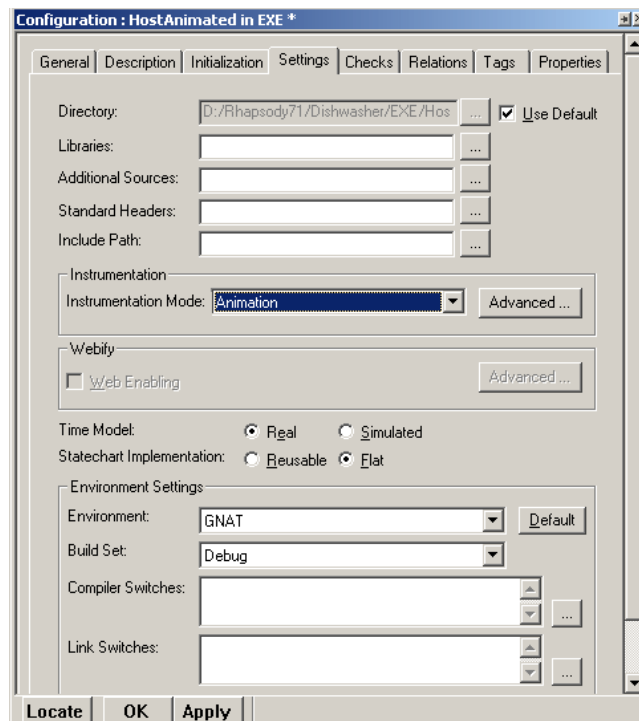
---

5. Click **Apply**.

Notice the name change on the Rational Rhapsody browser. Your browser should resemble the following figure:



6. On the **Setting** tab, in the **Instrumentation Mode** box, select **Animation**, as shown in the following figure:



7. Notice that Rational Rhapsody sets the values in the **Environment Settings** group based on the compiler settings you configured during installation of the Rational Rhapsody product. This example uses a system with the GNAT compiler, as shown in the above figure.
8. On the **Initialization** tab, select the **Explicit** option button if it is not already selected.
 

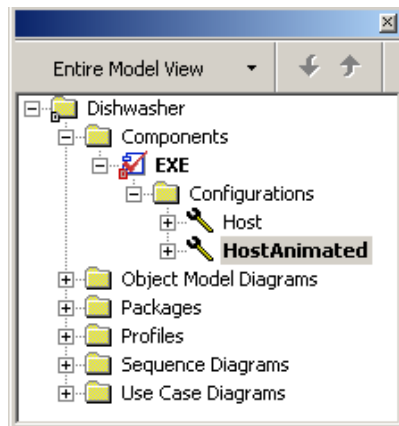
**Note:** Because we are using the entrypoint stereotype in this model, you do not have to select which instance to initialize on this tab. Click the + signs and notice that none of the check boxes are selected, which is as it should be for this model.
9. Click **OK**.
10. Save your model.


## Task 2b: Generating Code

To generate code and the instances necessary for animation, follow these steps:

1. Set **HostAnimated** as the active configuration:
  - Right-click **HostAnimated** and select **Select as Active Configuration**, or
  - Select **HostAnimated** from the drop-down list on the **Code** toolbar.

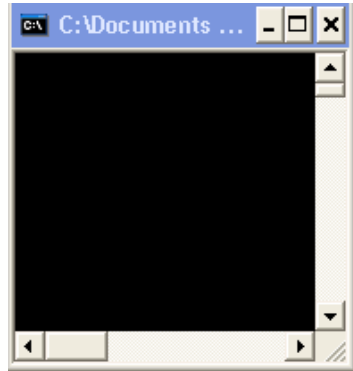
Once **HostAnimated** is set as the active configuration it should appear in boldtype on your browser, as shown in the following figure:



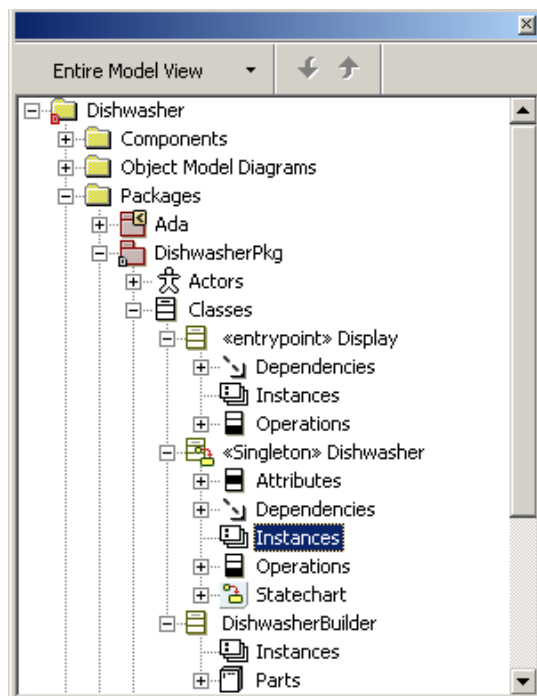
2. From the **Code** toolbar, click the **GMR** button . This generates code, builds the configuration, and runs the executable image for the active configuration. It is the same as choosing **Code > Generate/Make/Run**.
3. If you are asked if you want to create the `HostAnimated` directory, click **Yes**.

4. If you are asked if you want to run the executable, click **Yes**. (You might not be asked.)

An Application window opens, as shown in the following figure:



5. In the browser, check to be certain that instances were created.
  - a. Expand the **DishwasherPkg** package, the **Classes** category, and then each class.
  - b. See that there is an **Instances** category under the **Display**, **Dishwasher**, **DishwasherBuilder** classes, as shown in the following figure:



6. If you are successful, continue with [Exercise 3: Running Your Model](#). If not, continue with [Task 2c: Troubleshooting the Build](#).

## Task 2c: Troubleshooting the Build

If the instances were not created, there are two typical reasons:

- ◆ You might not have an Ada compiler installed on your machine or the current version of the compiler available on your computer for Rational Rhapsody to use. If that is the problem, an error message displays at the bottom of the Output window. Be certain the correct compiler for your code is accessible to Rational Rhapsody to correct the problem. This is usually set up when Rational Rhapsody is installed. Refer to the Rational Rhapsody Release Notes for the supported Ada compilers.
- ◆ One or more steps might have been skipped or entered with typographical errors during the design process. Examine the error message and return to the section of the tutorial covering that feature. A typographical error prevents the system from recognizing relationships and items.

If these two problems did cause the error, follow these steps:

1. Double-click the error in the message list in the Output window.
2. The system displays the code containing the error in the drawing area. Examine the code and click on the diagram tabs and browser to research the problem and make the necessary changes.
3. Continue with [Task 2d: Roundtripping](#).

## Task 2d: Roundtripping

*Roundtripping* is an on-the-fly method used to update the model quickly with small changes entered to previously generated code. If you have made small changes in the code in the previous section, follow these steps to incorporate it into the model:

1. Choose **Code > Roundtrip > Host**.
2. Look at the results display in the Output window.

However, roundtripping should not be used for major changes in the model that would require the model to be rebuilt. Do not use roundtripping to incorporate changes to any of the following in a model:


- ◆ Packages
- ◆ Dependencies
- ◆ Stereotypes
- ◆ States
- ◆ Transitions
- ◆ Component / Configuration information

## Exercise 3: Running Your Model

In this exercise, you are going to run your model.

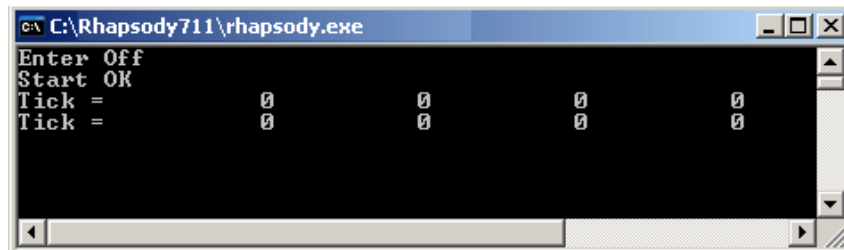
### Task 3a: Running your Dishwasher Model

To run your dishwasher model, follow these steps:

