Pathfinder Solutions

PathMATE™ for IBM Rational Rhapsody

Quick Start Guide

Version 8.0.3

September 1, 2009

PathMATE™ Series

Pathfinder Solutions
www.pathfindermdd.com
+1 508-568-0068

Table of Contents

Overview

This document is provided as a first step in learning how to use PathMATE with the IBM Rational Rhapsody UML environment to build platform independent models and transform them to an executable system. The instructions in this guide are detailed with screen shots and other aids to keep you moving forward quickly.

The Platform-Independent Model Driven Development approach for building systems software and the PathMATE environment for automating this approach are specifically intended to address the key challenges faced when building complex, high-performance systems. Unfortunately we cannot have you quickly build a highly complex system that shows off all of PathMATE's capabilities – at least not as your first step. Instead we will start you with SimpleOven. This is a very simple example system that covers some of the key elements of a PathMATE system. The goal of this Quick Start Guide is to expose you to some of the core aspects of modeling, code generation, system construction and execution as quickly as possible. After completing this Guide please explore the more sophisticated example systems provided with PathMATE, such as ExperimentControl, to gain a more complete understanding of how real-world systems are constructed.

How to Use this Guide

If you are not yet familiar with the PathMATE toolset, please continue to read this Overview section.

If you have not installed the PathMATE toolset on your computer, follow the Download PathMATE section to download the software from the PathTECH area of www.pathfindermdd.com.

Continue to learn how PathMATE works by completing the walk-though tutorial beginning with the "Model a Simple Oven" section.

Whether you have created hundreds of models and systems or none at all, you can follow this tutorial. Learn quickly how to use PathMATE to develop an executable system.

Audience

The *Quick Start Guide* is for software modelers who want to learn how to design high performance and embedded systems with PathMATE. It's helpful but not essential to have some familiarity with the IBM Rational Rhapsody toolset.

Additional Resources

The Pathfinder Solutions website at www.pathfindermdd.com offers information on products, services, and model-driven approaches and technology. After completing this walkthrough tutorial, you'll want to explore the extensive collection of whitepapers available as PDF downloads. Direct email support is available from support@pathfindermdd.com.

Eclipse

Eclipse is supported by an international open source effort and is freely downloaded by thousands of professionals. PathMATE provides feature plugins to Eclipse which extend its functionality using the framework as a basis for interoperability. For further information on Eclipse, go to **www.eclipse.org**. This tutorial will often refer to common framework elements reflecting this heritage; e.g., "Eclipse Menu" or "Eclipse Toolbar". The tutorial assumes that you are familiar with basic Eclipse terminology including the specialized terms, "View", "Editor", "Perspective", and "Navigator".

Conventions

The *Quick Start Guide* uses these conventions:

- **Bold** is for clickable buttons and menu selections.
- *Italics* is for screen text, path and file names, and other text that needs special emphasis.
- `Courier` denotes code, or text in a log or a batch file.
- A **Note** contains important information, or a timesaving tip.

✂

- The scissors icon marks text that you copy from this document and paste elsewhere.

What You'll Need

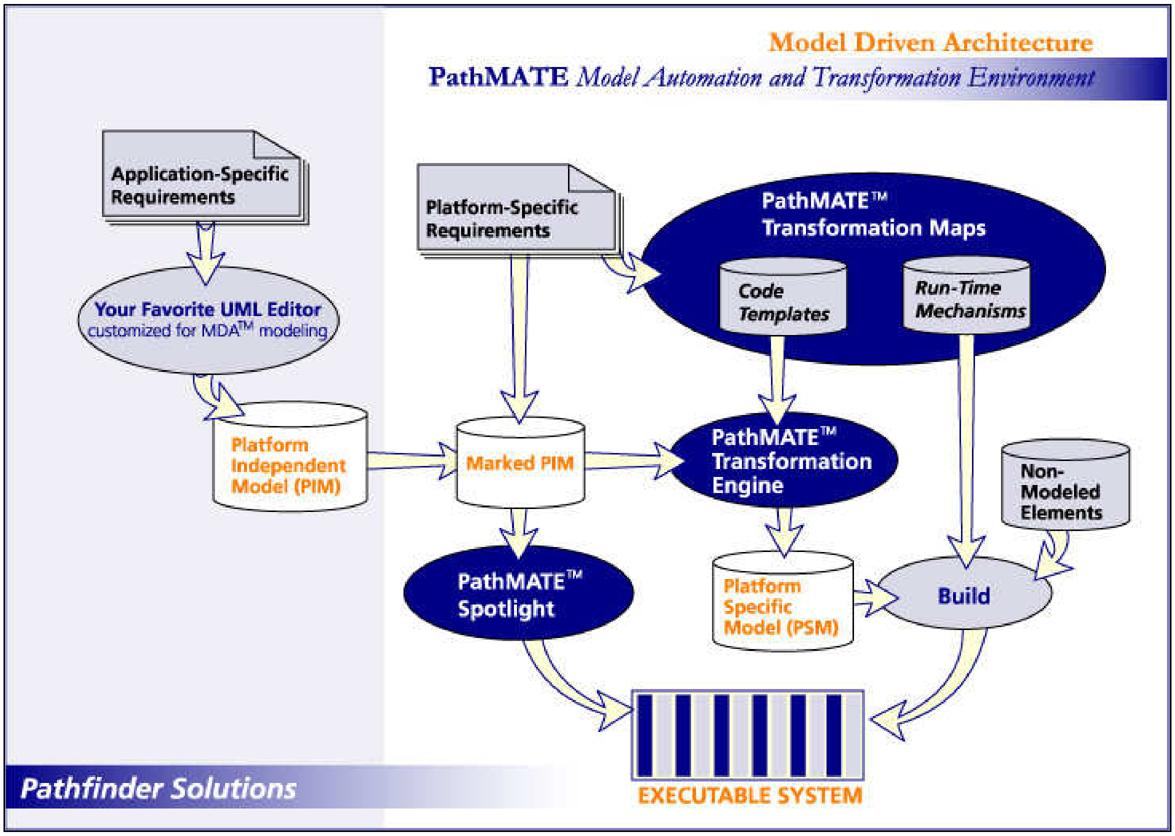
To complete the steps in this guide, you'll need the following software on your computer:

- Microsoft Windows 2000 or Windows XP Professional Edition
- PathMATE Version 8.0.1+
- IBM Rational Rhapsody version 7.1.0, 7.2.0, 7.3.0, 7.4.0, or 7.5.0
- Microsoft Visual C++ version 6, 7, 8, or 9

PathMATE Platform Independent MDD Toolset

Through its architectural focus, platform independence and advanced automation, PathMATE is the leading MDD environment for embedded, real-time systems development with its PathMATE PI-MDD toolset. This powerful technology executes and tests platform-independent UML models and automatically transforms them into high performance C, C++, and Java systems that are specifically optimized for resource-constrained embedded environments. Over years of rigorous refinement in several industries, PathMATE tools have proven their value in rapid and effective embedded systems development.

The PathMATE Model Automation and Transformation Environment includes all the tools required to transform your models into high-performance embedded systems:


Model Driven Architecture
PathMATE Model Automation and Transformation Environment
Application-Specific Requirements
Your Favorite UML Editor
customized for MDA™ modeling
Platform Independent Model (PIM)
Platform-Specific Requirements
PathMATE™ Transformation Maps
Code Templates
Run-Time Mechanisms
Marked PIM
PathMATE™ Transformation Engine
Non-Modeled Elements
PathMATE™ Spotlight
Platform Specific Model (PSM)
Build
EXECUTABLE SYSTEM
Pathfinder Solutions

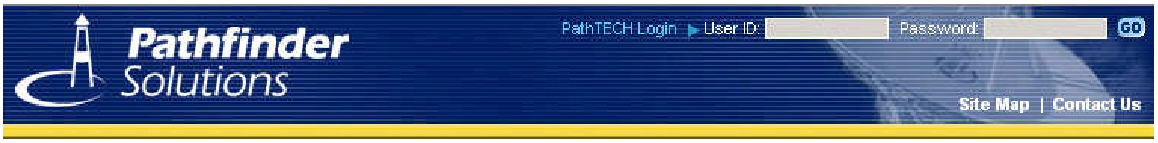
The PathMATE PI-MDD Toolset is comprised of three parts which work together to turn your models into executable embedded systems:

- *Transformation Engine* – generates embedded software applications from your application-specific platform-independent models.
- *Transformation Maps* – guide the conversion process from model to platform-specific executable code. You can choose from standard off-the-shelf C, C++, and Java maps. To meet the demands for other languages or specific platform needs, our consultants can work with you to develop customized high-performance maps.
- *Spotlight* – provides the most advanced model testing environment available to verify and debug your application logic.

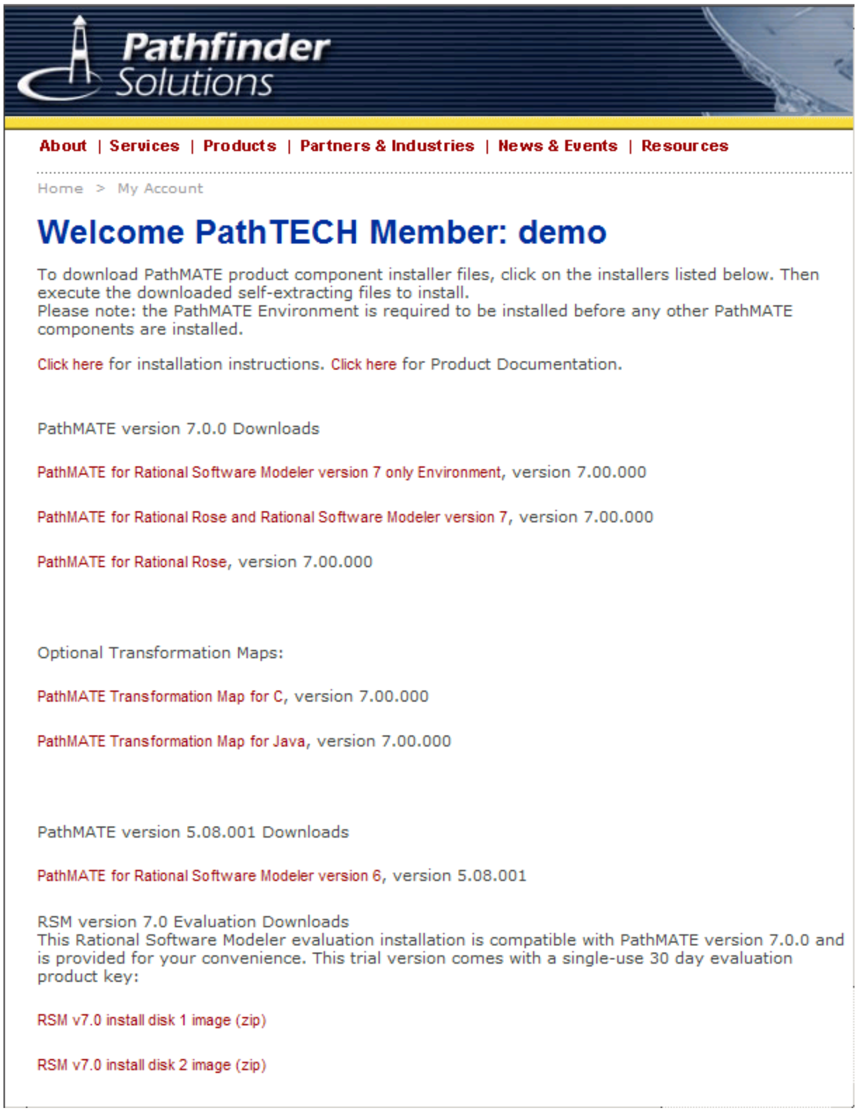
No other MDD transformation environment offers a more open or configurable set of development tools, designed to meet the requirements of embedded systems engineers.

Download and Install PathMATE

1. Navigate to http://www.pathfindermdd.com. The Pathfinder Solutions home page opens.
2. Log into PathTECH at the top of the main web page with user ID *demo*. Please contact your account manager to obtain your password.



On the PathTECH download page you'll be able to select the appropriate PathMATE product downloads:


Pathfinder
Solutions
About | Services | Products | Partners & Industries | News & Events | Resources
Home > My Account

Welcome PathTECH Member: demo

To download PathMATE product component installer files, click on the installers listed below. Then execute the downloaded self-extracting files to install.
Please note: the PathMATE Environment is required to be installed before any other PathMATE components are installed.

Click here for installation instructions. Click here for Product Documentation.

PathMATE version 7.0.0 Downloads

PathMATE for Rational Software Modeler version 7 only Environment, version 7.00.000

PathMATE for Rational Rose and Rational Software Modeler version 7, version 7.00.000

PathMATE for Rational Rose, version 7.00.000

Optional Transformation Maps:

PathMATE Transformation Map for C, version 7.00.000

PathMATE Transformation Map for Java, version 7.00.000

PathMATE version 5.08.001 Downloads

PathMATE for Rational Software Modeler version 6, version 5.08.001

RSM version 7.0 Evaluation Downloads
This Rational Software Modeler evaluation installation is compatible with PathMATE version 7.0.0 and is provided for your convenience. This trial version comes with a single-use 30 day evaluation product key:

RSM v7.0 install disk 1 image (zip)

RSM v7.0 install disk 2 image (zip)

3. Download the following files:
    - *PathMATE version 8.0.1 + only Environment.*
    - *PathMATE Transformation Map for Java, version 8.0+ (if you will be generating Java implementation code.)*

After installation, see the file C:/pathmate/doc/PathMATE_with_Visual_ Studio_Express.pdf for specific instructions on how use Microsoft Visual Studio 2005 Express Edition.

Create Eclipse Project

The table below, outlines the model-building part of the tutorial, and introduces some PathMATE terminology:

Action	Term	Definition of Term
Create	Eclipse Project	Eclipse projects are containers for related development resources, often with a one-to-one correspondence with file system objects.
Create	UML Project	Rhapsody UML projects are containers for related elements that represent UML Model. These elements are represented with various file system objects, located in a central location.
Create	System Model	A PathMATE refinement of a Rhapsody project container for our overall logical architecture.
Add	Domain Model	A PathMATE refinement of a UML package logical architectural component of our system.
Create	Domain Services	Input/Output/Event interfaces exposed by a Domain Model.
Complete	(System) Domain Chart	Drawing showing the Domains and their and relationships that comprise the System Model.
Complete	(Domain) Class Diagram	Drawing showing the interfaces, classes, and their relationships that comprise a Domain.
Complete	Application Domain	Architecture intended to solve a specific design need. In this case, control a simple microwave oven.
Create	Door State Machine	A logical specification of states, possible transitions between states, actions that can be performed, and the conditions or events under which transitions and actions can occur. In this case, the description of door open/closed processing.
Create	Light State Machine	In this case, the description of light on/off processing.

Add	Entry Actions	Detailed specification of actions that occur on the *entry* into a state. Other, possible times are while *in* a state, or when *exiting* a state.

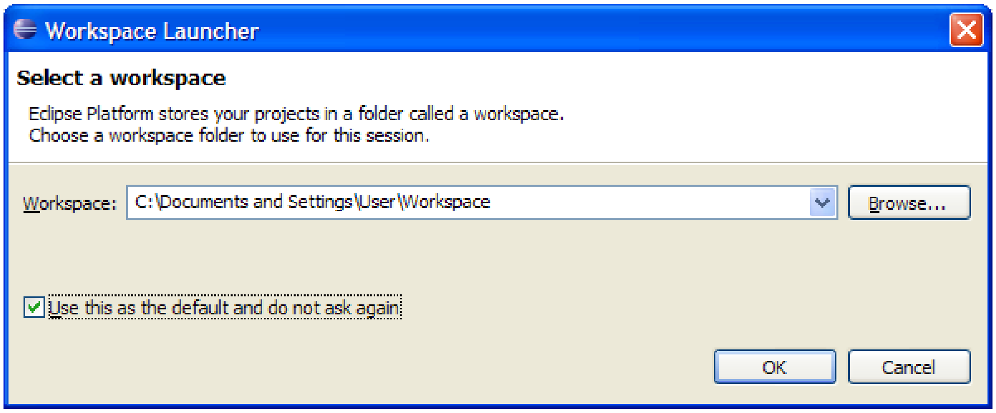
Task 1 Create a Modeling Project

In Eclipse projects represent a file Structure. We will create a new Project and set it up to store both the Rhapsody project, that will be created in the Model A Simple Oven Section, and the PathMATE support files created in the Prepare for Transformation Section.

PROCEDURE: Use Eclipse File → New to setup project

1. Launch Eclipse through using the PathMATE Launcher from , Start > All Programs > Pathfinder Solutions > Eclipse.
2. From the Workspace Launcher dialog select to use the default workspace or browse to another location on the file system to house the project information.

   Please Note, to avoid being prompted for a workspace each time Eclipse is launched select "Use this as the default and do not ask again."



Workspace Launcher

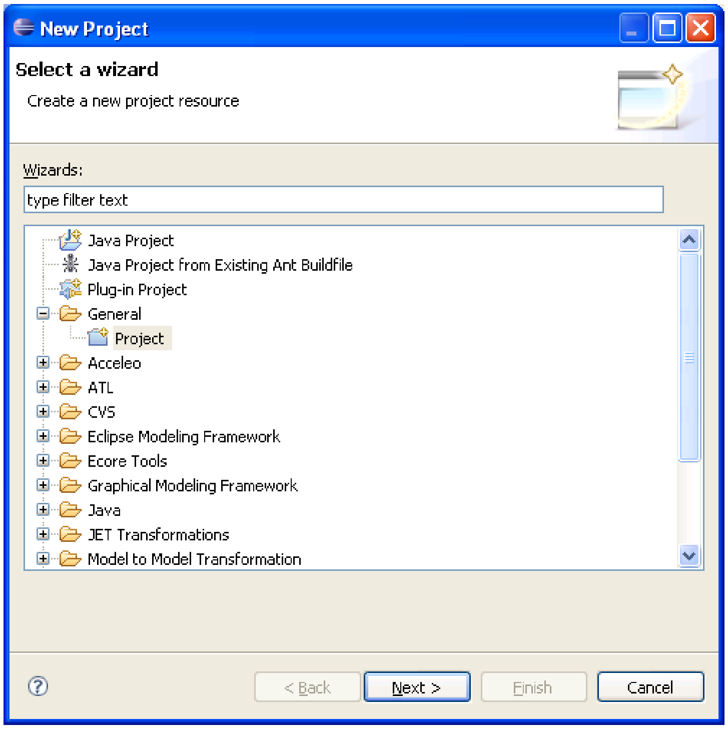
Select a workspace

Eclipse Platform stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Documents and Settings\User\Workspace

☑ Use this as the default and do not ask again

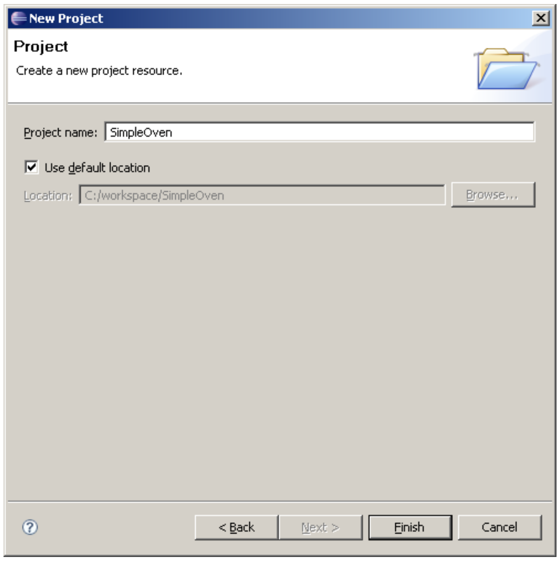
3. Once the work space has loaded, select **File → New → Project**, launching the Project Wizard Selection Dialog.
4. Under **General** select Project.



Select a wizard

Create a new project resource

5. Press **Next**, bringing up the Project Name Screen.
6. Specify the project name as *SimpleOven*.



Project

Create a new project resource.

Project name: SimpleOven

☑ Use default location

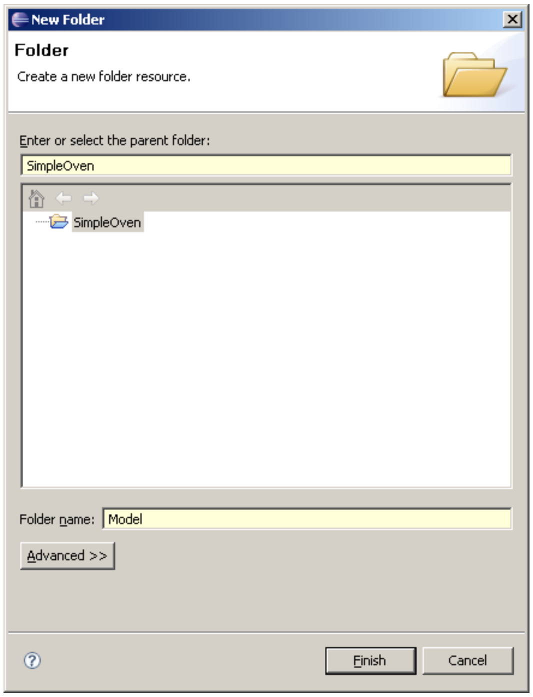
Location: C:/workspace/SimpleOven

7. Press **Finish**, to create the new Project.

PROCEDURE: Add a folder for model information

1. Once the project has been created, select **File>New>Folder**, launching the New Folder Dialog.
2. Select the *SimpleOven* project as the location for the folder

3. Specify the name of the Folder as *Model*.



New Folder

Folder

Create a new folder resource.

Enter or select the parent folder: SimpleOven

SimpleOven

Folder name: Model

4. Press **Finish**

Model A Simple Oven

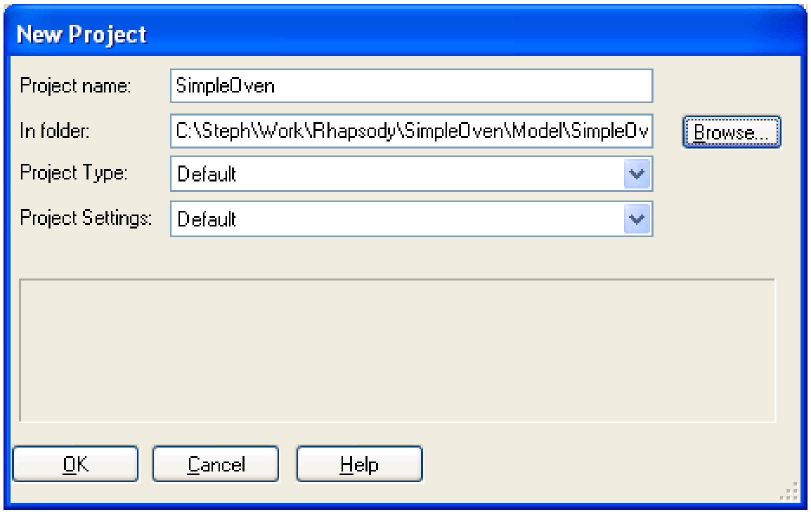
Task 1: Create and Set Up a UML Project

In Rhapsody all model data is contained within a project. We will create a new blank project and set it up for use with PathMATE.