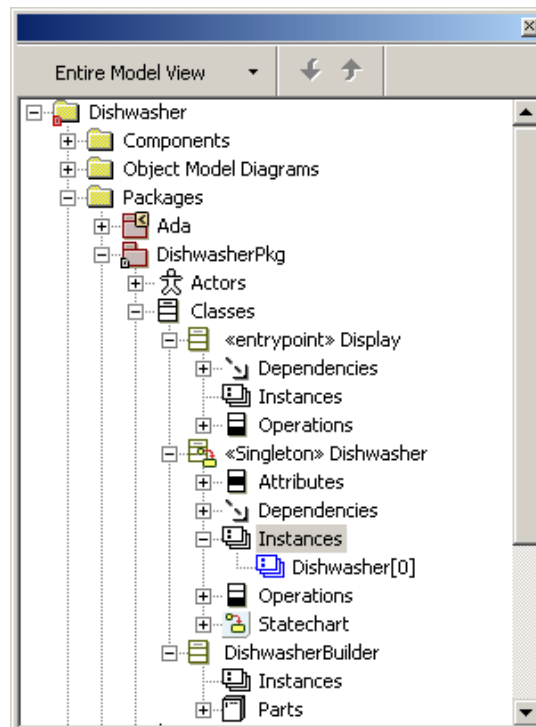
1. If needed, click the GMR button . You might not need to if you successfully did this in [Task 2b: Generating Code](#).
2. Arrange your desktop so that you can view the Rational Rhapsody browser, drawing area, and the Output window; and the Application window.

3. Click the Go button . The following occurs:

- On the Application window, the first lines of output display, as shown on the following figure:

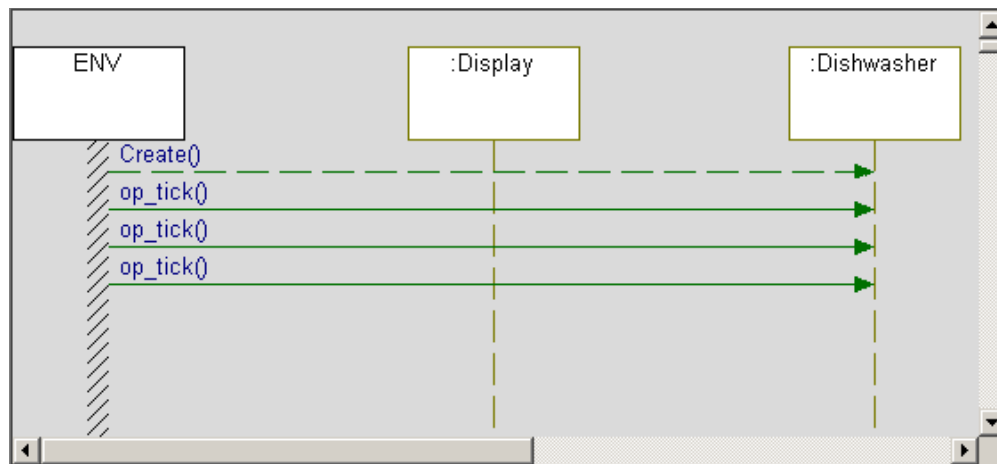


- On the Rational Rhapsody browser, a relationship item (**Dishwasher[0]**) appears for the **Dishwasher** class, as shown on the following figure:





– Your sequence diagram should resemble the following figure:



4. To end the program, click the Stop Make/Execution button .

This stops all processes and closes the Application window.

## Summary

In this lesson, you built the Build object model diagram, set up your project for animation, generated code, built your model, and ran your executable for your model.

You are now ready to proceed to the next lesson, where you are going to animate your model.



# Lesson 7: Animating Your Application

---

*Animation* is the observable execution of behaviors and associated definitions in the model. Rational Rhapsody animates the model by executing the code generated, with instrumentation, for classes, operations, and associations. Once you start model animation, you can open animated diagrams, which let you observe the model as it is running and perform design-level debugging. You can step through the model, set and clear breakpoints, inject events, and generate an output trace.

It is good practice to test the model incrementally using model execution, which you have practiced in earlier lessons. You can animate pieces of the model as it is developed. This gives you the opportunity to determine whether the model meets the requirements and find defects early on. Then you can test the entire model. In this way, you iteratively build the model, and then with each iteration perform an entire model validation.

## Goals for this Lesson

In this lesson you are going to animate your application by stepping through the program, invoking commands, and setting breakpoints.

## Exercise 1: Animating your Application

In this exercise you are going to animate your model, step through it, invoke commands for it, set breakpoints, and exit animation mode.

### Task 1a: Starting Animation

Because you compiled the application with animation instrumentation in the previous lesson, when the application starts, it connects to the Rational Rhapsody application via a TCP/IP socket whose number is set in the `rhapsody.ini` file.

**Note:** This task assumes you are starting after having completed [Task 3a: Running your Dishwasher Model](#) from [Lesson 6: Building and Running the Model](#).

To start animation, follow these steps:

1. Choose **Code > Run EXE.exe**.

Rational Rhapsody displays the animation toolbar, which includes tools that enable you to control and test the application. In addition, an Application window opens.



2. Position and resize the Rational Rhapsody window and the Application window so that both are easily visible on your desktop.
3. Notice that the following output panes are displayed at the bottom of the Rational Rhapsody window:
  - ◆ **Animation** shows run-time messages from the application.
  - ◆ **Call Stack** displays the logical call stack of the executing model at the design level, rather than the code level.

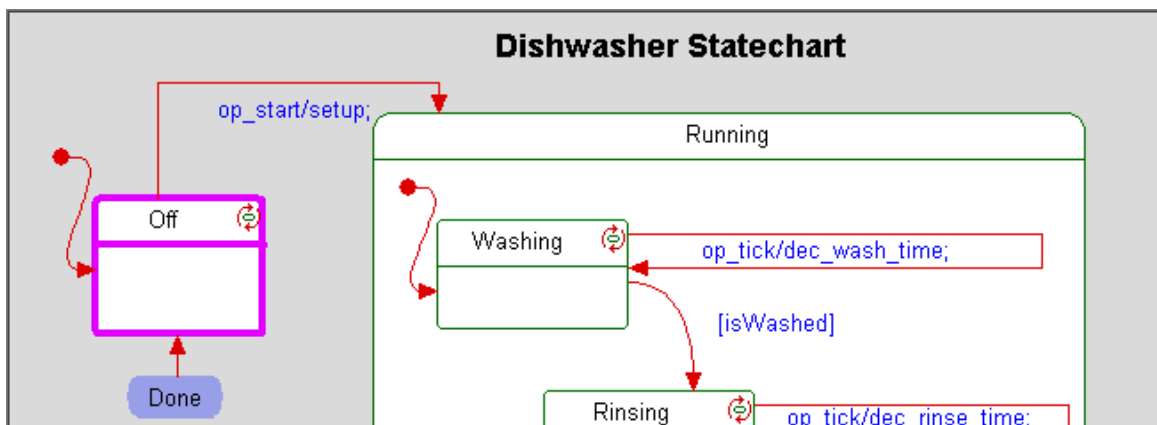
You can adjust the horizontal sliders between the panes as needed to view the contents. These windows are dockable, so you can move them out of the Rational Rhapsody client area if you want to increase the viewable area for animations. Simply click and hold the double bars at the left edge of a dockable window, move the window, and drop it where you want.

## Task 1b: Viewing the Animated Statechart

As you do this task, in addition to looking at the Call Stack tab, look at what is happening on the Application window too. To do this, you have to arrange your desktop so that you can view Rational Rhapsody and the Application window clearly.

To step through the model animation, follow these steps:

1. Click the Go Step button . The Call Stack displays nothing.
2. Click Go Step button . The Call Stack displays the message `Initialize()`.
3. Click Go Step button. The `Initialize()` message is removed from the Call Stack.
4. Click Go Step button. The message `Dishwasher[0] -> Start Behavior` is displayed in the Call Stack and the browser shows the new instance, **Dishwasher[0]**. To see it in the browser, expand the **Instances** category under the **Dishwasher** class.
5. In the browser, right-click the **Dishwasher[0]** instance and then select **Open Instance Statechart**. Rational Rhapsody displays an animated version of your Dishwasher statechart.
6. Click Go Step button. The message is removed from the Call Stack, and your statechart should resemble the following figure:



**Note:** Magenta denotes what is active and olive denotes what is inactive.


7. Continue with [Task 1c: Invoking Commands for your Program](#).

## Task 1c: Invoking Commands for your Program

You can invoke commands for your program with the use of the Application window because this model used triggered operations.

Note that if the model used events instead, you could generate events for your model. For more information about events, refer to the *IBM Rational Rhapsody User Guide*.