PROCEDURE: Use Rhapsody File → New to setup project

1. Launch Rhapsody and select **File → New**.
2. Specify the project name as *SimpleOven*.
3. Specify the location of the Model Folder created in "PROCEDURE: Add a folder for model information" to store the project files.

   This folder is located in the Eclipse Project directory, which is located un the workspace directory specified when Eclipse was launched, see "PROCEDURE: Use Eclipse File → New to setup project".
4. Specify the Project Type as *Default*.
5. Select **OK**.



New Project

Project name: SimpleOven
In folder: C:\Steph\Work\Rhapsody\SimpleOven\Model\SimpleOv
Project Type: Default
Project Settings: Default

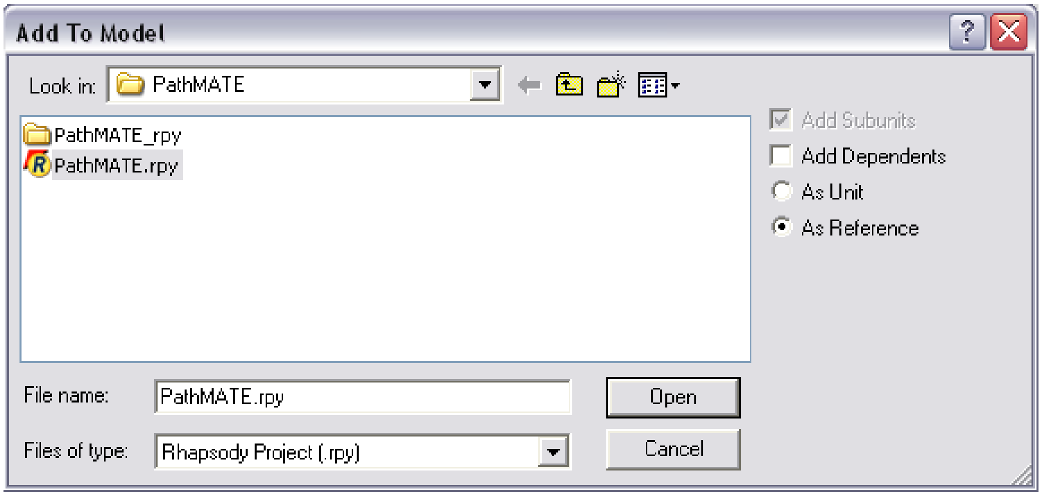
6. Click Yes on the Rhapsody dialog to confirm the creation of the model directory.

For versions of Rhapsody before 7.5.0, there is no Project Settings field. You now have an empty project to work in.

PROCEDURE: Import PathMATE Profile and SoftwareMechanisms Domain

In order to take advantage of PathMATE's features, we will import the PathMATE Profile, which contains PathMATE specific types and special stereotypes containing PathMATE settings. We will also import the SoftwareMechanisms domain, which contains many useful services that can be called from components you create. Once this procedure is complete, you will be able to browse the services offered by the SoftwareMechanisms domain on your own.

1. Select **File → Add To Model…**
2. Browse to *C:\pathmate\config\Rhapsody\PathMATE*.
3. Select the *PathMATE.rpy* file.
4. Select the **As Reference** radio button.
5. Make sure the **Add Dependents** check box is cleared.



Add To Model

Look in: PathMATE

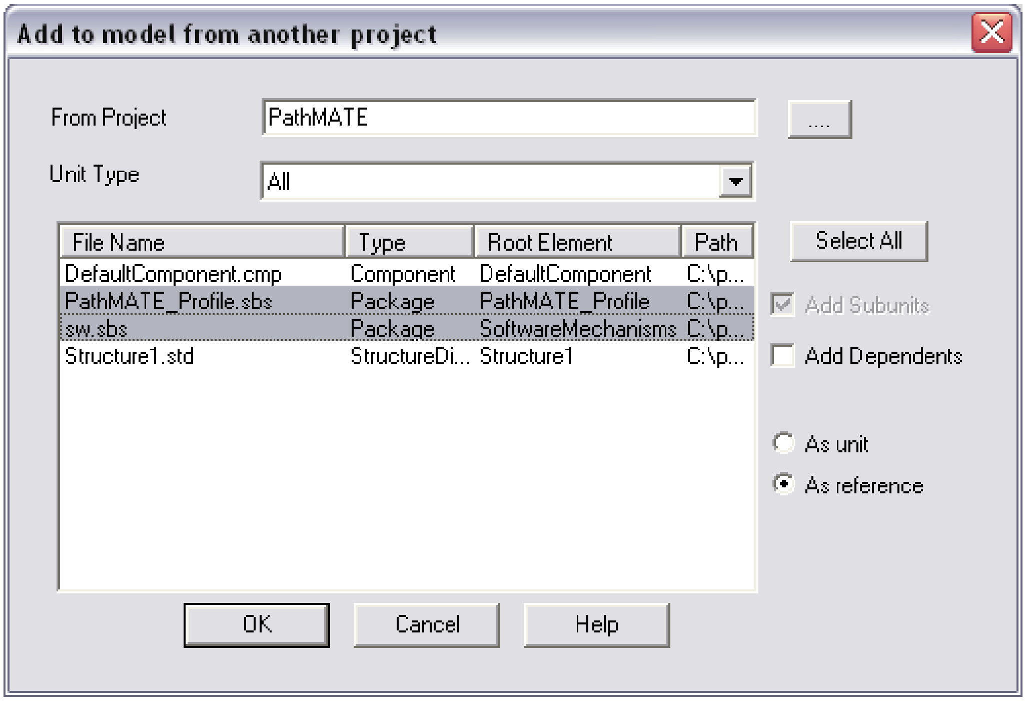
PathMATE_rpy
PathMATE.rpy

☑ Add Subunits
□ Add Dependents
○ As Unit
● As Reference

File name: PathMATE.rpy

Files of type: Rhapsody Project (.rpy)

6. Select **Open**.
7. Select the *PathMATE_Profile.sbs* and *sw.sbs*.
8. Select the **As reference** radio button.



Add to model from another project

From Project: PathMATE

Unit Type: All

File Name	Type	Root Element	Path
DefaultComponent.cmp	Component	DefaultComponent	C:\p...
PathMATE_Profile.sbs	Package	PathMATE_Profile	C:\p...
sw.sbs	Package	SoftwareMechanisms	C:\p...
Structure1.std	StructureDi...	Structure1	C:\p...

☑ Add Subunits

☐ Add Dependents

○ As unit

● As reference

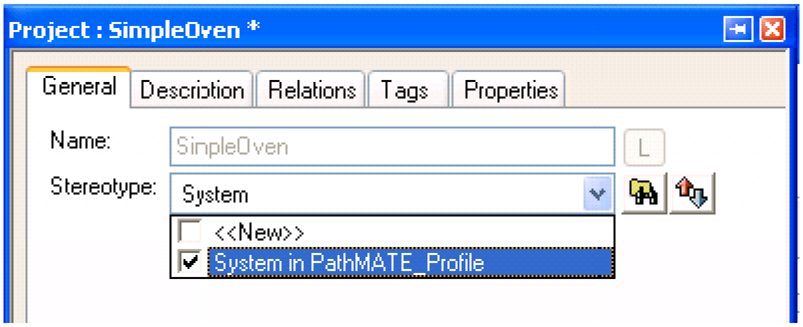
9. Select **OK** to import the *PathMATE profile* and the *SoftwareMechanisms domain* into the model.

We now have the PathMATE elements that will be used in creating the SimpleOven model.

PROCEDURE: Set Up the System

Before PathMATE can use this model, we need to identify this project as a system model.

1. Open up the features window for the project by double clicking on the **SimpleOven** element at the top of the project explorer.
2. Apply the *PathMATE System stereotype* by checking **System in PathMATE_Profile** in the Stereotype dropdown menu.



Project : SimpleOven *

Name: SimpleOven

Stereotype: System

☐ <<New>>

☑ System in PathMATE_Profile

3. Select the **Tags** tab to see the properties available from this stereotype. See the PathMATE documentation for more information about what each of these does.
4. Select **OK**.

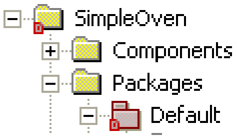
Task 2: Add a Modeled Domain

Domains are logical components – the primary top-level building block throughout the PathMATE modeling process. A domain is a separate conceptual "world," inhabited by a distinct set of elements that rely on each other but do not rely on any elements outside of their own domain.

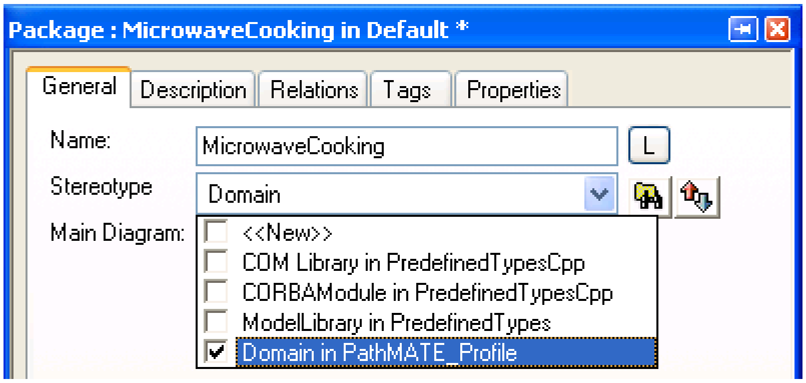
A "modeled" or "analyzed" domain is one that we have created a platform independent model for. The model is complete and executable, in the sense that all behavior will be included in the model.

PROCEDURE: Add a new Package and Apply the Domain Stereotype

1. Under the Packages folder, right click on the **Default** package in the project explorer.



2. Select **Add New → Package**.
3. Type *MicrowaveCooking* as the name of the package.
4. Double click on the new package to bring up the features dialog.
5. Select the **General** tab if it is not already selected.
6. In the Stereotype dropdown menu, select the PathMATE Domain stereotype: **Domain in PathMATE_Profile**.


Package : MicrowaveCooking in Default *

Name: MicrowaveCooking

Stereotype: Domain

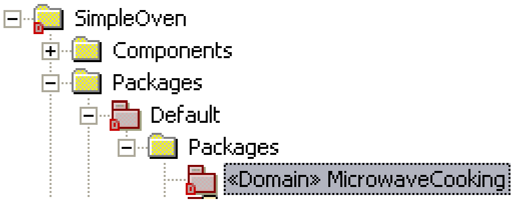
Main Diagram:
General
Description
Relations
Tags
Properties
L
☐ <<New>>
☐ COM Library in PredefinedTypesCpp
☐ CORBAModule in PredefinedTypesCpp
☐ ModelLibrary in PredefinedTypes
☑ Domain in PathMATE_Profile

7. Select **OK**.

PROCEDURE: Set up the Domain Interface for the MicrowaveCooking Domain

Each domain provides a set of services called a “domain interface”, which allows domains to communicate and make requests of each other. Now that the domain exists, we can set up its interface.

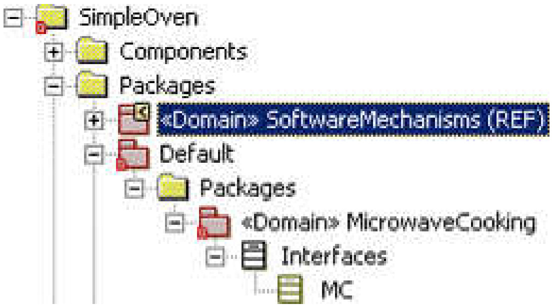
1. Right click on **<<Domain>>MicrowaveCooking** in the project explorer.



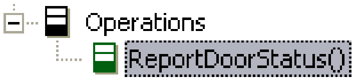
2. Select **Add New → Interface**.
3. Name the interface *MC, an abbreviation of the domain name.*

PROCEDURE: Add a Service to the Domain Interface

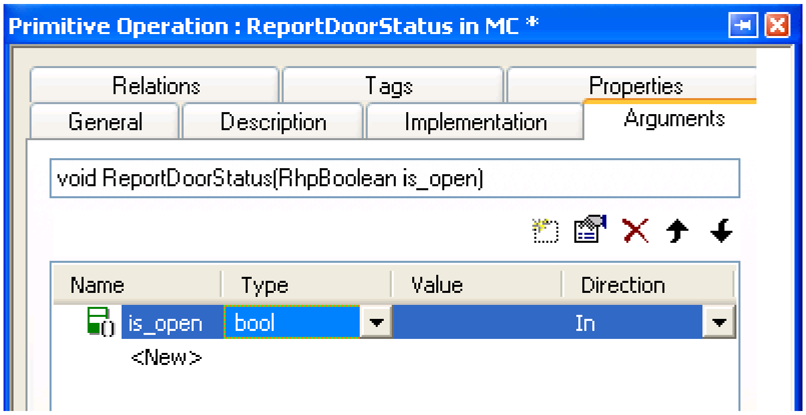
1. In the project explorer pane, right click on the new **MC** interface in the *MicrowaveCooking domain*.



2. Select **Add New → Operation**.
3. Name the operation *ReportDoorStatus*.



4. Double click on the new **ReportDoorStatus** domain service to bring up the features dialog.
5. Select the **Arguments** tab.
6. Add a new argument by clicking **<New>** in the name column.
7. Call the new argument *is_open*.
8. In the type dropdown menu for the new *is_open* argument, select **bool**. (PathMATE will also recognize RhpBoolean as a Boolean. See the PathMATE for Rhapsody Modeling Guide for a complete list of mapped types).



void ReportDoorStatus(RhpBoolean is_open)

Name	Type	Value	Direction
is_open	bool		In
<New>			

9. Select **OK**.

PROCEDURE: Add Action Language for ReportDoorStatus Domain Service

PathMATE uses a Platform-independent Action Language (PAL) to specify the behavior of model elements. PAL directly accesses/manipulates model-level constructs for simplicity and convenience and is transformed to self-optimized implementation code for optimal runtime efficiency and performance. For more information about PAL specifics, see the PathMATE documentation.

1. Double click on the new **ReportDoorStatus** domain service to bring up the features dialog.
2. Select the **Implementation** tab of the *ReportDoorStatus* operation's features window.
3. Copy the text below into the implementation tab's text field.

```
Ref<Door> door = FIND CLASS Door;
IF (door != NULL)
{
    IF (is_open)
    {
        GENERATE Door:IsOpen() TO (door);
    }
    ELSE
    {
        GENERATE Door:IsClosed() TO (door);
    }
}
```

TIP: If you are reading this using Adobe Acrobat, click the Select Text tool in your Acrobat toolbar. Then select the text, copy it, and paste using Control-V in the Action Language view.

4. Select **OK** to close the operation's features.

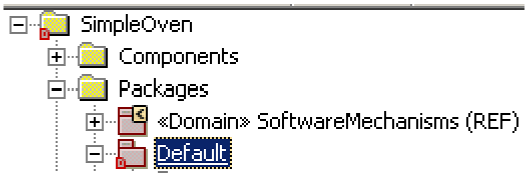
Task 3: Create a Realized Domain

ExternalDeviceControl is a realized domain that provides domain services for the system. These procedures explain how to abstract the interface to the *ExternalDeviceControl* services at the Platform Independent Model (PIM) level for use by modeled domains.

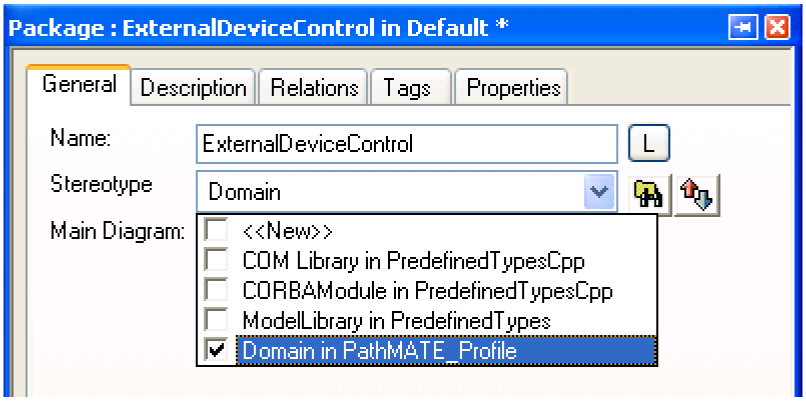
The first step is to create a new Domain, which is by default a Modeled Domain. Then, the Domain's **"***UseAsExternal***"** Property is changed from *False* to *True*, making the Domain Realized. (For those of you already familiar with PathMATE, the Rhapsody "*UseAsExternal*" property has been mapped to the PathMATE "Analyzed" property. For a complete list of all mapped PathMATE properties, see the PathMATE for Rhapsody Modeling Guide.)

PROCEDURE: Create the ExternalDeviceControl Domain and Set the UseAsExternal Property

1. Right click on the **Default** package in the project explorer.



2. Select **Add New → Package**.
3. Type *ExternalDeviceControl* as the name of the package.
4. Double click on the new package **ExternalDeviceControl** to bring up the features dialog.
5. Select the **General** tab if it is not already selected.
6. In the Stereotype dropdown menu, select **Domain in PathMATE_Profile** stereotype.

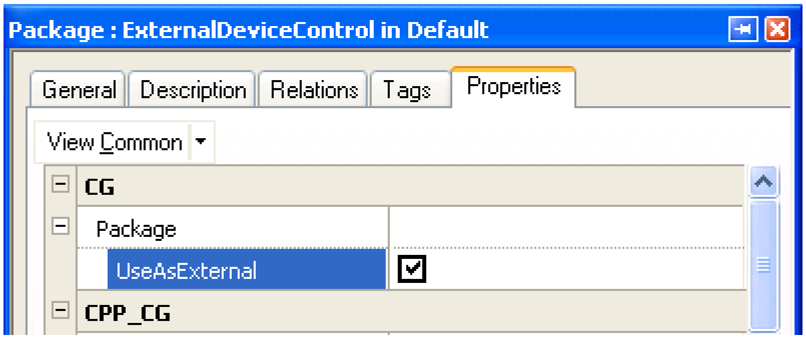

Package : ExternalDeviceControl in Default *

Name: ExternalDeviceControl

Stereotype: Domain

General
Description
Relations
Tags
Properties
L
Main Diagram:
☐ <<New>>
☐ COM Library in PredefinedTypesCpp
☐ CORBAModule in PredefinedTypesCpp
☐ ModelLibrary in PredefinedTypes
☑ Domain in PathMATE_Profile

7. Select the **Properties** tab.
8. Expand *CG* and *Package* if not already expanded.
9. Check the **UseAsExternal** box.



Package : ExternalDeviceControl in Default

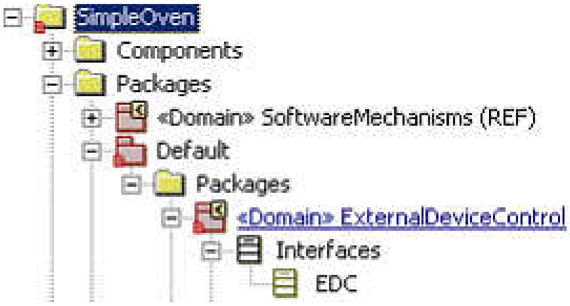
CG
Package
UseAsExternal ☑
CPP_CG

10. Select **OK**.

The *ExternalDeviceControl* domain now has a yellow arrow icon on it, denoting that it is an external package.

PROCEDURE: Set up the Domain Interface for the ExternalDeviceControl Domain

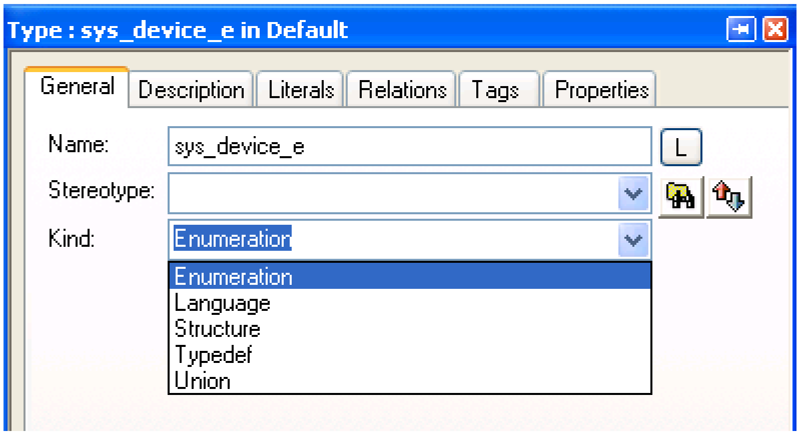
1. Right click on **<<Domain>>ExternalDeviceControl** in the project explorer.


SimpleOven
Components
Packages
«Domain» SoftwareMechanisms (REF)
Default
Packages
«Domain» ExternalDeviceControl
Interfaces
EDC

2. Select **Add New → Interface**.
3. Name the interface *EDC*.

PROCEDURE: Add a UML Enumeration Type to the System's Public Types

1. Right click on the **Default** package in the project explorer.
2. Select **Add New → Type**.
3. Name the new type *sys_device_e*.
4. Double click on the newly created **sys_device_e** type to open the features dialog.
5. In the Kind dropdown menu select **Enumeration** (NOTE: PathMATE supports only Enumeration and Typedef type kinds.).



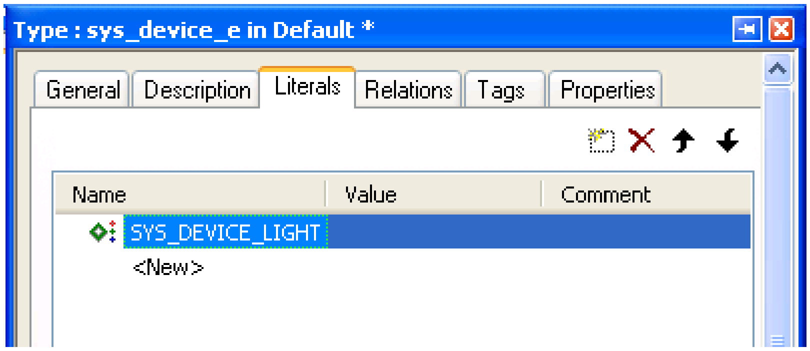
Type : sys_device_e in Default

Name: sys_device_e

Stereotype:

Kind: Enumeration

6. Select the **Literals** tab.
7. Create a new literal by clicking on **<New>** in the name column.
8. Name the new literal *SYS_DEVICE_LIGHT*.