To invoke commands for your program, follow these steps:

1. If not already, arrange your desktop so that you can view the Rational Rhapsody window and the Application window clearly.
2. Click the Go button  on the **Animation** toolbar.
3. Notice on the Application window that your program starts running. Because **op\_tick** is set for every second, there is a tick every second, as shown in the following figure:

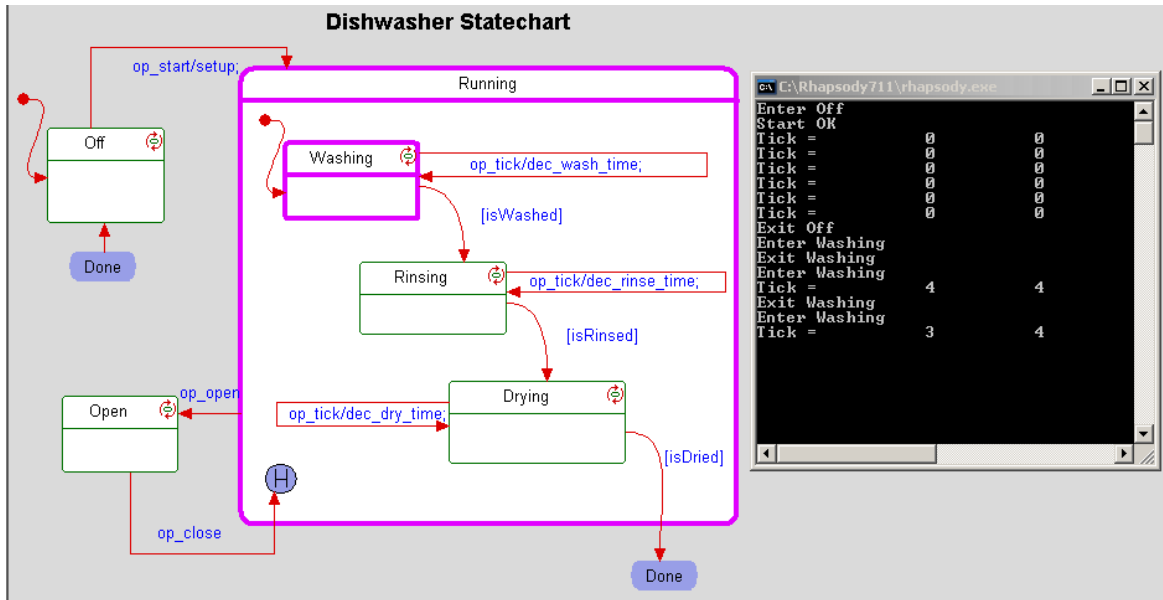




```
C:\Rhapsody711\rhapsody.exe
Enter Off
Start OK
Tick =      0      0      0      0
Tick =      0      0      0      0
```

4. With focus on the Application window, press **s** on your keyboard.

- Notice what happens on your statechart and the Application window.

On your statechart, in the **Running** state, the dishwasher cycles through the **Washing** (as shown in the following figure) **Rinsing**, and **Drying** states, until it goes to the **Off** state.



**Note:** Magenta denotes what is active and olive denotes what is inactive. Depending on what timer functions might be coded for a program, the color changes might happen rapidly. You can click the Animation Pause button  on the **Animation** toolbar to pause the animation. Click the Go button  again to continue.

- Once you are in the **Off** state, press `s` again to start the dishwasher.





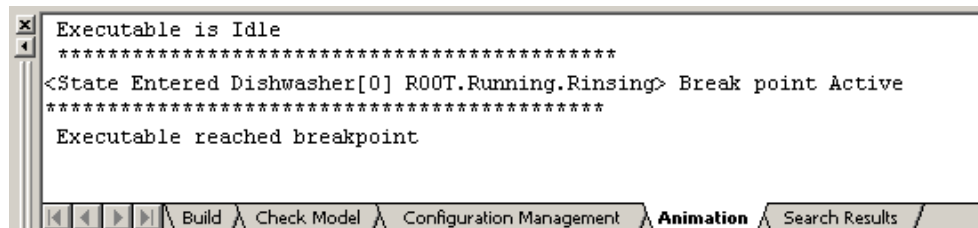
9. Optional. For another view, watch what happens on your animated sequence diagram when you do this task.
10. Let the program get to the **Off** state (as seen on your animated statechart).

## Task 1d: Setting Breakpoints


*Breakpoints* enable you to stop the execution at a point that is entirely under your control so you can examine the state of the application. For more information about breakpoints, refer to the *IBM Rational Rhapsody User Guide*.