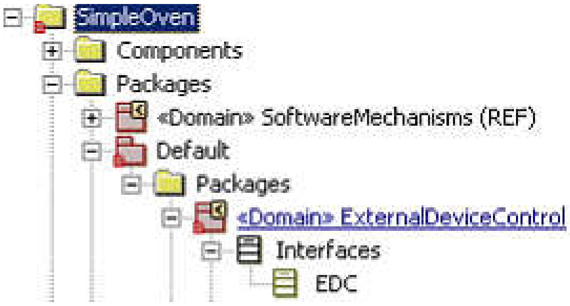
Type : sys_device_e in Default *

Name	Value	Comment
SYS_DEVICE_LIGHT		
<New>		

9. Select **OK**.

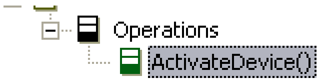
PROCEDURE: Add a Service to the Domain Interface

1. In the project explorer pane, right click on the new **EDC** interface in the *ExternalDeviceControl* domain.

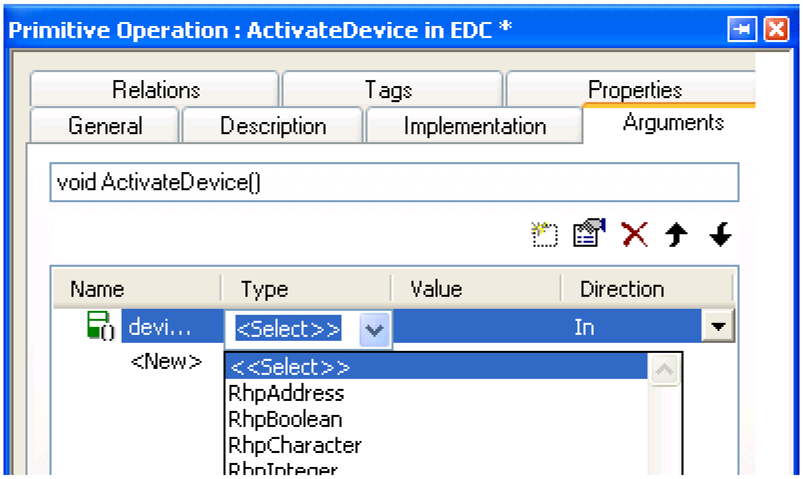


SimpleOven
Components
Packages
«Domain» SoftwareMechanisms (REF)
Default
Packages
«Domain» ExternalDeviceControl
Interfaces
EDC

2. Select **Add New → Operation**.
3. Name the operation *ActivateDevice*.



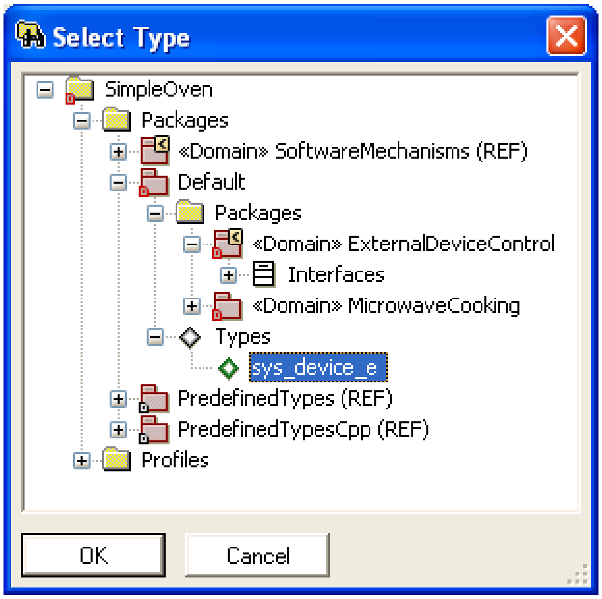
4. Double click on the new **ActivateDevice** domain service to bring up the features dialog.
5. Select the **Arguments** tab.
6. Add a new argument by clicking **<New>** in the name column.
7. Call the new argument *device_id*.
8. In the type dropdown menu for the new *device_id* argument, select **<<Select>>**.



void ActivateDevice()

Name	Type	Value	Direction
devi...	<Select>>		In

9. Select the **sys_device_e** type in the Default package.



10. Select **OK** on both dialogs to close them.
11. Repeat this procedure to add another service called *DeactivateDevice* with one parameter, called *device_id*, of type sys_device_e.
12. Save the project by selecting **File → Save**.

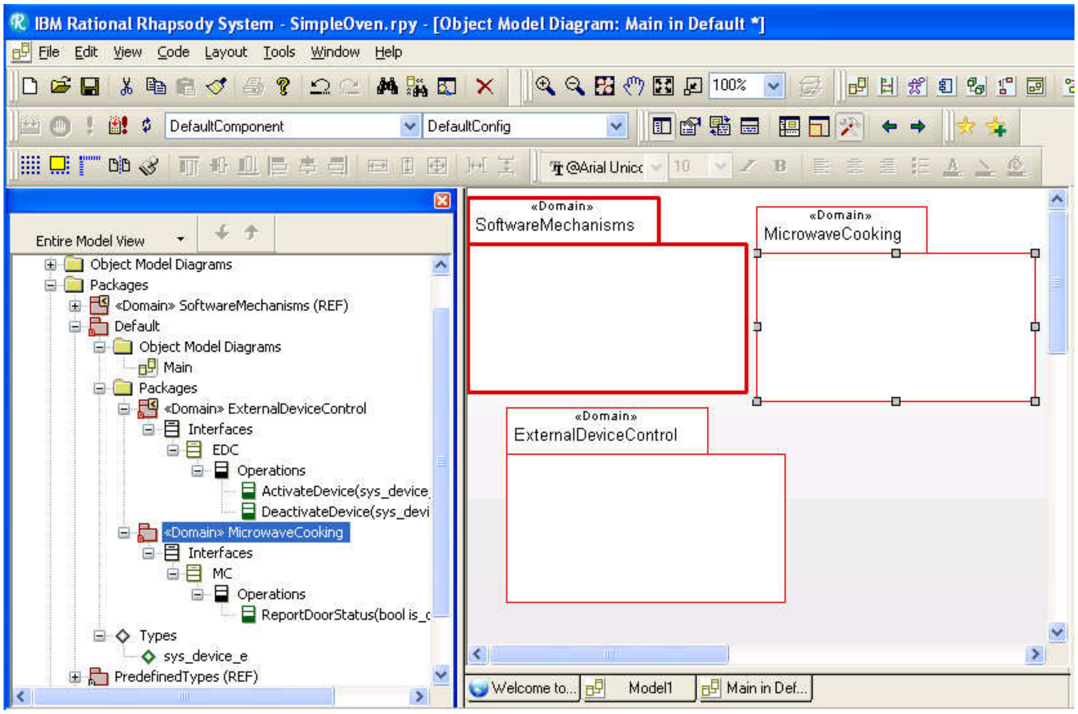
Task 4: Complete the System Domain Chart

Domains are connected to each other using “bridges.” The bridge allows one domain to utilize the capabilities of another. A domain is considered a “server domain” if it is providing services to another “client domain.” The server domain does not have any information about the implementation of any of its clients. A client domain can only access published services of a server domain. This keeps the functionalities of each domain completely encapsulated, making the system more flexible and easier to change.

We will now create the domain hierarchy for the SimpleOven model. Keep in mind that System and Domain models are PathMATE refinements of UML Packages as you work through this task.

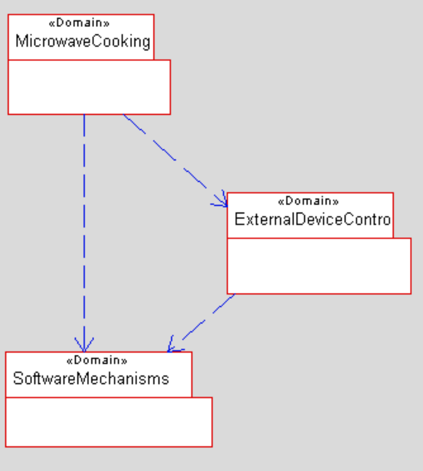
PROCEDURE: Use a Rhapsody Object Model Diagram to Specify Dependencies between Domains

1. Right click on the **Default** package.
2. Select **Add new → Diagrams → Object Model Diagram**.
3. Type *Main* as the name into the dialog box.
4. Select **OK**.
5. Double click on the new **Main** diagram to open it if it is not already opened.
6. Drag each of the three domains, *SoftwareMechanisms, ExternalDeviceControl,* and *MicrowaveCooking* from the project explorer pane into the drawing pane.

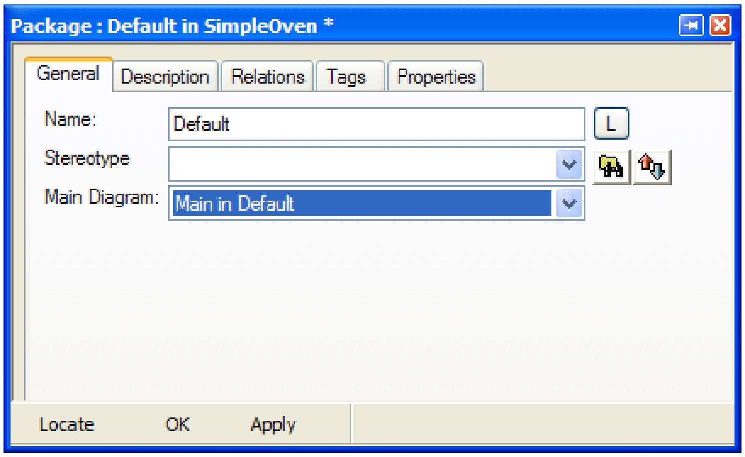


7. Select the **Dependency** Tool (↘).
8. Use the dependency tool to draw a dependency from *MicrowaveCooking* to *ExternalDeviceControl*.
9. Use the dependency tool to draw a dependency from *MicrowaveCooking* to *SoftwareMechanisms.*
10. Use the dependency tool to draw a dependency from *ExternalDeviceControl* to *SoftwareMechanisms*.

The completed domain chart should look something like this:



11. Double click on the **Default** package to bring up the Features Dialog.
12. In the "Main Diagram" pull down menu, select *Main*.


Package : Default in SimpleOven *

Name: Default

Stereotype

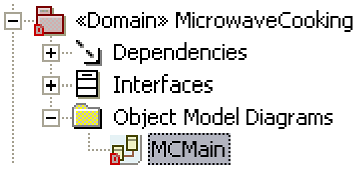
Main Diagram: Main in Default

Task 5: Complete the Class Diagram for the MicrowaveCooking Domain

The MicrowaveCooking Domain is the heart of the application. In this task, you will complete the MicrowaveCooking's Domain Class Diagram by adding Classes, Attributes, and Associations.

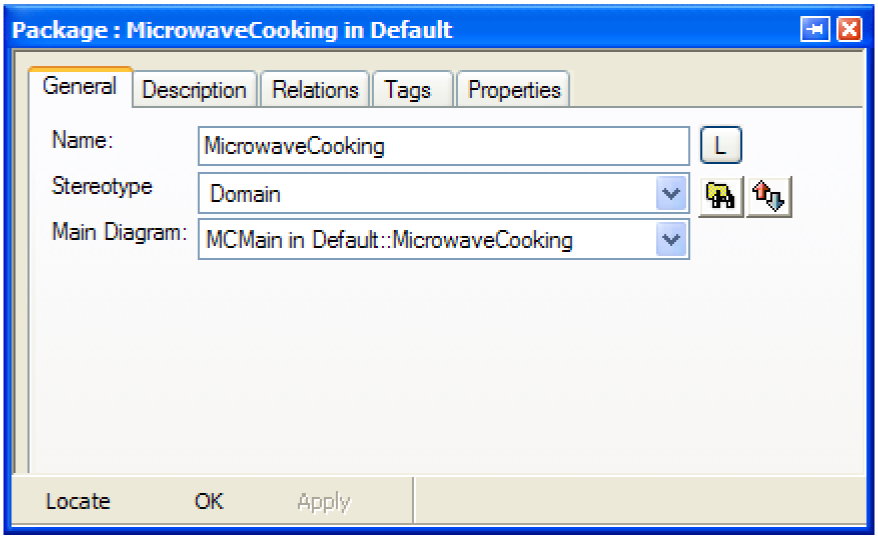
PROCEDURE: Create the Main, Object Model Diagram for the Microwave Cooking Domain

1. Right click on the **<<Domain>>MicrowaveCooking** domain in the project explorer window.
2. Select **Add new → Diagrams → Object Model Diagram**.
3. Title it *MCMain*
4. Select **OK**.



«Domain» MicrowaveCooking
Dependencies
Interfaces
Object Model Diagrams
MCMain

5. Double Click on <<Domain>> MicrowaveCooking to open the features dialog.
6. If MCMain is not already selected as the "Main Diagram" for Microwave Cooking, select it from the pull-down menu.



Package : MicrowaveCooking in Default

Name: MicrowaveCooking

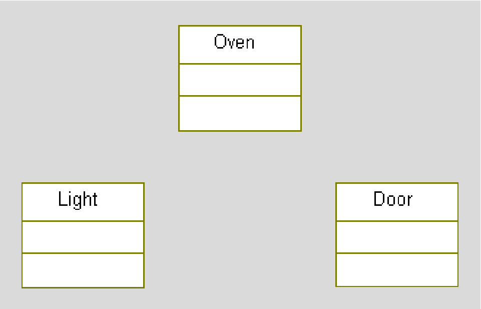
Stereotype Domain

Main Diagram: MCMain in Default::MicrowaveCooking

PROCEDURE: Use a Rhapsody Object Model Diagram to Create Classes in the Microwave Cooking Domain

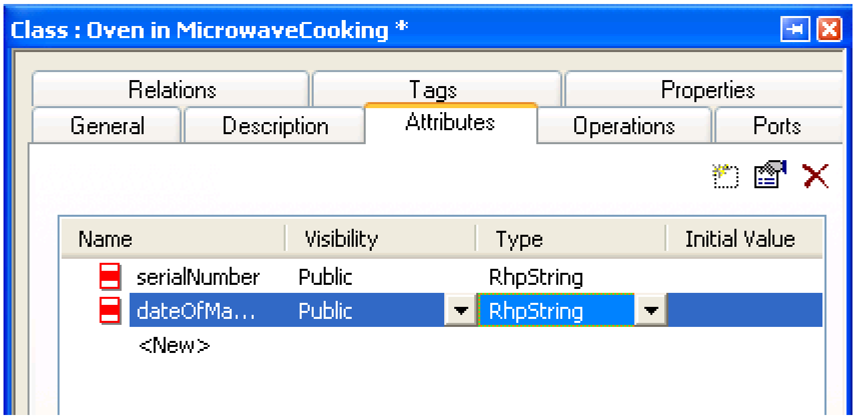
1. Double click the **MCMain** object model diagram to open it if it is not already opened.

2. Select the **Class** tool (▤).
3. Click in the drawing area to create a new class.
4. Title it *Oven*.
5. Create two more classes in the same manner called *Light* and *Door*.



PROCEDURE: Add Attributes to the Oven and Light Classes

1. Right click on the **Oven** class in the drawing area.
2. Select **Features…** to bring up the features dialog.
3. Select the **Attributes** tab.
4. Click **<New>** in the name column to create a new attribute.
5. Title it *serialNumber*.
6. In the type dropdown menu for the new attribute select **RhpString**.
7. Click **<New>** in the name column to create a new attribute.
8. Title it *dateOfManufacture*.
9. In the type dropdown menu for the new attribute select **RhpString**.

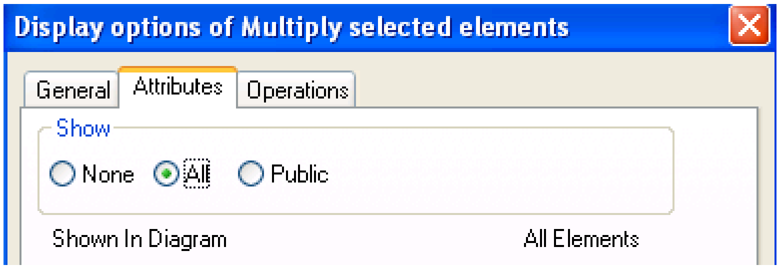


Name	Visibility	Type	Initial Value
serialNumber	Public	RhpString	
dateOfMa...	Public	RhpString	
<New>			

10. Select **OK** to close the features dialog.

By default, Rhapsody hides the attributes on the diagram to keep clutter down in code-oriented models. Here is how to make them show up:

11. Hit control-A to select all 3 classes. Right click on the **Oven** class in the drawing pane.
12. Select **Display Options…**.
13. Select the **Attributes** tab.
14. In the Show section, set the radio button to **All**.



Display options of Multiply selected elements

General | Attributes | Operations

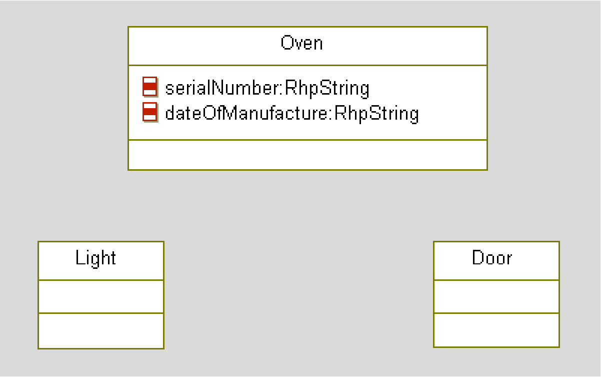
Show

○ None ● All ○ Public

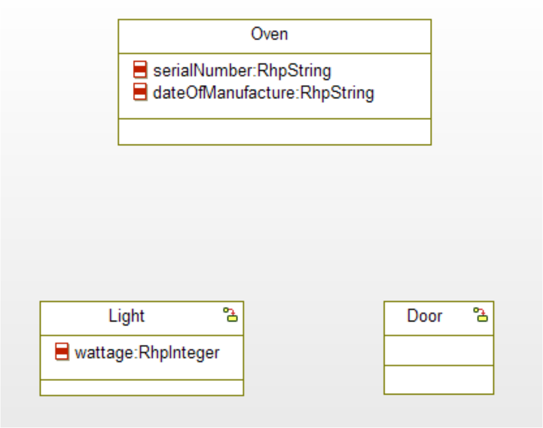
Shown In Diagram — All Elements

15. Select **OK** to close the display options dialog.

Now we can see the attributes that were added.



16. Follow the same procedure to add an attribute called *wattage* of type *RhpInteger* to the *Light* class.

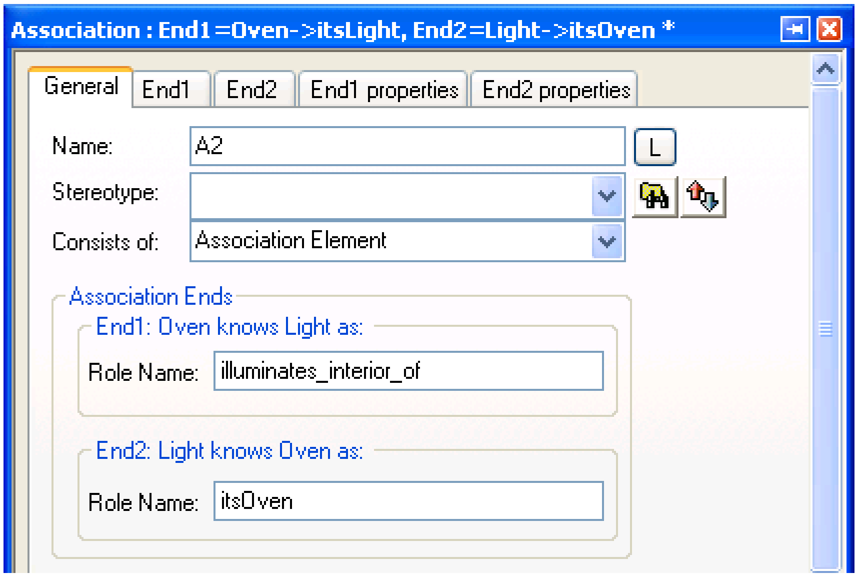


PROCEDURE: Add Associations between Classes on the MicrowaveCooking Class Diagram

Classes within a domain work together. Associations are used to capture how classes work together.

1. Select the **Association** tool (⮡).
2. Use the **Association** tool to draw an association between the *Oven* and *Light* classes.
3. Double click on the association to bring up the features dialog.
4. On the **General** tab in the Consists of: dropdown menu, select **Association Element**.
5. In the Name field, enter *A2* (this is its PAL tag).
6. Under Association Ends, End1: Oven knows Light as, replace the Role Name *itsLight* with *illuminates_interior_of*.

NOTE: End1 and End2 may be reversed if you started drawing the association from the Light class. That will work just fine, so long as "Oven knows Light as" is associated with the Role Name "illuminates_interior_of".


Association : End1=Oven->itsLight, End2=Light->itsOven *

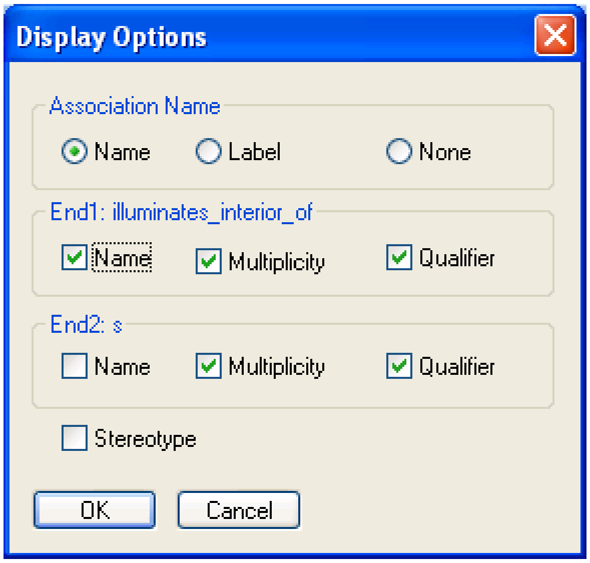
Name: A2
Stereotype:
Consists of: Association Element

Association Ends

End1: Oven knows Light as:
Role Name: illuminates_interior_of

End2: Light knows Oven as:
Role Name: itsOven

7. Select **OK.**
8. Right click on the association.
9. Select **Display Options….**
10. Check the **Name** box for the end titled *illuminates_interior_of*.


Display Options

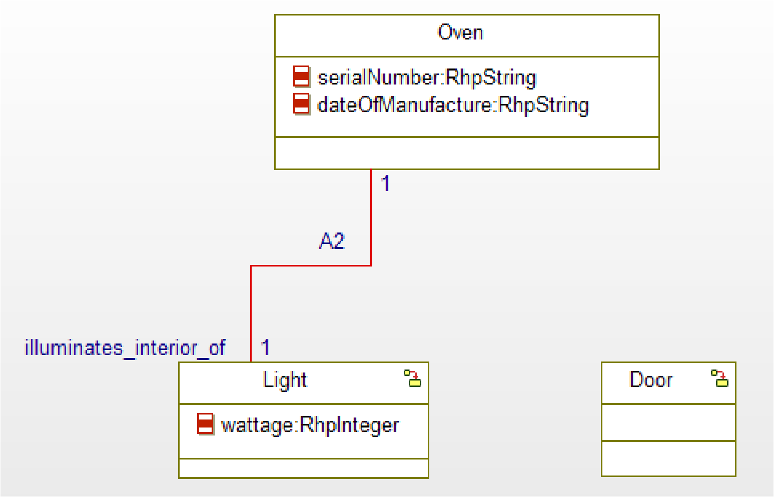
Association Name
● Name ○ Label ○ None

End1: illuminates_interior_of
☑ Name ☑ Multiplicity ☑ Qualifier

End2: s
□ Name ☑ Multiplicity ☑ Qualifier

□ Stereotype

11. Select **OK.**

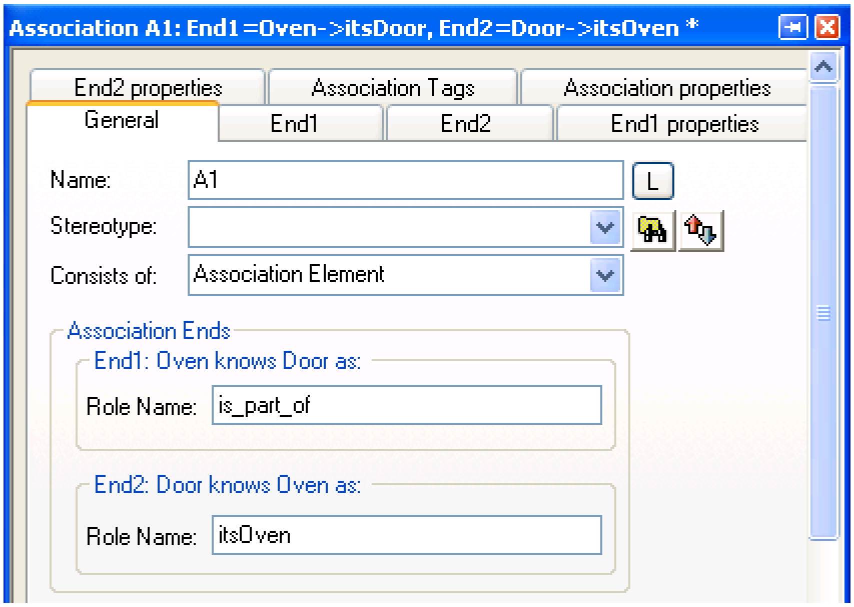


12. Use the **Association** tool to create an association between the *Oven* and *Door* classes.

13. Immediately after creating the association, a solid box appears over the line. Enter *A1* to name the association and turn it into an Association Element. *(This is a quick way of naming the association.)*

14. Double click the new association to bring up the features window.

15. On the **General** tab in **Association Ends**, enter *is_part_of* in place of *itsDoor* for the Role Name of End1: Oven knows Door as:.


Association A1: End1=Oven->itsDoor, End2=Door->itsOven *

Name: A1

Stereotype:

Consists of: Association Element

Association Ends

End1: Oven knows Door as:

Role Name: is_part_of

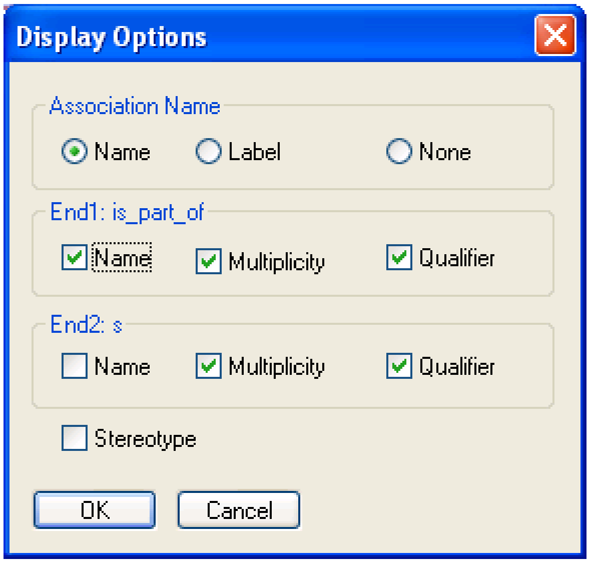
End2: Door knows Oven as:

Role Name: itsOven

16. Select **OK.**

17. Right click on the association **A1** and select **Display Options**…

18. Check the **Name** box for the end titled *is_part_of.*



Display Options

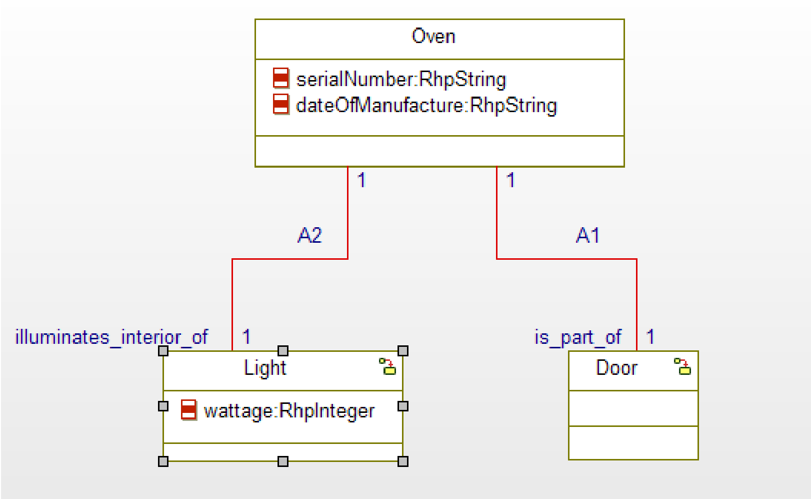
Association Name
● Name ○ Label ○ None

End1: is_part_of
☑ Name ☑ Multiplicity ☑ Qualifier

End2: s
□ Name ☑ Multiplicity ☑ Qualifier

□ Stereotype

19. Select **OK**.



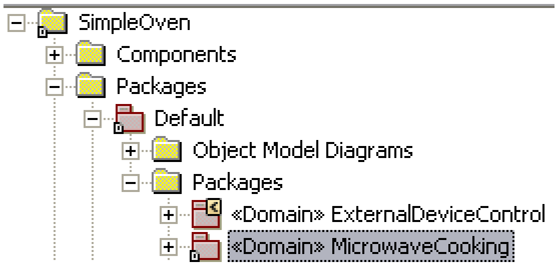
At this point we will leave the class diagram to specify behavior.

Task 6: Complete the Application Domain

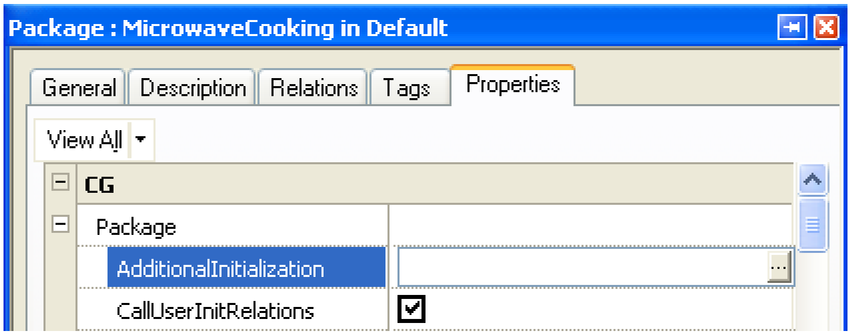
To complete the application domain, we must incorporate action language that initializes the MicrowaveCooking Domain.

PROCEDURE: Add MicrowaveCooking Initialization Action

1. Double click on the **MicrowaveCooking** domain in the Project Explorer to open the features dialog.



2. Select the **Properties** tab.
3. Select the **View Common** triangle near the top left of the window, then select **View All**.
4. Expand *CG* and *Package* if needed.
5. Click in the **AdditionalInitialization** entry field to activate text entry.



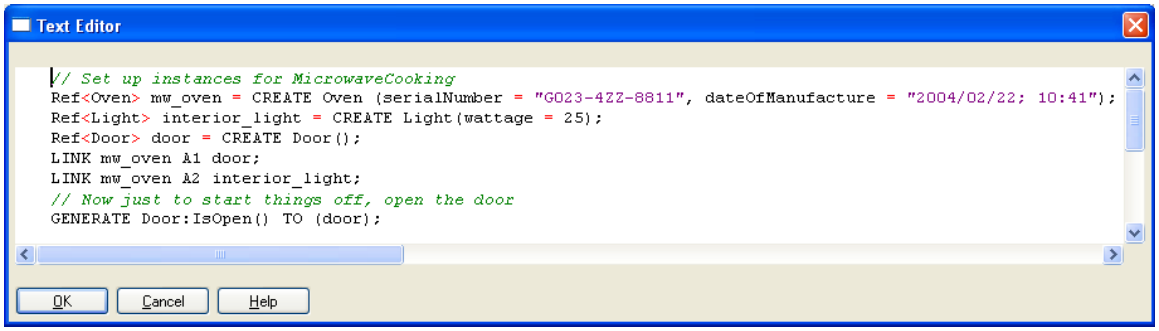
Package : MicrowaveCooking in Default

AdditionalInitialization

CallUserInitRelations ☑

6. Press the **"…"** button in the *AdditionalInitialization* field to bring up a bigger "Text Editor" window to work with.
7. Copy this PAL code into the text editor.

```
// Set up instances for MicrowaveCooking
Ref<Oven> mw_oven = CREATE Oven (serialNumber = "G023-4ZZ-8811", dateOfManufacture = "2004/02/22; 10:41");
Ref<Light> interior_light = CREATE Light(wattage = 25);
Ref<Door> door = CREATE Door();
LINK mw_oven A1 door;
LINK mw_oven A2 interior_light;
// Now just to start things off, open the door
GENERATE Door:IsOpen() TO (door);
```



```
// Set up instances for MicrowaveCooking
Ref<Oven> mw_oven = CREATE Oven (serialNumber = "G023-4ZZ-8811", dateOfManufacture = "2004/02/22; 10:41");
Ref<Light> interior_light = CREATE Light(wattage = 25);
Ref<Door> door = CREATE Door();
LINK mw_oven A1 door;
LINK mw_oven A2 interior_light;
// Now just to start things off, open the door
GENERATE Door:IsOpen() TO (door);
```

8. Select **OK** on the text editor to save the changes.
9. Select **OK** on the features dialog to close it.

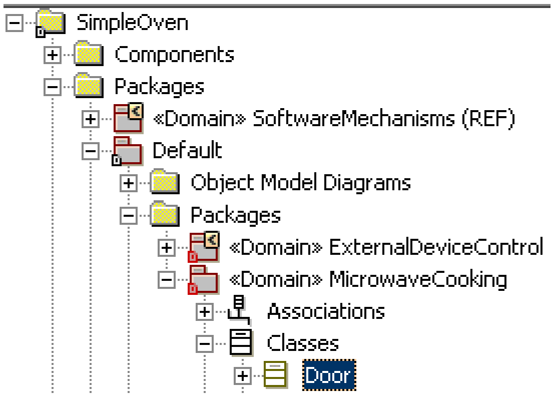
Task 7: Create the Door State Machine

State Machines are used to describe the way a class will react depending on what state that class is in. State Machines respond to signals produced by the state machine's behavior, or by other actions within the domain. Create the door State Machine in four procedures:

- Create the Door State Machine diagram
- Place the Door State Symbols
- Draw Transition Lines for the Door States
- Create Signals and Triggers for the Door State Machine

PROCEDURE: Create the Door State Machine Diagram

1. Right-click on the **Door** class in the Project Explorer.

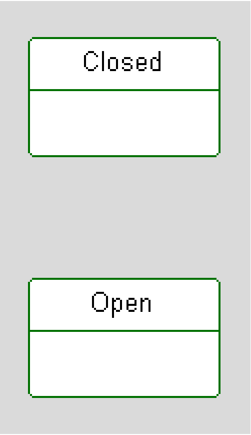


- SimpleOven
  - Components
  - Packages
    - «Domain» SoftwareMechanisms (REF)
    - Default
      - Object Model Diagrams
      - Packages
        - «Domain» ExternalDeviceControl
        - «Domain» MicrowaveCooking
          - Associations
          - Classes
            - Door

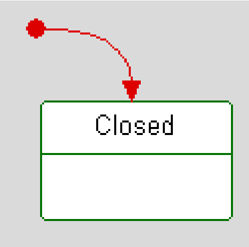
2. Select **Add New → Diagrams → Statechart**.

PROCEDURE: Place the Door State Symbols

1. Use the **State** tool () to create two states: *Closed* and *Open*.

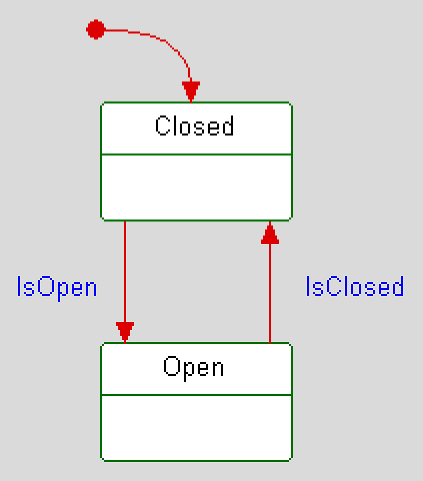


2. Use the **Default Transition** tool () to make an initial transition into the *Closed* state.



PROCEDURE: Draw Transition Lines for the Door States

1. Using the **Transition** tool (↘) draw these transitions:
   a. From Closed to Open, called *IsOpen.*
   b. From Open to Closed, called *IsClosed.*



2. You can move the transition titles by simply clicking and dragging them to make the diagram more readable.

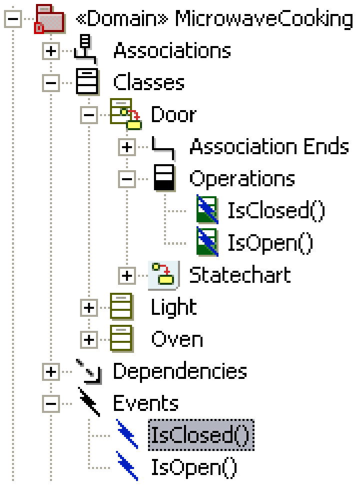
NOTE: Explore the *Layout* menu at the top of the Rhapsody window for tools to improve your diagram’s appearance.

Task 8: View Signals for the Door State Machine

UML 2.0 defines Signals and Signal Triggers. A Signal describes an event and parameters. A Signal is an element of a Package (Domain) and in Rhapsody cannot be scoped to a class. A Signal Trigger is a property of a State Transition and indicates which transitions occur when the class receives its corresponding Signal.

PROCEDURE: Use Project Explorer to See Signals in the MicrowaveCooking Domain

1. Under the *MicrowaveCooking* domain in the Project Explorer, expand **Events**.
2. Expand the class **Door** and then expand **Operations**.



```
«Domain» MicrowaveCooking
    Associations
    Classes
        Door
            Association Ends
            Operations
                IsClosed()
                IsOpen()
            Statechart
        Light
        Oven
    Dependencies
    Events
        IsClosed()
        IsOpen()
```

Under Events, you will see the Signals. Under Door’s Operations, you will see the Signal Triggers. Rhapsody automatically created these when you created the transitions in the Door State Machine.

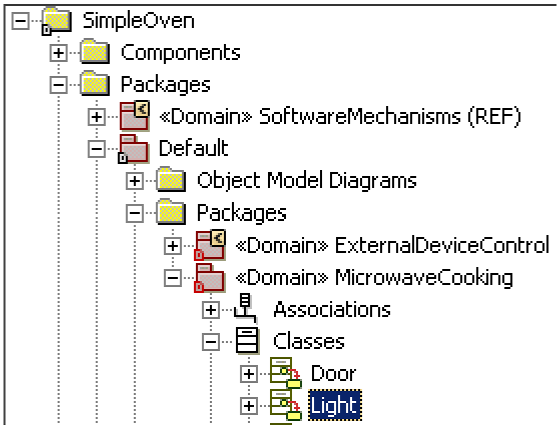
Task 9: Create the Light State Machine

The Light State Machine shows transitions between on and off. Create the Light State Machine in four parts:

- Create the Light State Machine diagram
- Place the Light State Symbols
- Draw Transition Lines for the Light States
- Create Signals and Triggers for the Light State Machine

PROCEDURE Create the Light State Machine Diagram

1. Right click on the **Light** class in the Project Explorer.



SimpleOven
Components
Packages
«Domain» SoftwareMechanisms (REF)
Default
Object Model Diagrams
Packages
«Domain» ExternalDeviceControl
«Domain» MicrowaveCooking
Associations
Classes
Door
Light

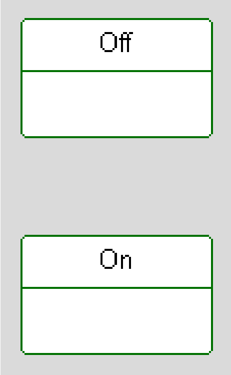
2. Select **Add New → Diagrams → Statechart**.

PROCEDURE: Place the Light State Symbols

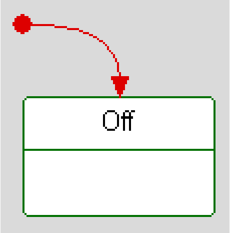
3. Use the **State** tool (□) to create two states called:

   a. *Off*

   b. *On*

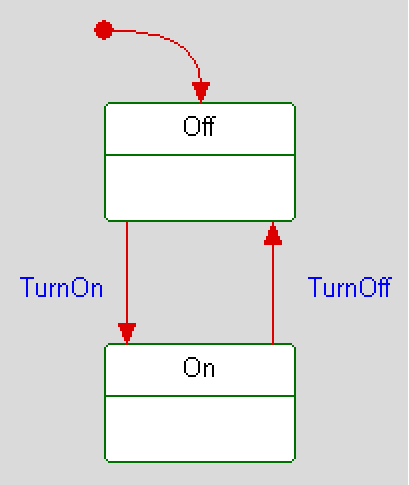


4. Use the **Default Transition** tool (↘) to make an initial transition into the Off state.



PROCEDURE: Draw Transition Lines for the Light States

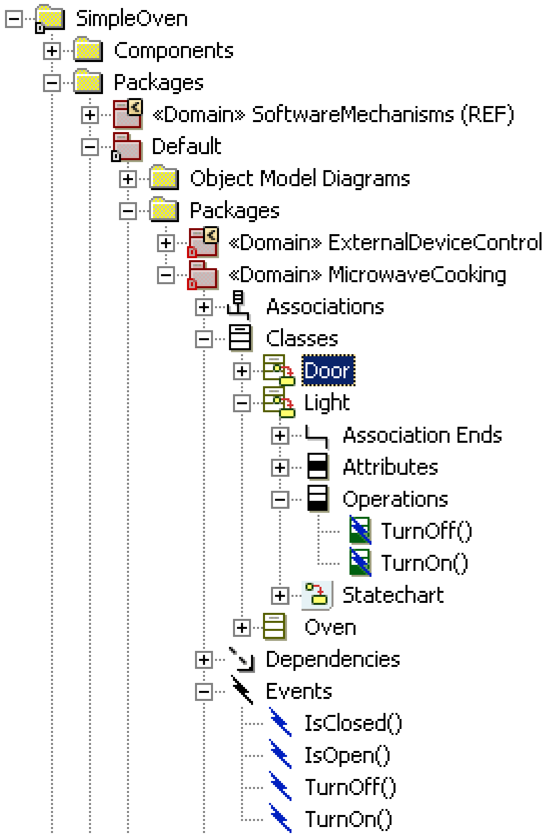
1. Using the **Transition** tool (↘) draw these transitions:
   a. From Off to On, called *TurnOn*
   b. From On to Off, called *TurnOff*



2. You can move the transition titles by simply clicking and dragging them to make the diagram more readable.

Task 10: View Signals for the Light State Machine

Rhapsody has automatically created the Signals and Signal triggers for the Light State Machine, just like it did for the Door state machine.

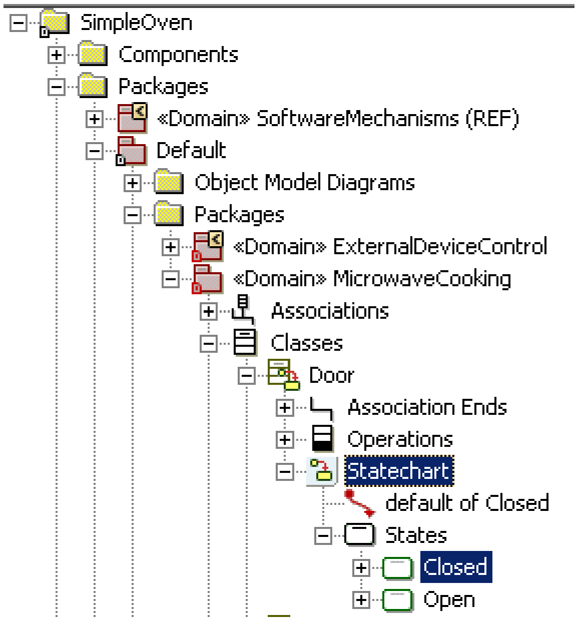


- SimpleOven
  - Components
  - Packages
    - «Domain» SoftwareMechanisms (REF)
    - Default
      - Object Model Diagrams
      - Packages
        - «Domain» ExternalDeviceControl
        - «Domain» MicrowaveCooking
          - Associations
          - Classes
            - Door
            - Light
              - Association Ends
              - Attributes
              - Operations
                - TurnOff()
                - TurnOn()
              - Statechart
            - Oven
          - Dependencies
          - Events
            - IsClosed()
            - IsOpen()
            - TurnOff()
            - TurnOn()

Task 11: Add State Actions

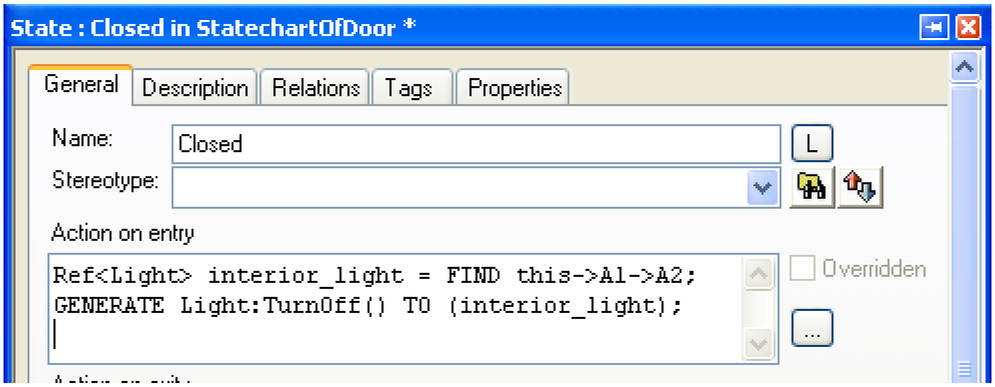
When a class transitions to a state, various actions may be executed: State Exit, Transition and/or State Entry. We will use State Entry Actions to communicate between the Door and the Light classes. When the oven door is closed, the interior light will be turned off. When the oven door is open, the interior light will be turned on.