To set a breakpoint, follow these steps:

1. Right-click the **Rinsing** state and select **Add Breakpoint** to open the Define Breakpoint dialog box.
2. Click the **Select** button to open the Instances Selection dialog box.
3. Select **Dishwasher[0]** and then click **OK**
4. On the Define Breakpoint dialog box, leave the default values in the **Reason** and **Data** boxes and click **OK**.
5. On the Application window, press **s** on your keyboard. Notice that when your program enters **Rinsing** state, execution stops and a message displays on the Output window, as shown in the following figure:



6. To disable the breakpoint:



- a. Click the Breakpoints button  on the **Animation** toolbar to open the Breakpoints dialog box.
- b. Select the breakpoint you want to disable and then click the **Disable** button.
- c. Click **OK** to close the Breakpoints dialog box.

**Note:** You can use the Breakpoints dialog box to see a list of the available breakpoints; plus add, enable, disable, and delete breakpoints.

7. Click the Go button  to restart the program.

### Task 1e: Quitting Animation

To end the animation session, follow these steps:

1. If necessary, click the Animation Pause button .
2. Click the Quit Animation button .
3. Click **Yes** to confirm ending the animation session.

The message `Animation session terminated` displays on the Animation tab of the Output window.

Note that you can also click the Stop Make/Execution button .

## Summary

In this lesson, you became familiar with animation and animated your model. You performed the following:

- ◆ Generated code, built the model, and ran the program
- ◆ Animated the statechart and sequence diagram
- ◆ Stepped through and invoked commands to your application and saw it progress through states and pass messages

# Index

## Symbols

\_rpy file 13  
\_RTC directory 13

## A

Actions 85  
Active configuration 69  
Activity diagrams 5, 77  
Actors 25, 26, 30  
Ada code examples 3  
    main operation 50  
    setup operation 49  
Animation 107, 115, 116  
    breakpoints 121  
    configuration 69, 105  
    generating code 69  
    invoking commands for your program 118  
    output windows 116  
    quitting 122  
    running the application 115  
    starting 116  
    statecharts 117  
    tab 122  
    viewing 117  
Application 115  
    running 111, 115  
    workflow 98  
Application window 108, 112, 118, 119  
Arguments tab 94  
Association 33  
Asynchronous 88  
Attributes 43  
    viewing in diagram 46  
Autosave 10

## B

Backup 10  
Boundary boxes 30  
Breakpoints 121  
Browser 15, 17, 23  
Build 18  
Build object model diagram 102  
Build tab 70, 91

Building the model 71, 92

## C

Call Stack 116  
Categories 17  
Classes 40  
    create operations 44  
    predefined for Ada 56  
    singleton 42  
Code  
    examples 3  
    generated from 101  
    roundtripping 111  
Code generation 69, 107  
    creating configurations 69, 105  
    debugging 70, 71, 72  
    source files 71  
Collaboration diagrams 5  
Compilers 107, 110  
Component diagrams 5  
Components 66  
    creating 66  
    creating configurations 69, 105  
    default description 66  
    features 67  
Configurations 69, 105  
    creating animation 69, 105  
    default 69  
    Host 69  
    HostAnimated 105  
    set as active 69, 107  
Connectors 79  
    default 81  
    diagram 79  
    history 79  
    transitions 83  
Constructors 62  
Creating  
    animation configuration 69, 105  
    components 66  
    dishwasher project 6  
    object model diagram 39  
    sequence diagram 97  
    statechart 77  
    use case diagram 25

### D

- Debugging 70, 71, 72
- Default
  - component 66
  - configuration 69
- Dependencies 40
- Dependency 40
- Deployment diagrams 5
- Description tab 20
- Diagram connectors 79
- Diagrams 4, 5
  - Build 102
  - Dishwasher 27, 38
  - Dishwasher object model diagram 37, 38
  - Dishwasher statechart 76
  - Dishwasher use case diagram 25
  - Execution sequence diagram 93, 96
  - object model 38, 76
  - UML 5
- Directory structure 55
- Dishwasher 1
  - animating 115, 117
  - creating 6
  - creating statecharts 77
  - instance 117
  - object model diagram 37, 38
  - opening 11
  - statechart 75, 76
  - use case diagram 25, 27
- Display options 46, 47
- Docking the Features dialog box 23
- Domains 9
- Drawing 16
  - area 15, 18
  - default connectors 81
  - diagram connectors 79
  - history connectors 79
  - object model diagrams 39
  - sequence diagrams 97
  - statecharts 77
  - toolbar 18
  - toolbars 15
  - transition connectors 83
  - use case diagrams 28

### E

- ehl file 13
- Elements, external 71
- Entrypoint 53
- Environment settings 107
- Error messages 110
- Errors 110
- Event history file 13
- Events 94
  - naming conventions 14