PROCEDURE: Define Entry Action for Door's Closed State



1. Double click on the Door State Machine's **Closed** State in either the Statechart Diagram or the package explorer to open the features dialog.
2. Select the **General** tab if it is not already selected.
3. Copy the following PAL into the Action on entry field:

```
Ref<Light> interior_light = FIND this->A1->A2;
GENERATE Light:TurnOff() TO (interior_light);
```



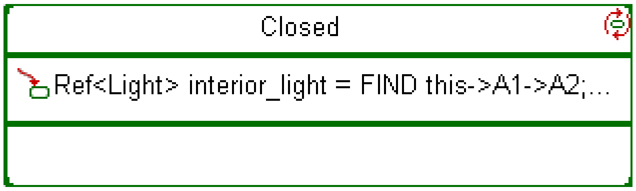
Name: Closed

Stereotype:

Action on entry

```
Ref<Light> interior_light = FIND this->A1->A2;
GENERATE Light:TurnOff() TO (interior_light);
```

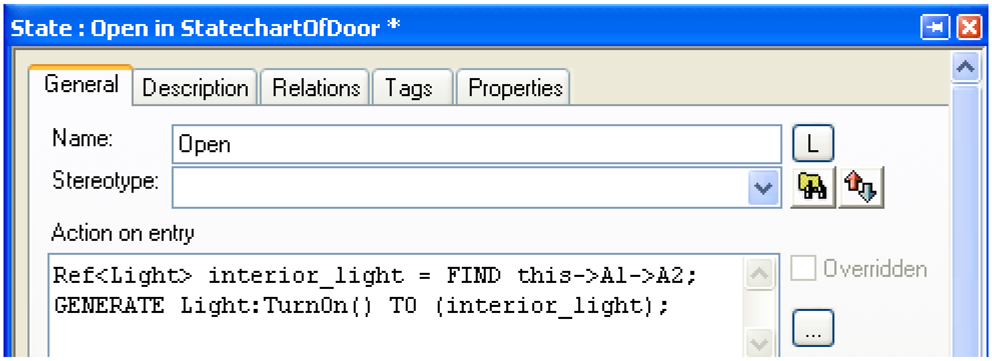
4. Press **Apply** to save the changes.
5. Click **OK** to exit dialog box.
6. On the **Closed** state symbol in the Door Diagram, there now appears an action visibility control . Click it to make the entry action PAL visible.



Ref<Light> interior_light = FIND this->A1->A2;...

7. Follow the same procedure to add this PAL to the entry action for the Door's Open State:

```
Ref<Light> interior_light = FIND this->A1->A2;
GENERATE Light:TurnOn() TO (interior_light);
```



State : Open in StatechartOfDoor *

Name: Open

Stereotype:

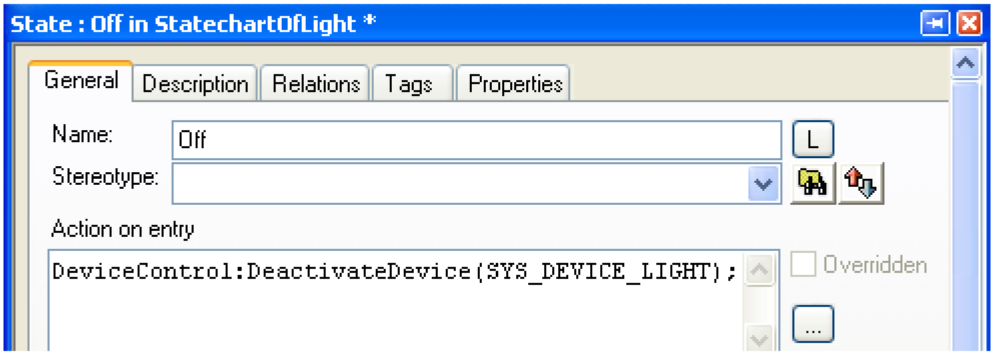
Action on entry

```
Ref<Light> interior_light = FIND this->A1->A2;
GENERATE Light:TurnOn() TO (interior_light);
```

□ Overridden

8. Follow the same procedure to add this PAL to the entry action for Light's Off state:

```
ExternalDeviceControl:DeactivateDevice(SYS_DEVICE_LIGHT);
```



State : Off in StatechartOfLight *

Name: Off

Stereotype:

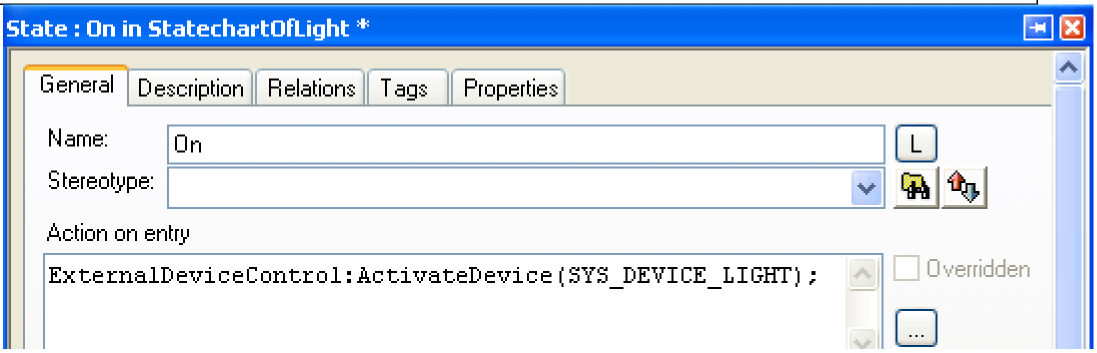
Action on entry

```
DeviceControl:DeactivateDevice(SYS_DEVICE_LIGHT);
```

□ Overridden

9. Follow the same procedure – one more time – to add this PAL to the entry action for Light's On state:

```
ExternalDeviceControl:ActivateDevice(SYS_DEVICE_LIGHT);
```



State : On in StatechartOfLight *

Name: On

Stereotype:

Action on entry

```
ExternalDeviceControl:ActivateDevice(SYS_DEVICE_LIGHT);
```

☐ Overridden

Congratulations! Your Simple Oven project has a Real Model.

Save the project by selecting **File → Save**.

Prepare for Transformation

You now have a complete, executable platform independent model and your project is just a few steps from being ready for transform into an executable system. You'll reuse some items from the SimpleOven Example project which comes with PathMATE. This section takes you through the following steps:

- Utilize a complete Example SimpleOven project and use it as a reference, copy C++ transformation properties and realized code from it.
- Add a PathMATE Project to the SimpleOven project which will control transformation and deployment.
- "Smoke Test" your model by generating documentation from it.

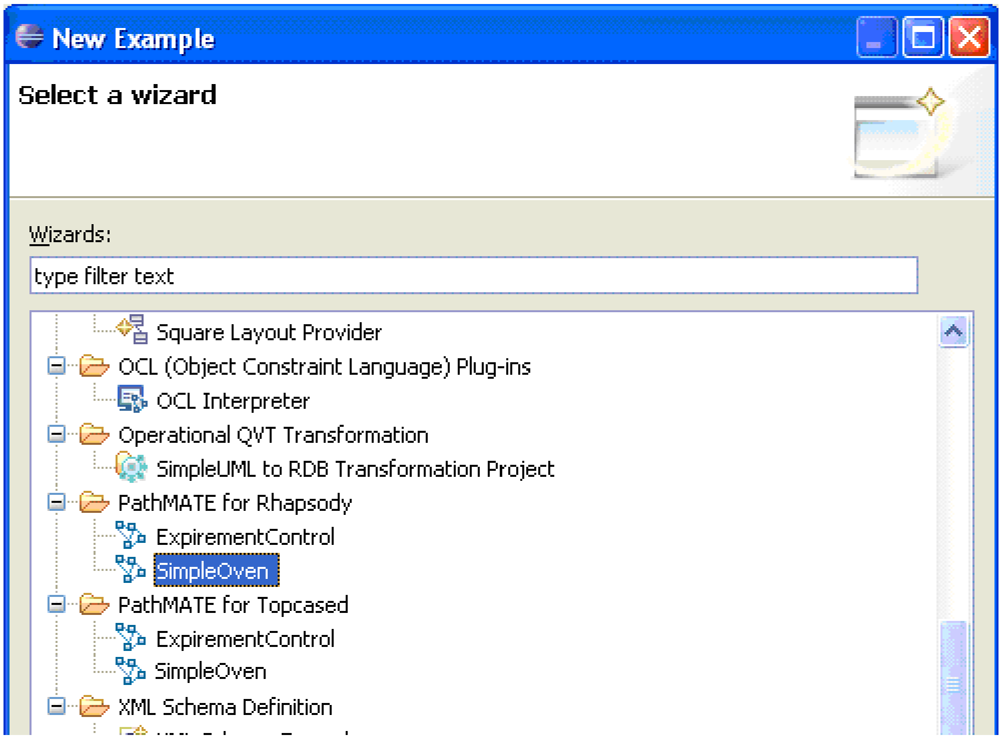
Task 1: Reuse elements from Example Project

It is now time to start working in the Eclipse environment. Open the version of Eclipse supplied with PathMATE. Designate a workspace to store all the files for the sample system. Typically you designate your workspace once, just after you install Eclipse. We refer to this directory as your Workspace Directory. Use the same workspace whenever you work on this tutorial.

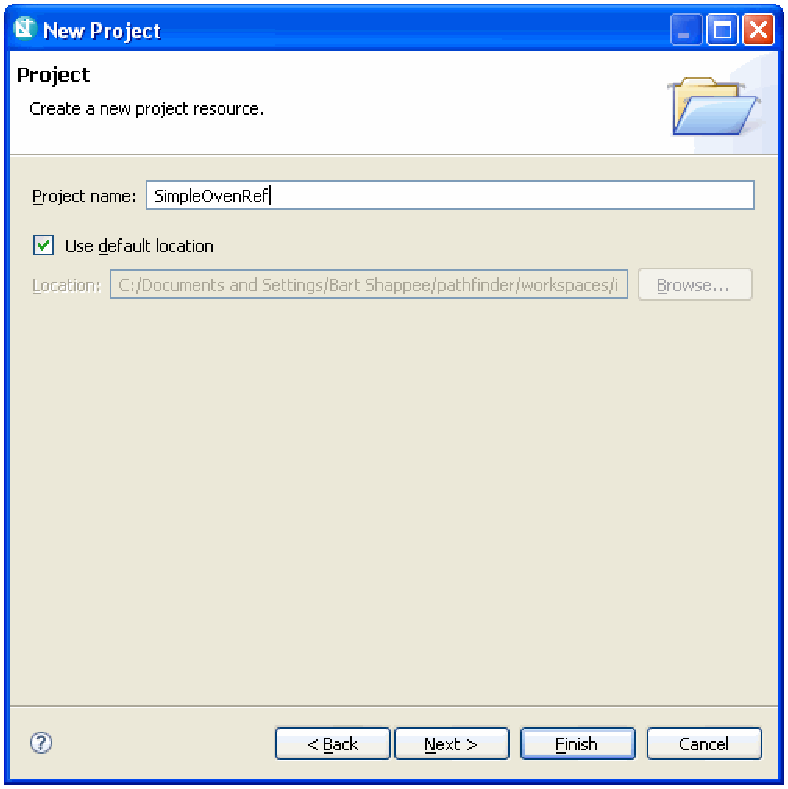
SimpleOven exists as a sample model in the PathMATE examples library. We will use SimpleOven as a reference project to copy the realized ExternalDeviceControl implementation and properties that guide the automatic generation of a C++ and Java artifacts for this system.

PROCEDURE: Use Eclipse to Instantiate an Example Project for SimpleOven

1. Select **File → New → Example**
2. Select **PathMATE for Rhapsody → Simple Oven**



3. Select **Next.**
4. Name the Project *SimpleOvenRef*.



Create a new project resource.

Project name: SimpleOvenRef

☑ Use default location

Location: C:/Documents and Settings/Bart Shappee/pathfinder/workspaces/i

5. Press **Finish**.

PROCEDURE: Use Eclipse Resource Perspective to Copy Files Needed for C++ and Java Systems

1. Ensure you are in the Eclipse Resource perspective. Typically this is done from the upper right of your Eclipse desktop, but on your first time in you may need to use the Eclipse menu *Window→Open Perspective→ Other…* and select **Resource**.
2. In the *SimpleOvenRef* project, select the **cpp** subdirectory then right click and select **Copy**.
3. Select the **SimpleOven** project, right click and select **Paste.**
4. Repeat this process for the **java** subdirectory.

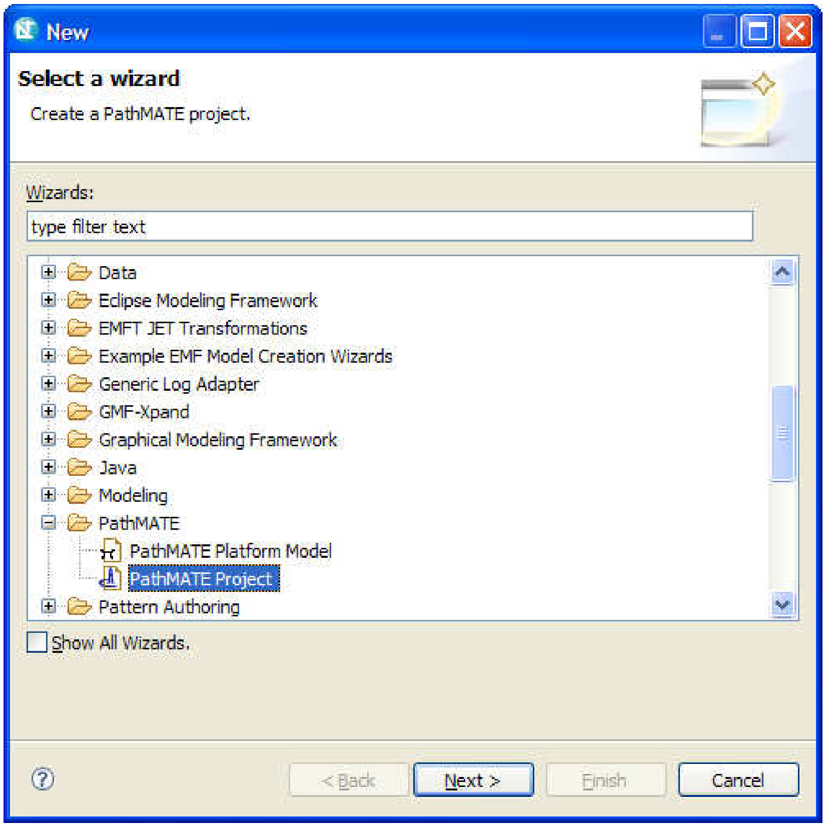
Task 2: Create A PathMATE Project

PROCEDURE: Refresh Eclipse Project

1. Right click on the SimpleOven Project in the Project Explorer
2. Select Refresh

PROCEDURE: Use Eclipse to Create a New PathMATE Project

1. Select **File → New → Other**
2. Select **PathMATE → PathMATE Project**



New

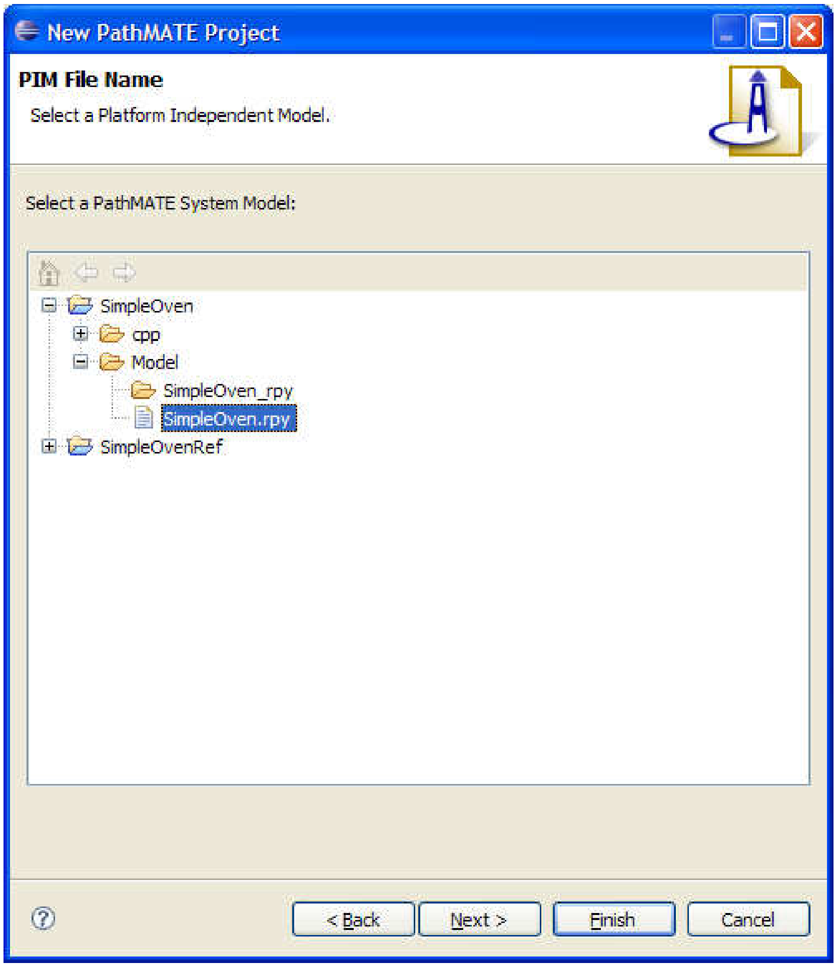
Select a wizard

Create a PathMATE project.

Wizards:

type filter text

3. Press **Next**
4. Select **SimpleOven>Model>SimpleOven>SimpleOven.rpy**



New PathMATE Project

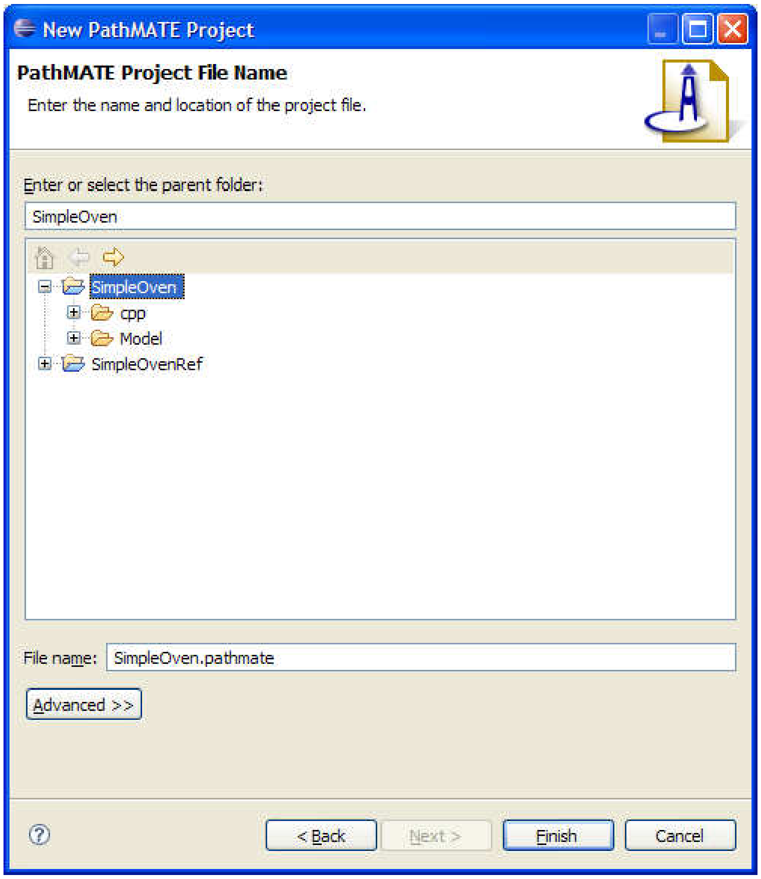
PIM File Name

Select a Platform Independent Model.

Select a PathMATE System Model:

- SimpleOven
  - cpp
  - Model
    - SimpleOven_rpy
    - SimpleOven.rpy
- SimpleOvenRef

5. Press **Next**
6. Select **SimpleOven** to create the PathMATE project at the project level.



New PathMATE Project

PathMATE Project File Name

Enter the name and location of the project file.

Enter or select the parent folder:

SimpleOven

File name: SimpleOven.pathmate

7. Press **Finish**

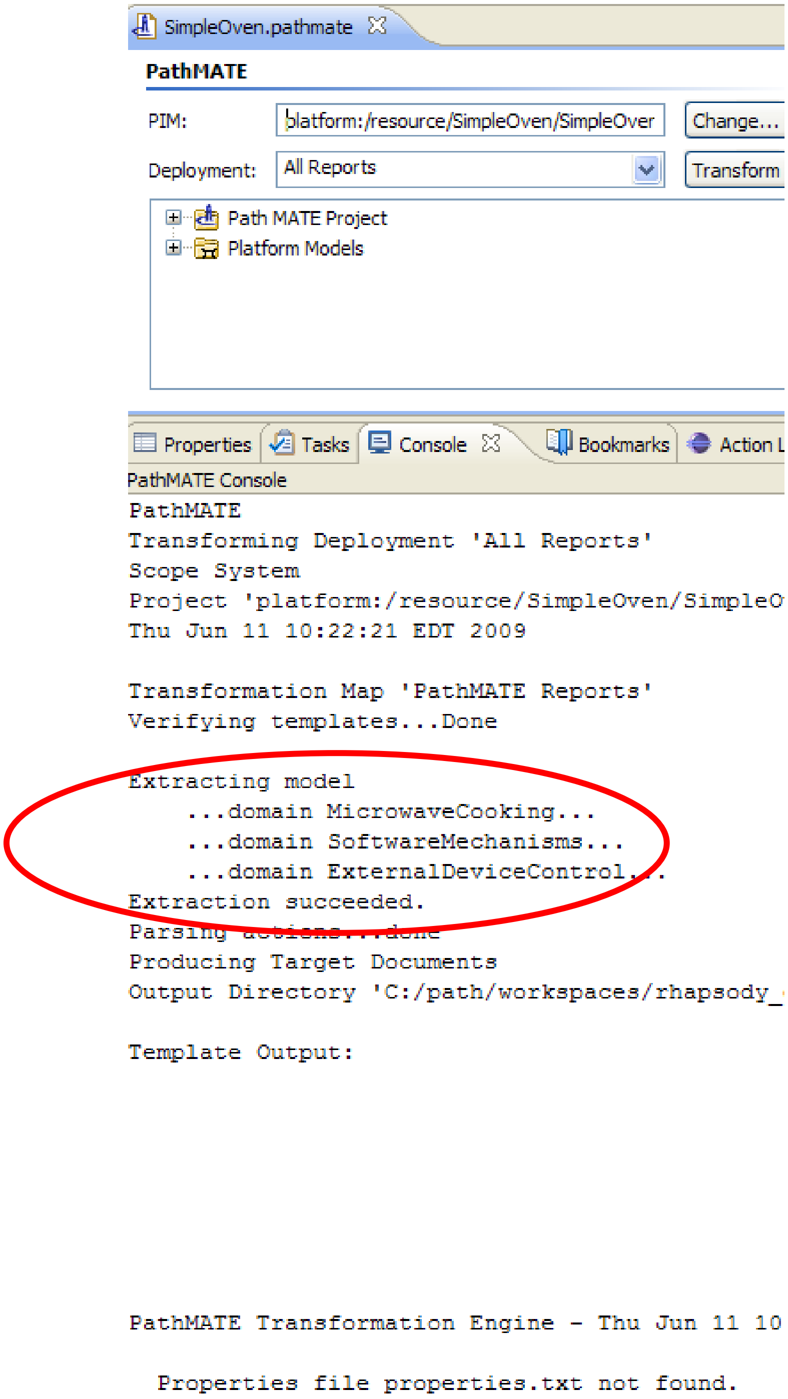
Task 3: Check Your Model

Before going on, this is a good point to check for any errors that may have occurred in entering your model.

We will use the documentation transformation All Reports, and transform to let the PathMATE Transformation Engine automatically validate our model. The transformation maps applied in this procedure can optionally be controlled by markings specified in a properties.txt file, but we will not use one in this case.

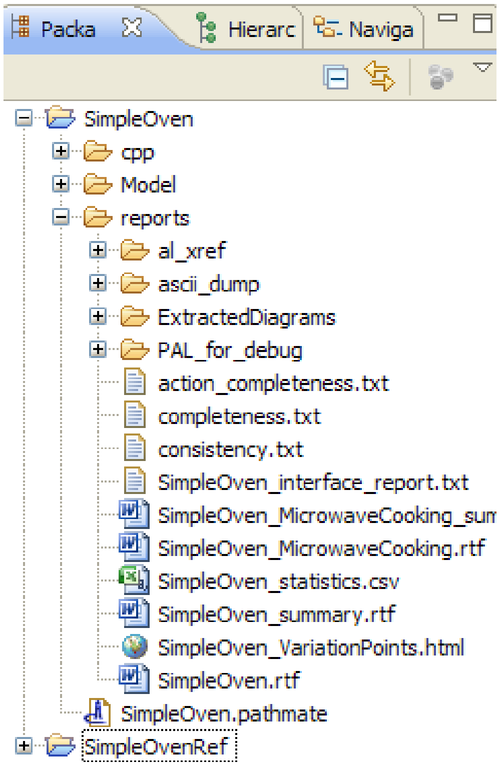
PROCEDURE: Use the PathMATE Project Editor to test Your Project

1. Double click on SimpleOven > SimpleOven.pathmate in the project explorer.
2. Select the deployment, **All Reports,** from the Deployment pick list.
3. Click **Transform**. A progress dialog will appear. When it finishes, your Console window will show your results.
4. Verify that no model extraction errors or other errors are listed.



```
PathMATE
Transforming Deployment 'All Reports'
Scope System
Project 'platform:/resource/SimpleOven/SimpleO
Thu Jun 11 10:22:21 EDT 2009

Transformation Map 'PathMATE Reports'
Verifying templates...Done

Extracting model
    ...domain MicrowaveCooking...
    ...domain SoftwareMechanisms...
    ...domain ExternalDeviceControl...
Extraction succeeded.
Parsing actions...done
Producing Target Documents
Output Directory 'C:/path/workspaces/rhapsody_

Template Output:





PathMATE Transformation Engine - Thu Jun 11 10

  Properties file properties.txt not found.
```

5. Check in the Project Explorer, under the SimpleOven>Reports, that you have the generated documentation files. If not, you may have a problem with your model.



- SimpleOven
  - cpp
  - Model
  - reports
    - al_xref
    - ascii_dump
    - ExtractedDiagrams
    - PAL_for_debug
    - action_completeness.txt
    - completeness.txt
    - consistency.txt
    - SimpleOven_interface_report.txt
    - SimpleOven_MicrowaveCooking_sum
    - SimpleOven_MicrowaveCooking.rtf
    - SimpleOven_statistics.csv
    - SimpleOven_summary.rtf
    - SimpleOven_VariationPoints.html
    - SimpleOven.rtf
  - SimpleOven.pathmate
- SimpleOvenRef

6. Some problems that might turn up are:
   - Extraneous elements that were created in the process of experimentation are not fully specified.
   - Spaces in Names; e.g. "External Device Control" should be "ExternalDeviceControl".
   - Type not specified for a parameter.
   - Issues with directory structure.
7. You may wish to use the Problems View to quickly navigate to the source of the error (**Window → Show View → Problems**).
8. Correct any problems identified in the messages, then return to SimpleOven.pathmate in the editor pane and click **Transform** again.

Congratulations! You are Ready to Generate Real Code, Real Fast.

Generate C++ Code and Visual Studio Project Files

You are now ready to generate C++ implementation code and Visual Studio project files.

- Configure Development Environment Options
- Transform Your Model to Code and Project Files

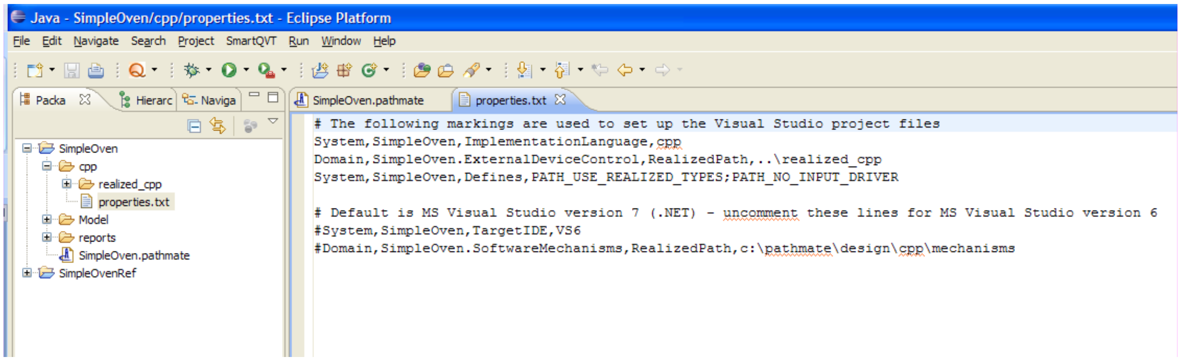
NOTE – if you prefer to generate Java implementation code, please skip ahead to the Generate Java Code section.

Task 1: Configure Development Environment Options

The *markings* file properties.txt controls many aspects of transformation, inclusion of "Execution Instrumentation" in runtime code to support the Spotlight debugger and IDE environment when generating IDE project files. The markings file is located in the project folder for the implementation language you are generating. So for C++ generation, it is in the SimpleOven/cpp folder.

PROCEDURE: Use Eclipse Text Editor to Enable Spotlight Instrumentation by Changing "Markings" in properties.txt

1. Double click **properties.txt** to open it.



```
# The following markings are used to set up the Visual Studio project files
System,SimpleOven,ImplementationLanguage,cpp
Domain,SimpleOven.ExternalDeviceControl,RealizedPath,..\realized_cpp
System,SimpleOven,Defines,PATH_USE_REALIZED_TYPES;PATH_NO_INPUT_DRIVER

# Default is MS Visual Studio version 7 (.NET) - uncomment these lines for MS Visual Studio version 6
#System,SimpleOven,TargetIDE,VS6
#Domain,SimpleOven.SoftwareMechanisms,RealizedPath,c:\pathmate\design\cpp\mechanisms
```

2. Add the following as the top line:

```
Domain,SimpleOven.*,SpotlightEnabled,T
```

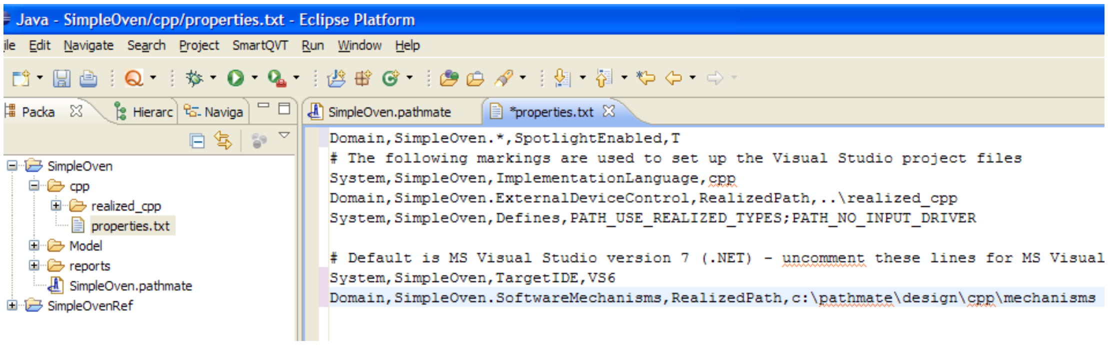
3. Select **File → Save** to save the changes.

PROCEDURE: Change Markings for Visual Studio 6 ONLY

The default target when generating IDE project files is to support Visual Studio version 7 (SimpleOven.vcproj). To generate Visual Studio version 6 project files (SimpleOven.dsp and SimpleOven.dsw) change properties.txt to specify Visual Studio 6 is the Target IDE.

If you are using Visual Studio version 7+, skip this procedure.

1. Uncomment the last 2 lines in the file.



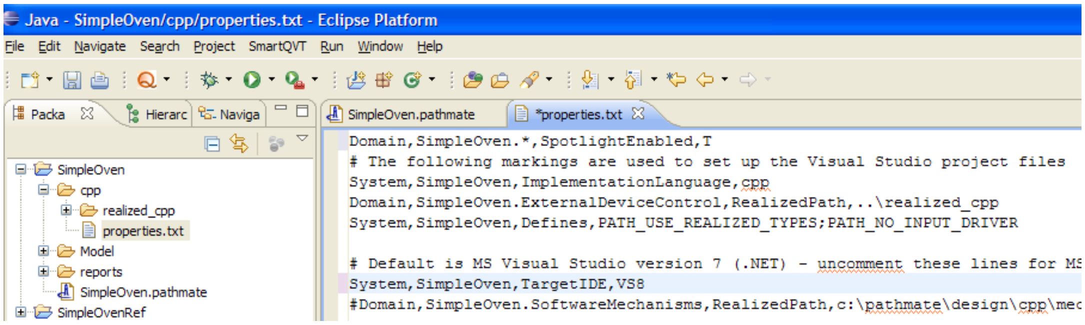
```
Domain,SimpleOven.*,SpotlightEnabled,T
# The following markings are used to set up the Visual Studio project files
System,SimpleOven,ImplementationLanguage,cpp
Domain,SimpleOven.ExternalDeviceControl,RealizedPath,..\realized_cpp
System,SimpleOven,Defines,PATH_USE_REALIZED_TYPES;PATH_NO_INPUT_DRIVER

# Default is MS Visual Studio version 7 (.NET) - uncomment these lines for MS Visual
System,SimpleOven,TargetIDE,VS6
Domain,SimpleOven.SoftwareMechanisms,RealizedPath,c:\pathmate\design\cpp\mechanisms
```

2. Select **File → Save** to save the changes.

PROCEDURE: Change Markings for Visual Studio 7+

The default target when generating IDE project files is to support the full version Visual Studio 7. To generate Visual Studio version 8 or later project files, change properties.txt to specify Visual Studio 8 or later is the Target IDE (Steps 1&2). Additionally, the Express Editions of Visual Studio 8+ require that some additional defines are used, and need to be added to properties.txt (Step 3).

1. If using Visual Studio 8 or later, uncomment the second to last line in the file.
2. If using Visual Studio 8 or later, change the marking to the version of visual studio you are using. For example VS 2005 or Version 8, is marked VS8.



```
Domain,SimpleOven.*,SpotlightEnabled,T
# The following markings are used to set up the Visual Studio project files
System,SimpleOven,ImplementationLanguage,cpp
Domain,SimpleOven.ExternalDeviceControl,RealizedPath,..\realized_cpp
System,SimpleOven,Defines,PATH_USE_REALIZED_TYPES;PATH_NO_INPUT_DRIVER

# Default is MS Visual Studio version 7 (.NET) - uncomment these lines for MS
System,SimpleOven,TargetIDE,VS8
#Domain,SimpleOven.SoftwareMechanisms,RealizedPath,c:\pathmate\design\cpp\mec
```

3. If using an express version, add the **_CRT_SECURE_NO_DEPRECATE** and

_AFX_NOFORCE_LIBS defines to the list on line 5., For example:

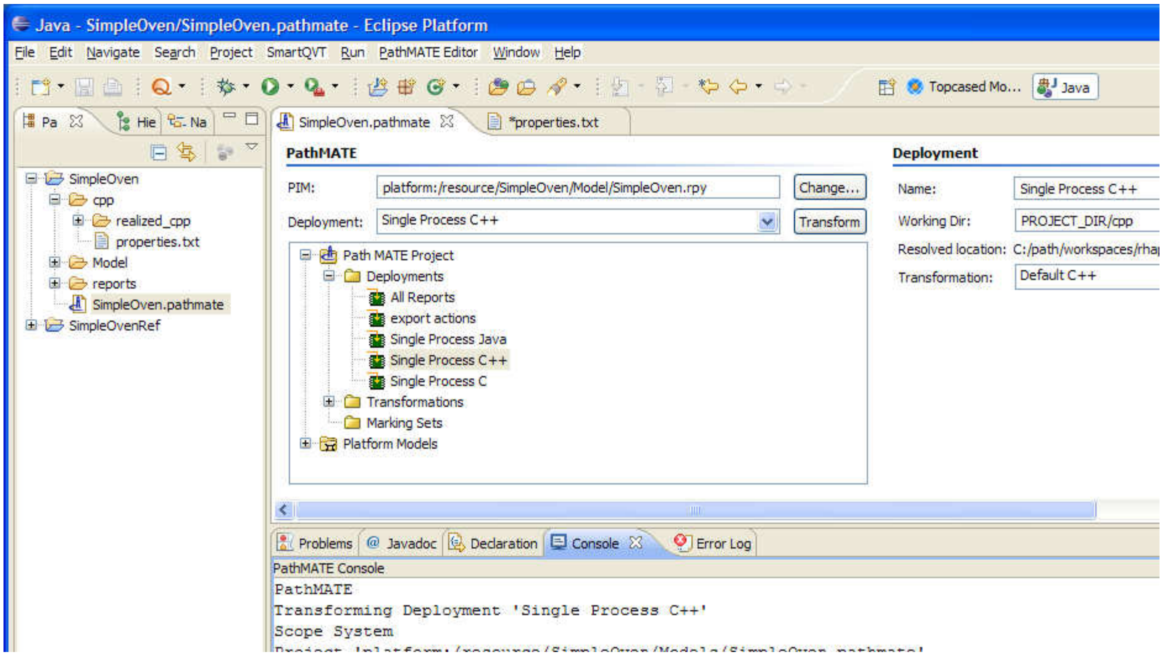
System,SimpleOven,Defines,PATH_USE_REALIZED_TYPES;PATH_NO_IPUT_DRIVER;CRT_SECURE_NO_DEPRECATE;_AFX_NOFORCE_LIBS

4. Select **File → Save** to save the changes.

Task 2: Transform Your Model to C++ Code and Visual Studio Project Files

PROCEDURE: Generate C++ Implementation Code and a Visual Studio Project File

1. Open *SimpleOven.pathmate* in the editor pane.
2. Verify that the deployment *Single Process C++* is selected in the Deployment pick list.



PIM: platform:/resource/SimpleOven/Model/SimpleOven.rpy

Deployment: Single Process C++

Name: Single Process C++

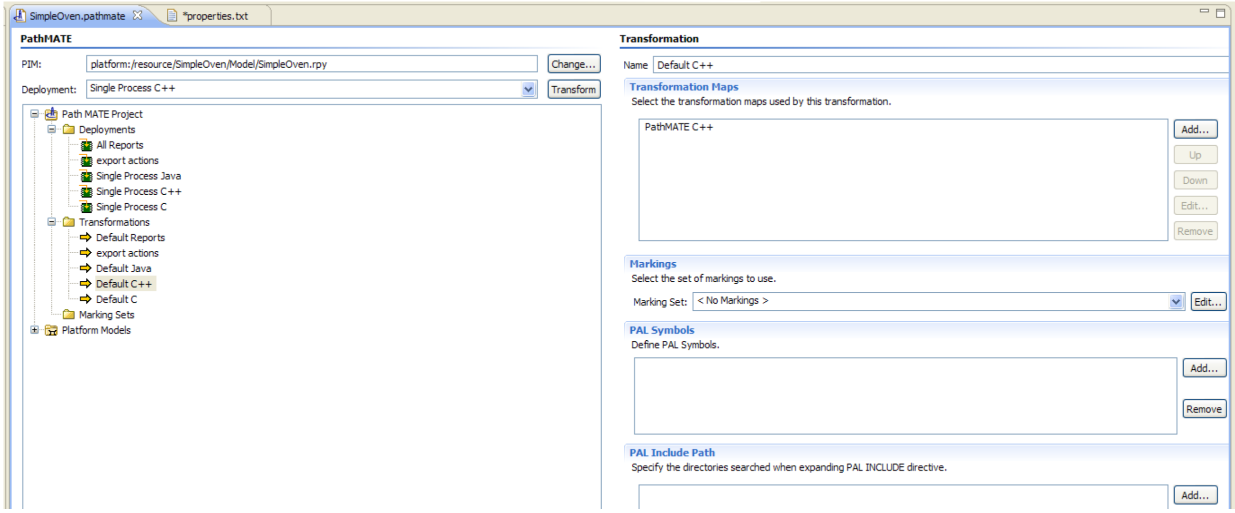
Working Dir: PROJECT_DIR/cpp

Resolved location: C:/path/workspaces/rha

Transformation: Default C++

```
PathMATE
Transforming Deployment 'Single Process C++'
Scope System
Project 'platform:/resource/SimpleOven/Models/SimpleOven.pathmate'
```

3. Click the **Edit…** button to the right of the Transformation drop down. In the Transformation Maps window ensure the *PathMATE C++* and *Build file generation* Maps appear in order. If not, use the Add and/or Up/Down buttons to get both maps to appear in the proper order.



PathMATE

PIM: platform:/resource/SimpleOven/Model/SimpleOven.rpy

Deployment: Single Process C++

- Path MATE Project
  - Deployments
    - All Reports
    - export actions
    - Single Process Java
    - Single Process C++
    - Single Process C
  - Transformations
    - Default Reports
    - export actions
    - Default Java
    - Default C++
    - Default C
  - Marking Sets
- Platform Models

Transformation

Name Default C++

Transformation Maps

Select the transformation maps used by this transformation.

PathMATE C++

Markings

Select the set of markings to use.

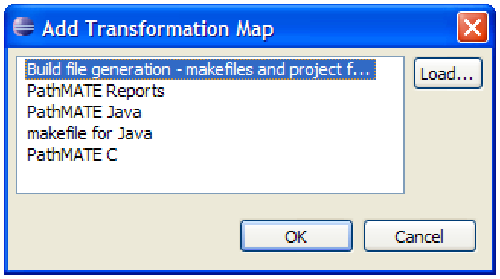
Marking Set: < No Markings >

PAL Symbols

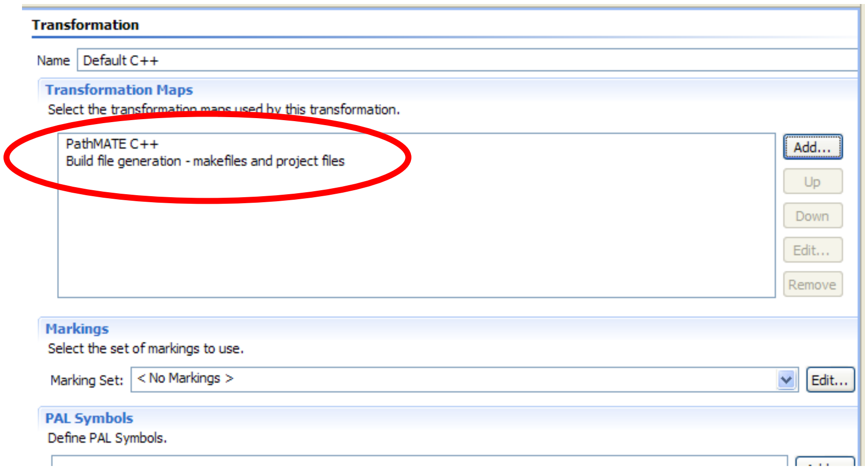
Define PAL Symbols.

PAL Include Path

Specify the directories searched when expanding PAL INCLUDE directive.



Build file generation - makefiles and project f…
PathMATE Reports
PathMATE Java
makefile for Java
PathMATE C



Transformation

Name Default C++

Transformation Maps

Select the transformation maps used by this transformation.

PathMATE C++
Build file generation - makefiles and project files

Markings

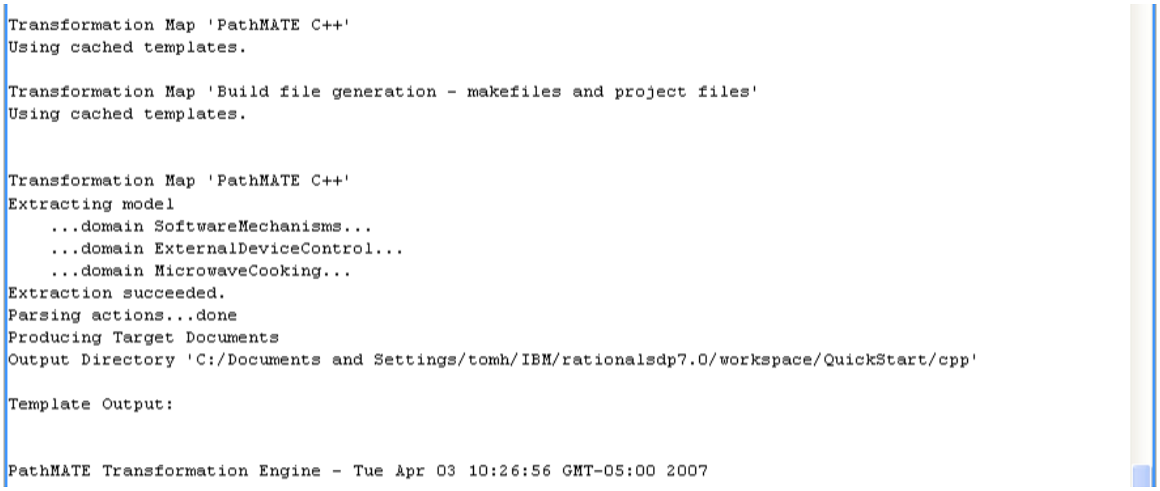
Select the set of markings to use.

Marking Set: < No Markings >

PAL Symbols

Define PAL Symbols.

4. Click **Transform**. A progress bar appears.
5. Verify in the Console tab that both transformations were successful. (If any problems arise at this point they are like due to *properties.txt* typos.)



```
Transformation Map 'PathMATE C++'
Using cached templates.

Transformation Map 'Build file generation - makefiles and project files'
Using cached templates.


Transformation Map 'PathMATE C++'
Extracting model
    ...domain SoftwareMechanisms...
    ...domain ExternalDeviceControl...
    ...domain MicrowaveCooking...
Extraction succeeded.
Parsing actions...done
Producing Target Documents
Output Directory 'C:/Documents and Settings/tomh/IBM/rationalsdp7.0/workspace/QuickStart/cpp'

Template Output:


PathMATE Transformation Engine - Tue Apr 03 10:26:56 GMT-05:00 2007
```

Task 3: Build an Executable System

In the resource perspective, browse to SimpleOven\cpp\MAIN. To launch the Visual C++ environment and build SimpleOven.exe:

PROCEDURE: Build SimpleOven.exe - Visual Studio Version 7+

1. Select **MAIN.sln**
2. Open it.
3. Build the SimpleOven system in Visual Studio.

PROCEDURE: Build SimpleOven.exe - Visual Studio Version 6

1. Right-click **SimpleOven.dsw** and select **Open With → System Editor.**
2. Build the SimpleOven system in Visual Studio 6.