- Sequence diagrams 98
- evKeyPress event 94
- Executable configuration 69, 105
- Execution sequence diagram 93, 96
- External elements 71

### F

- Features dialog box 19
  - Apply and OK buttons 19
  - Description tab 20
  - docking 23
  - General tab 20
  - keeping open 19
  - moving 23
  - Properties tab 21
  - Relations tab 21
  - tabs 20
  - Tags tab 21
- Files 13
  - code generation 71
  - log 13
  - project 12
  - source 71
- Folders 13

### G

- General tab 20
- Generate 101
- Generated source files 71
- Generating code for animation 69
- Graphical user interface 15
- Guards 84
- Guidelines 14

### H

- History connectors 79
- Host configuration 69, 107
- HostAnimated configuration 105

### I

- Implementation code 50
- Instance area 96
- Instances 117
- Interfaces 14

### K

- KeyPress event 94

### L

- Legacy code 71

Linux 6  
Log 18  
files 13

## M

Message pane 96  
Messages 110  
Model 1  
building 71, 92, 107  
naming conventions 14  
running 111  
stepping through 118  
system behavior 77  
troubleshooting 110

## N

Names pane 96  
Naming conventions 12, 14

## O

Object model diagrams 5, 39  
composite classes 40  
Dishwasher 37, 38  
simple classes 40  
viewing attributes 46  
viewing operations 47  
Opening  
project 11  
Rational Rhapsody 6  
Operations 44  
changing synchronization 88  
names 14  
naming conventions 14  
setup 48  
viewing in diagram 47  
Output window 15, 18, 70, 71, 72, 110  
check model tab 18  
log tab 18

## P

Packages 9, 17, 55  
adjust dependency 60  
establish dependency 59  
predefined 56  
setting dependency 60  
stereotypes 54  
storing separately 55  
SubsystemsPkg 39  
Panels  
Message 96  
Name 96  
Profiles 7

Project files 12, 13  
Project folder 17  
Project node 17  
Project profiles 7  
Project subfolders 13  
Project types 7  
Projects 7  
creating 6, 7  
directory structure 55  
files 13  
opening 11  
saving 10  
saving as 1  
Properties 68  
change directory scheme 55  
dependency 60  
tab 21

## R

Rational Rhapsody 4  
autosave 10  
backup 10  
browser 17, 23  
closing 6  
configuration 69, 105  
drawing area 18  
Drawing toolbar 18  
events 94  
exiting 6  
Features dialog box 19  
GUI 15  
guidelines 14  
interface 15  
naming conventions 12  
Output window 18  
project types 7  
sample models 1  
starting 6  
toolbars 16  
UML diagrams 5  
Rebuilding the application 70, 71, 72  
Regenerating code 70, 71, 72  
Relations tab 21  
Repository directory 13  
Requirements 25  
rhapsody.ini file 116  
Roundtripping 111  
rpy file 13  
Running application 111

## S

Sample models 1  
Sequence diagrams 5, 97  
define flow 98  
events 98

## Index

---

- Execution 93
- instance area 96
- Message pane 96
- Names pane 96
- set border 97
- Set as active configuration 69, 107
- Simple classes 40
- Singleton 42
- Source diagram connectors 79
- Source files 71
- Specialized profiles 7
- Starting animation 116
- Statecharts 5, 75, 77
  - Build 76
  - creating 77
  - default connector 81
  - Dishwasher 75
  - drawing 77
  - guard transitions 84
  - transitions 83
  - triggered operations 88
- States
  - adding actions 85
  - drawing 78
- Stepping through the program 117, 118
- Stereotypes 40
  - entrypoint 53
  - singleton 42
  - usage 41
- Structure diagram 5
- Subfolders 13
- Subsystems 9
- Synchronization 88

## T

- Tabs
  - build 18
  - check model 18
  - log 18
- Tags tab 21
- Target diagram connectors 79
- Toolbars 15, 16, 18
- Transitions 83, 84
- Triggers 84
- Troubleshooting 110

## U

- Units 10
- Use case diagrams 5, 25
  - actors 30
  - boundary boxes 30
  - Dishwasher 25, 27
  - use cases 31
- Use cases 31, 32

## V

- vba file 13
- Viewing
  - animation 117
  - attributes 46
  - operations 47

## W

- Windows 6
- Workflow 95, 98