Congratulations! You now have a complete C++ implementation and project file for your system!

Please skip the **Generate Java and Build with Eclipse JDT** section and move ahead to the **Run SimpleOven with Spotlight** section.

Generate Java and Build with Eclipse JDT

The Eclipse Java Development Toolkit (JDT) is included with the Eclipse installation that comes with PathMATE. This section takes you through the following steps:

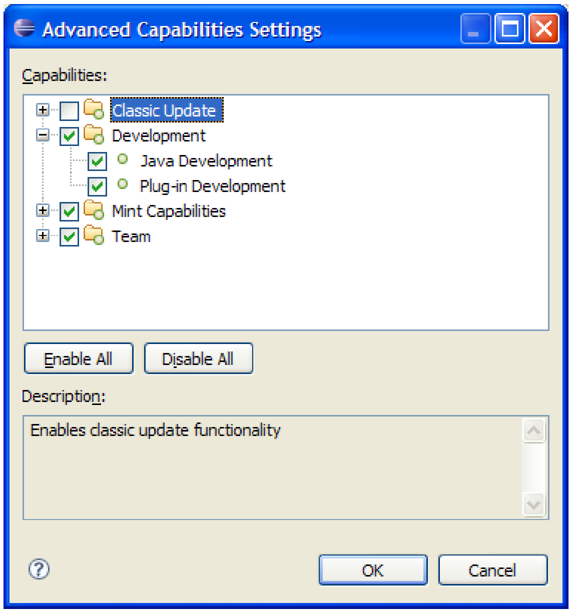
- Instantiate a Reference Project
- Create a JDT Project to build the generated code
- Create a PathMATE Project
- Transform Your Model
- Build an Executable System

Task 1: Set Up Java and Generate Java Code

Depending on the state of your Eclipse workbench, you may need to activate the JDT capability.

PROCEDURE: Enable Java Development in you Eclipse Workbench

1. Enable the Java Development capability. Select **Window → Preferences…** from the main menu bar. Select **General/ Capabilities** from the tree on the left side of the dialog. Click the **Advanced…** button.
2. Expand **Development** and ensure **Java Development** is checked.



Advanced Capabilities Settings

Capabilities:

- ☐ Classic Update
- ☑ Development
  - ☑ Java Development
  - ☑ Plug-in Development
- ☑ Mint Capabilities
- ☑ Team

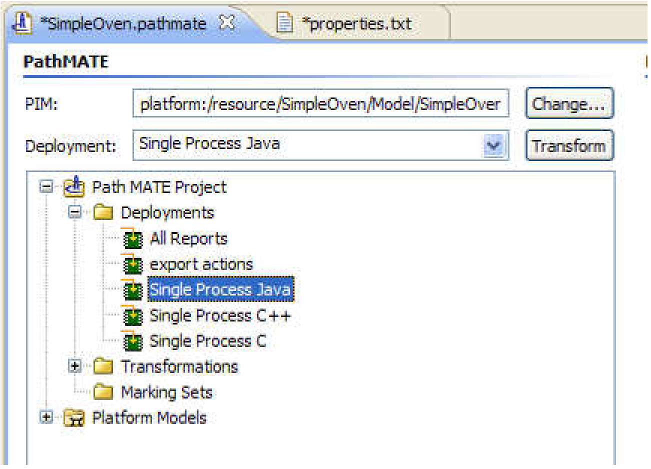
Description:

Enables classic update functionality

3. Click **OK** on both dialogs.

PROCEDURE: Generate Java Code

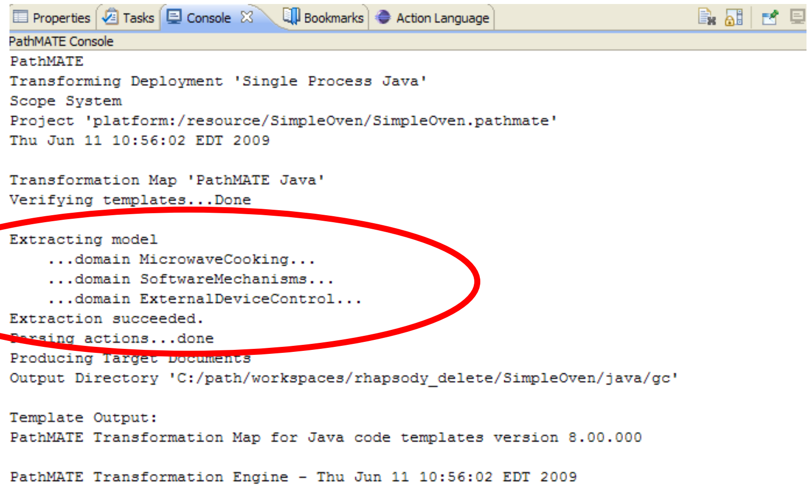
1. Open your PathMATE project file in your SimpleOven project. Select **Single Process Java** under the Deployments. Click **Transform**.



PIM: platform:/resource/SimpleOven/Model/SimpleOver

Deployment: Single Process Java

2. Verify no model extraction errors, or other errors are listed.



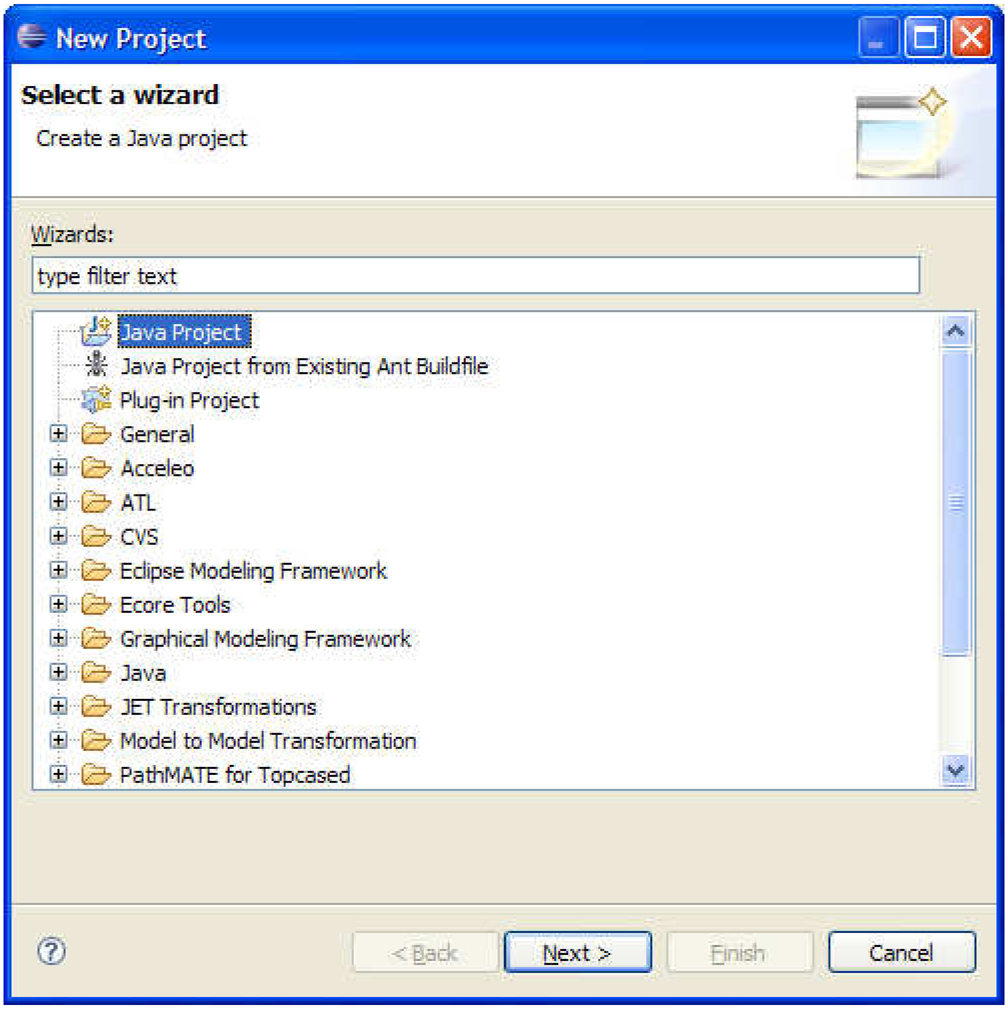
```
PathMATE
Transforming Deployment 'Single Process Java'
Scope System
Project 'platform:/resource/SimpleOven/SimpleOven.pathmate'
Thu Jun 11 10:56:02 EDT 2009

Transformation Map 'PathMATE Java'
Verifying templates...Done

Extracting model
    ...domain MicrowaveCooking...
    ...domain SoftwareMechanisms...
    ...domain ExternalDeviceControl...
Extraction succeeded.
Parsing actions...done
Producing Target Documents
Output Directory 'C:/path/workspaces/rhapsody_delete/SimpleOven/java/gc'

Template Output:
PathMATE Transformation Map for Java code templates version 8.00.000

PathMATE Transformation Engine - Thu Jun 11 10:56:02 EDT 2009
```

Task 2: Create a Java Project Ready the System for Debug

PROCEDURE: Create a Java Project and Build

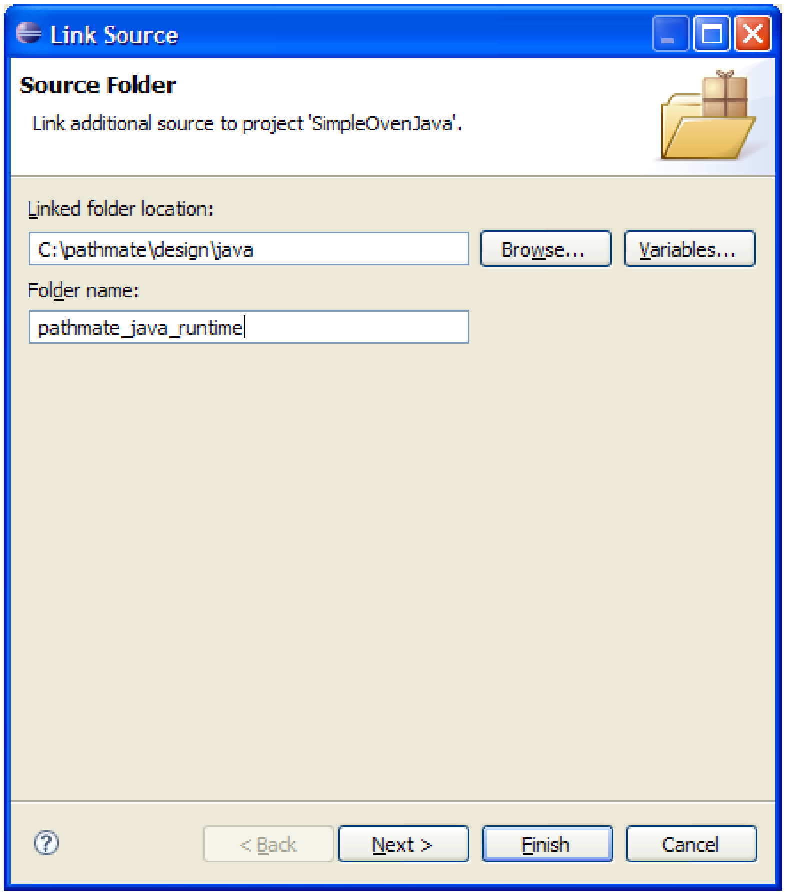
1. From the main Eclipse menu select **File → New → Project**.
2. The New Project Wizard Appears.

3. Select **Java / Java Project** from Wizards.



4. Click **Next**. The Create a Java Project page appears.
5. Enter the name *SimpleOvenJava*. Click **Finish**.
6. In the Project Explorer browser in your *SimpleOvenJava* project select the *src* source folder, right click and select **New → Package**.
7. Enter *realized_java* for the package name, and click **Finish**.
8. In the Project Explorer browser, navigate in the *SimpleOvenRef* project (created earlier) to the java/realized_java folder, select ExternalDeviceControlRealized.java, and hit control-C to copy it.
9. Back in your *SimpleOvenJava* project select **src/realized_java**, and hit control-V to paste in *ExternalDeviceControlRealized.java*.
10. In the Project Explorer browser, select your *SimpleOvenJava* project, right click, and select **Build Path → Configure Build Path**

11. Choose the Source tab and click **Link Source…**
12. Click **Browse…**, navigate to c:\pathmate\design and select the **java** folder.
13. Click **OK**.
14. Enter *pathmate_java_runtime* in Folder name.



Source Folder

Link additional source to project 'SimpleOvenJava'.

Linked folder location: C:\pathmate\design\java

Folder name: pathmate_java_runtime

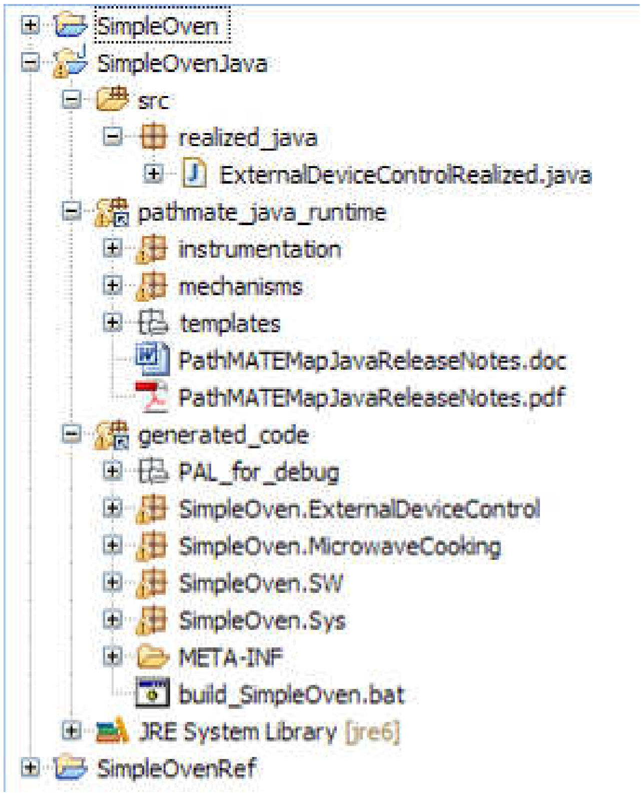
15. Click **Finish**.

16. Go back to your *SimpleOvenJava* project, right click, and select **Build Path → Configure Build Path**. Choose the Source tab and click **Link Source…** again.

17. Click **Browse…** and navigate to the generated code in your *SimpleOven* project: <your Eclipse workspace>/SimpleOven/ java/gc. Name this folder *generated_code*.

18. Click **Finish**.

19. Verify that your *SimpleOvenJava* project has these contents:



20. At this point your system should have automatically been built. Verify your Problems window reports no Errors. (Some Warnings are expected.)

PROCEDURE: Change Markings and Defines to Enable Spotlight Instrumentation

1. Open **SimpleOven/java/properties.txt**. The *properties.txt* file appears in the Editor pane.
2. Add the following to the top of *properties.txt* :

```
Domain,SimpleOven.*,SpotlightEnabled,T
```

3. Select **File** → **Save** from the main menu to save the changes.

The *markings* file *properties.txt* controls many aspects of transformation, including Spotlight debugger configuration.

1. In addition to generating instrumentation code (controlled by the SpotlightEnabled marking) you will need to enable Spotlight instrumentation in the runtime layer. In the Project Explorer view, expand **SimpleOvenJava** → **pathmate_java_runtime** → **mechanisms**.

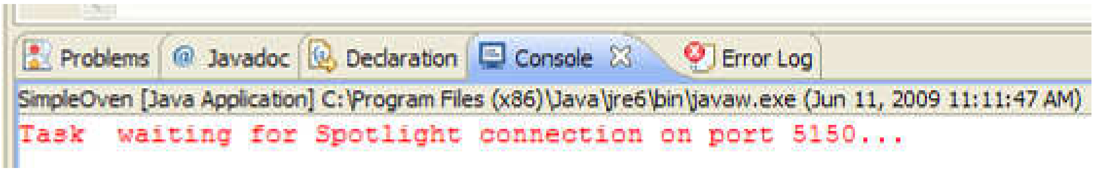
2. Double click to open **PfdDefine.java**. The file opens in the Editor pane. Set the *NO_PATH_IE* to *false* as shown below:

```
public static final boolean NO_PATH_IE = false;
```

3. Select **File → Save** from the main menu to save this change.

PROCEDURE: Create a Debug Configuration and Smoke Test it.

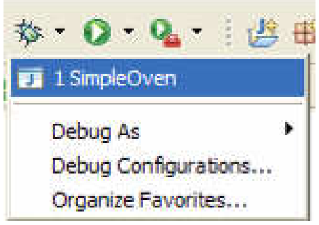
1. Go to **Window/Open Perspective/Other** to switch to the **Debug** perspective.
2. Expand debug toolbar menu .
3. Select Debug Configurations.
4. The Debug Configurations dialog appears.
5. Select **Java Application** from the Configurations tree.
6. Right click and select **New**.
7. Enter *SimpleOven* in the Name field.
8. Select *SimpleOvenJava* for the Project if it is not selected already.
9. In the Main Class section, press the **Search…** button. The Select Main Type dialog appears.
10. Select **SimpleOvenApp** from the Matching Types and click **OK**.
11. Click **Debug**. This starts the system. In the Console window you will see the program has started and is trying to connect to Spotlight:



```
SimpleOven [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Jun 11, 2009 11:11:47 AM)
Task waiting for Spotlight connection on port 5150...
```

12. Press the Terminate button . (We will connect to Spotlight in the next section).

To debug this application at a later time, go to the Debug button and select *1 SimpleOven*:



Congratulations! You now have a complete Java executable for your system!

Run SimpleOven with Spotlight

Task 1: Start the Simple Oven executable with Spotlight

Run the new program with the Spotlight to visualize SimpleOven system execution at the model level.

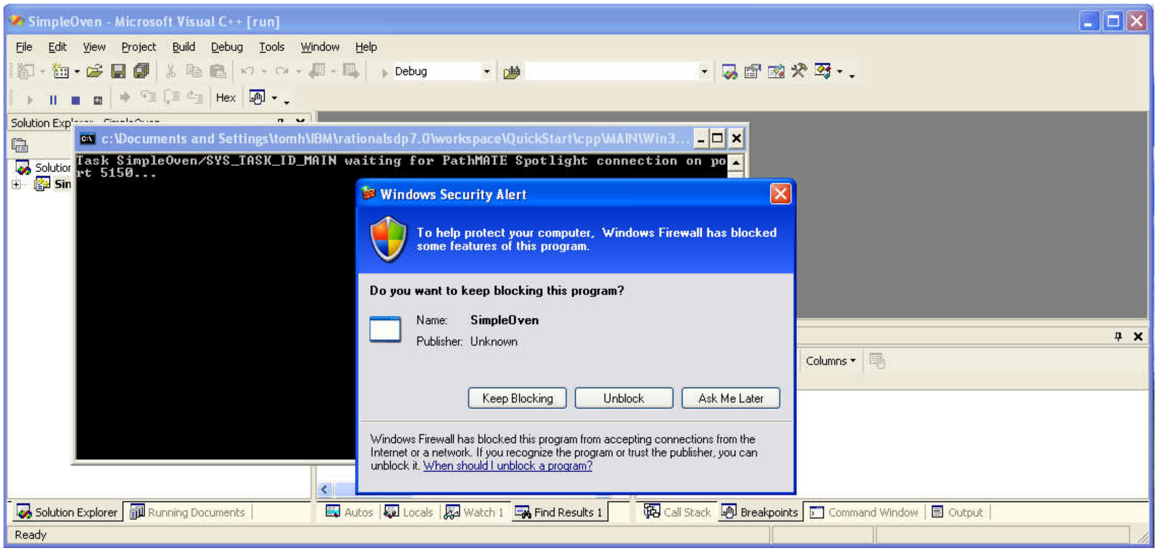
A command window opens to say that the application is running and waiting for a connection to Spotlight.

Please choose the appropriate Procedure below to start the SimpleOven executable, depending on whether you built it in C++ or Java.

PROCEDURE: Run C++ SimpleOven from Visual Studio

If you built Java, please skip this procedure.

1. Launch the application from within Visual Studio (usually **Debug → Start Debugging**, or the **F5** key). A command window opens to say that the application is running and waiting for a connection to Spotlight.



```
Task SimpleOven/SYS_TASK_ID_MAIN waiting for PathMATE Spotlight connection on po
rt 5150...
```

To help protect your computer, Windows Firewall has blocked some features of this program.

Do you want to keep blocking this program?

Name: SimpleOven
Publisher: Unknown

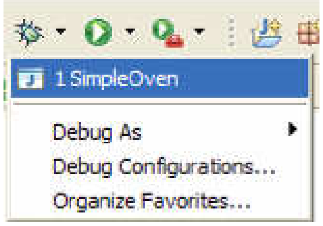
Windows Firewall has blocked this program from accepting connections from the Internet or a network. If you recognize the program or trust the publisher, you can unblock it. When should I unblock a program?

2. If a Windows Security Alert appears, click **Unblock**.
3. Skip ahead to PROCEDURE: Use Eclipse to Launch the Spotlight Debugger.

PROCEDURE: Run Java SimpleOven from the Eclipse JDT

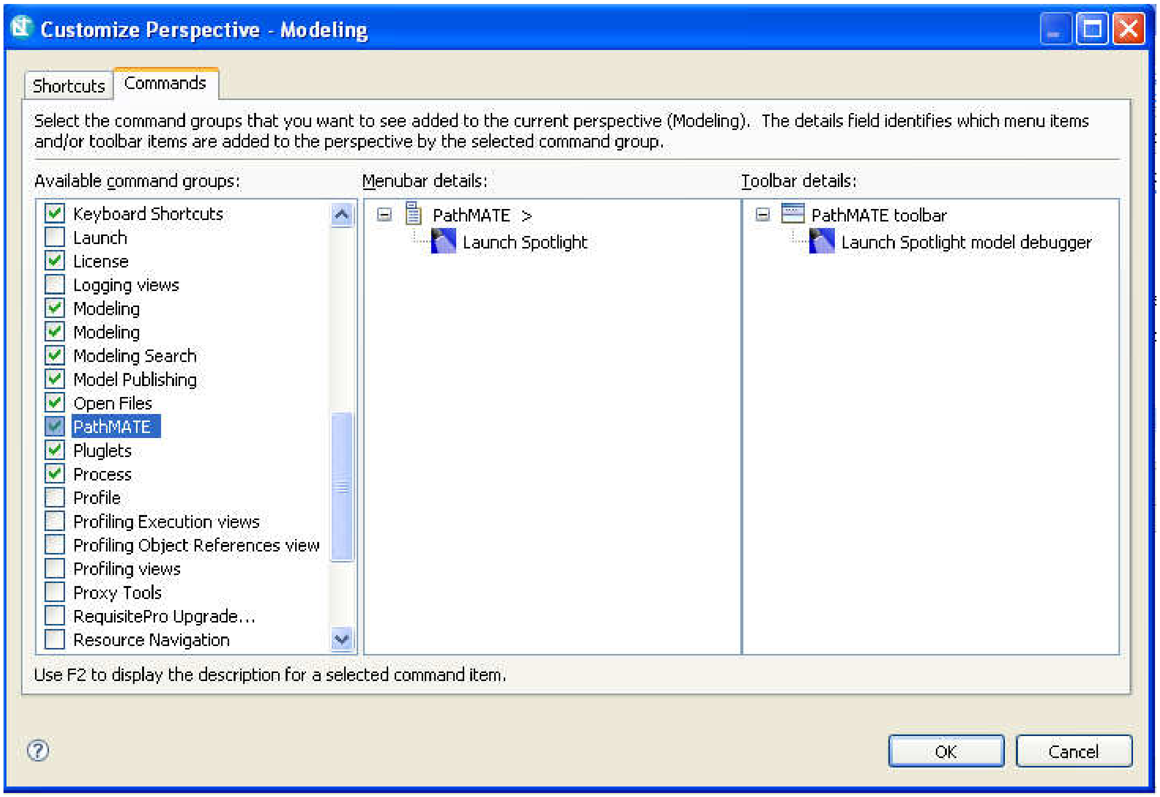
If you built C++, please skip this procedure.

1. Go to the Debug button and select **1 SimpleOven.**



PROCEDURE: Use Eclipse to Launch the Spotlight Debugger

1. If a PathMATE menu does not appear on the top menu bar, open the **Customize Perspective…** option in the Window menu. Select the **Commands** tab and check the **PathMATE** command group on the Commands tab:



Select the command groups that you want to see added to the current perspective (Modeling). The details field identifies which menu items and/or toolbar items are added to the perspective by the selected command group.

Available command groups:

- ☑ Keyboard Shortcuts
- □ Launch
- ☑ License
- □ Logging views
- ☑ Modeling
- ☑ Modeling
- ☑ Modeling Search
- ☑ Model Publishing
- ☑ Open Files
- ☑ PathMATE
- ☑ Pluglets
- ☑ Process
- □ Profile
- □ Profiling Execution views
- □ Profiling Object References view
- □ Profiling views
- □ Proxy Tools
- □ RequisitePro Upgrade...
- □ Resource Navigation

Menubar details:

- PathMATE >
  - Launch Spotlight

Toolbar details:

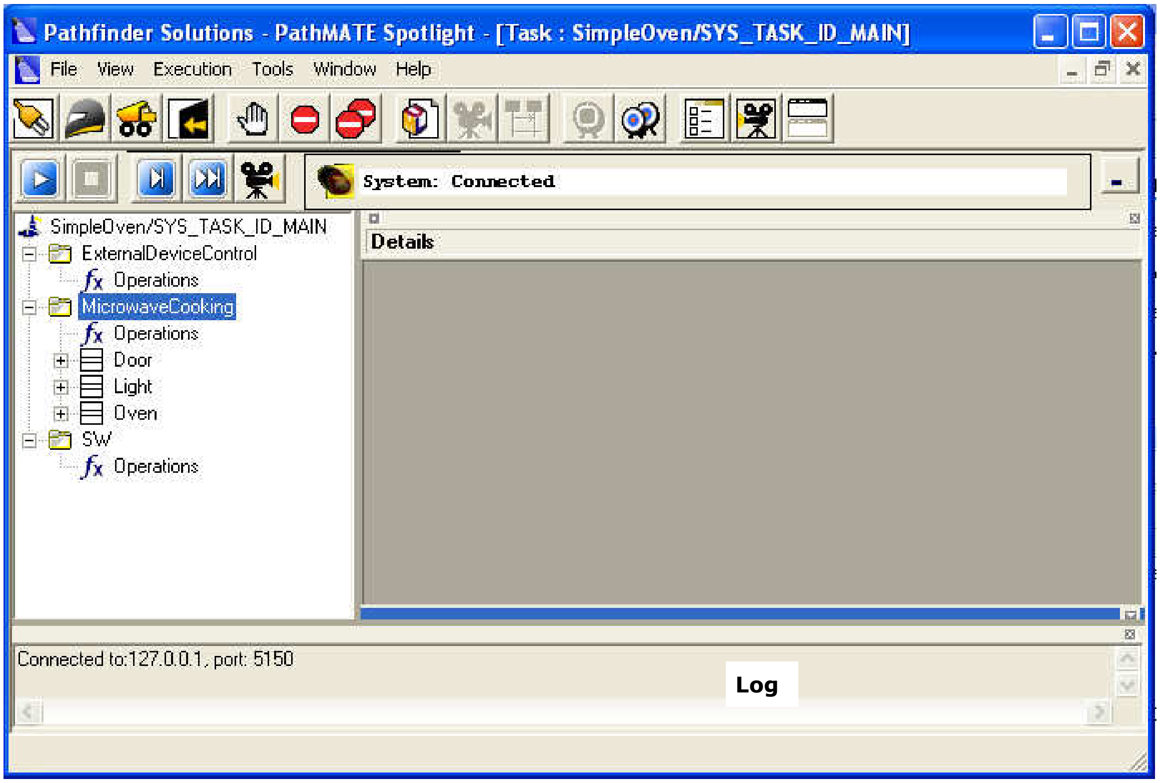
- PathMATE toolbar
  - Launch Spotlight model debugger

Use F2 to display the description for a selected command item.

2. Click **OK.**
3. To launch Spotlight, pick **PathMATE → Launch Spotlight** or choose the Eclipse toolbar **Launch Spotlight** button ( ).

PROCEDURE: Connect Spotlight to the Running SimpleOven Program

1. Once Spotlight starts, click the **Connect** button at the left end of the Spotlight toolbar to connect Spotlight to the target application.
2. Check to see that Spotlight is successfully connected. The three domains in *SimpleOven* appear in the browser on the left, and the status bar indicates *System: Connected*.
3. Expand the domains:

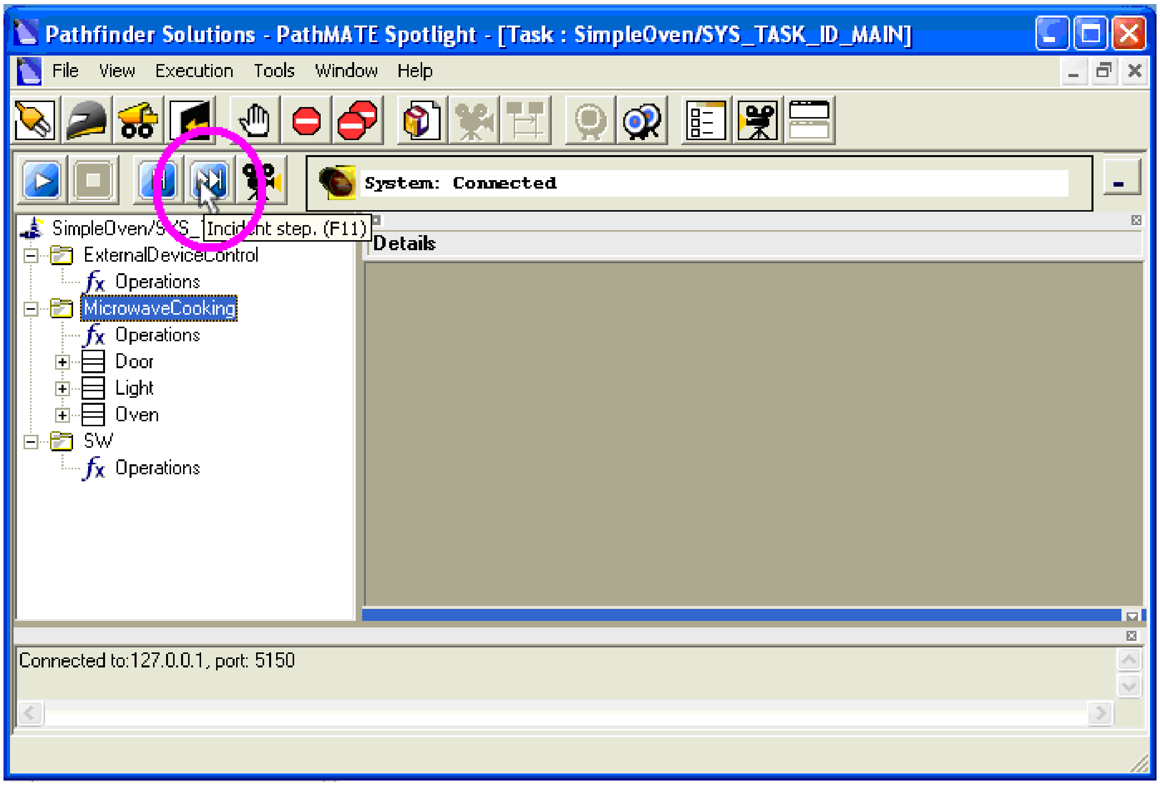


System: Connected

Connected to:127.0.0.1, port: 5150

PROCEDURE: Use Spotlight to Initialize the SimpleOven System

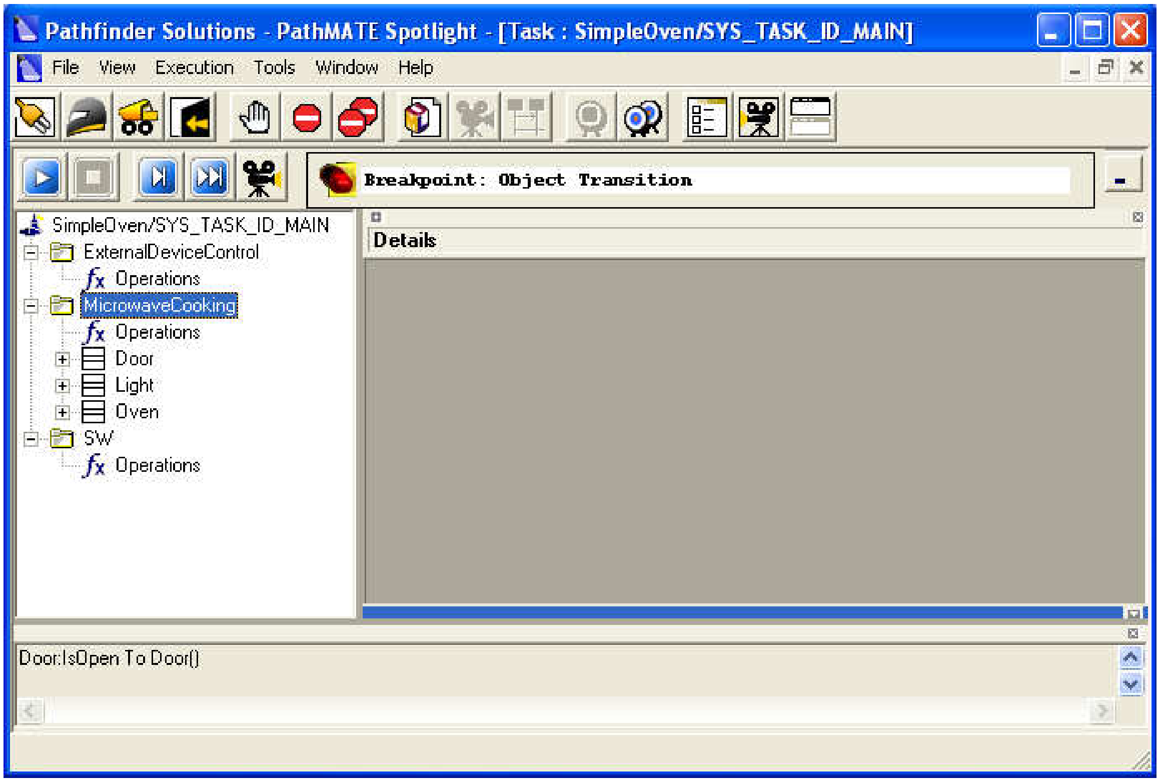
1. Locate the Incident Step button ⏭.



```
System: Connected
```

```
Connected to:127.0.0.1, port: 5150
```

2. Click the **Incident Step button** ⏭. The status indicator changes from *Connected* to *Object Transition*.



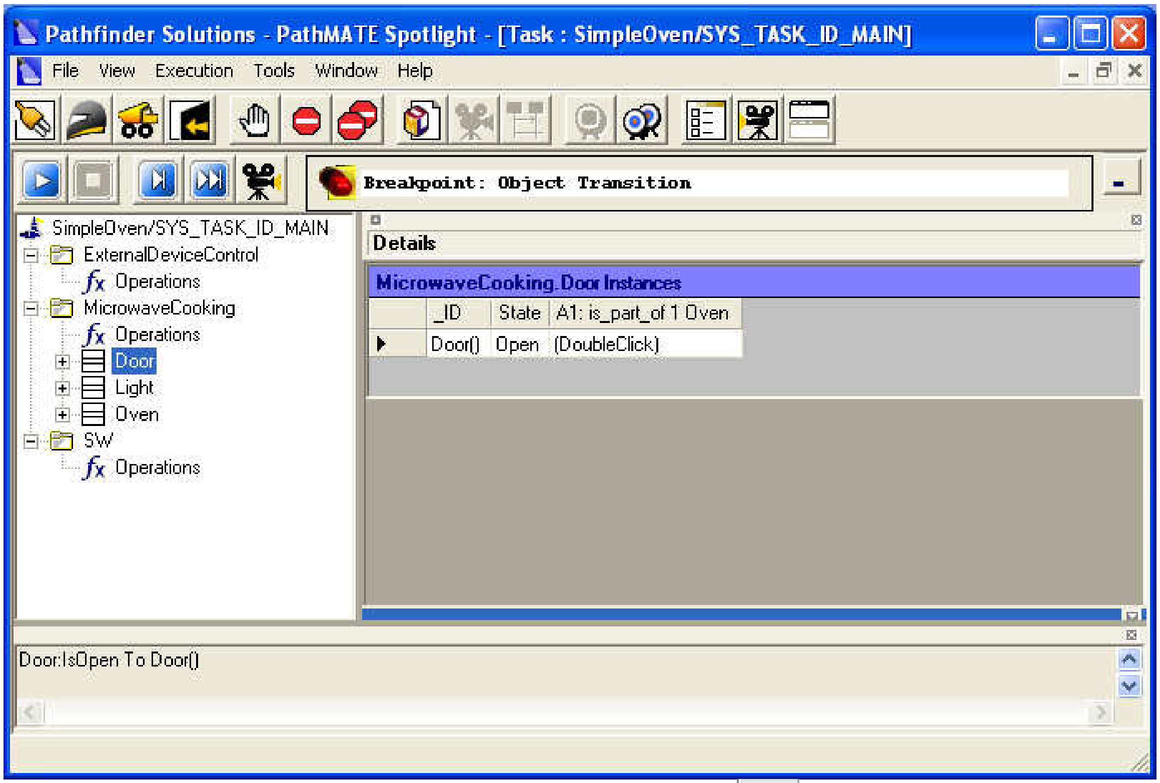
Breakpoint: Object Transition

Door:IsOpen To Door()

PROCEDURE: Use Spotlight to Browse the Door Class

1. Select the **Door** class in the browser.

The details pane shows one instance of the Door class. The door object is in the Open state, and it is part of one oven (association *A1*).



Pathfinder Solutions - PathMATE Spotlight - [Task : SimpleOven/SYS_TASK_ID_MAIN]
File View Execution Tools Window Help
SimpleOven/SYS_TASK_ID_MAIN
ExternalDeviceControl
Operations
MicrowaveCooking
Operations
Door
Light
Oven
SW
Operations
Details

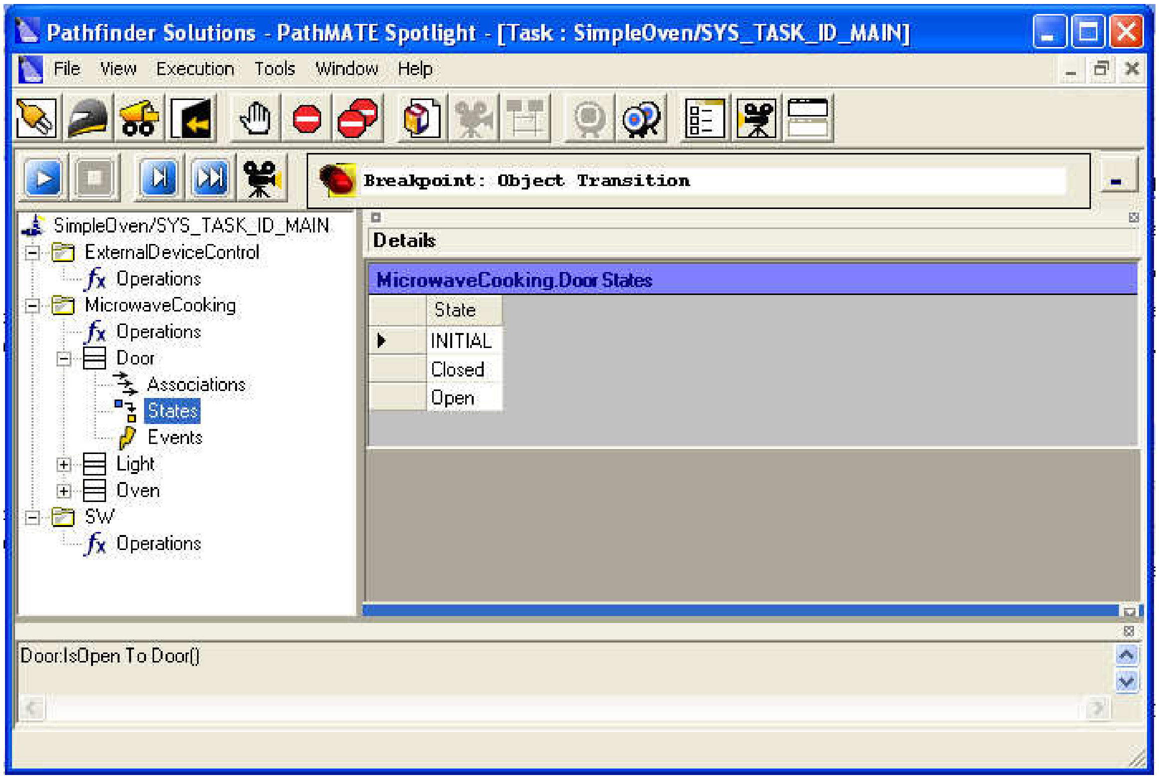
Breakpoint: Object Transition

MicrowaveCooking.Door Instances

_ID	State	A1: is_part_of 1 Oven
Door()	Open	(DoubleClick)

```
Door:IsOpen To Door()
```

2. Click the **Incident Step button** ⏭ to cause the Domain Initialization action to be executed and first Event to be processed.
3. Expand the **Door** class in the browser and review the run-time state of this class. Note the Door is in the Open State. Recall that the only transition out of the initial state is to the Closed state. To see why is the Door in the Open State, review the domain initialization action in the model, in the "Additional Initialization" property of the *<<Domain>>MicrowaveCooking*.



Pathfinder Solutions - PathMATE Spotlight - [Task : SimpleOven/SYS_TASK_ID_MAIN]
File
View
Execution
Tools
Window
Help
Details

Breakpoint: Object Transition

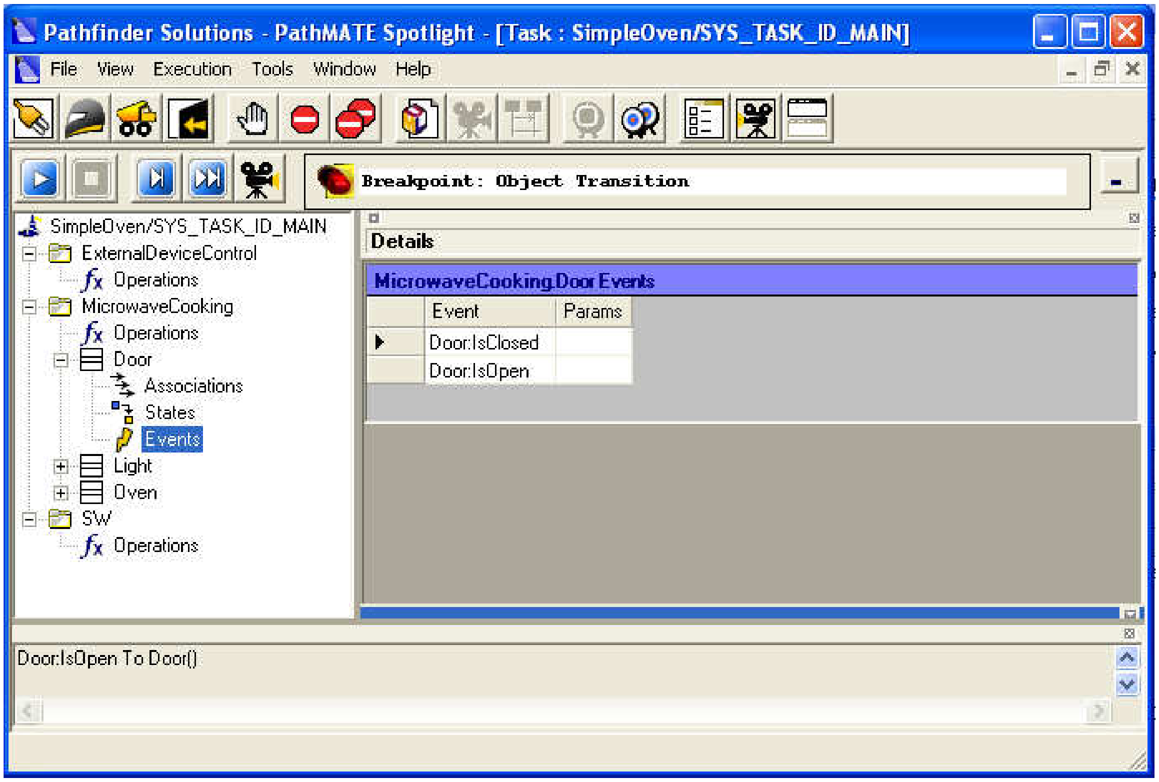
- SimpleOven/SYS_TASK_ID_MAIN
  - ExternalDeviceControl
    - Operations
  - MicrowaveCooking
    - Operations
    - Door
      - Associations
      - States
      - Events
    - Light
    - Oven
  - SW
    - Operations

MicrowaveCooking.Door States

	State
▶	INITIAL
	Closed
	Open

Door:IsOpen To Door()

4. Select **Events** in the browser. The details pane shows the events that you can send to the *Door* class.



Pathfinder Solutions - PathMATE Spotlight - [Task : SimpleOven/SYS_TASK_ID_MAIN]

Breakpoint: Object Transition

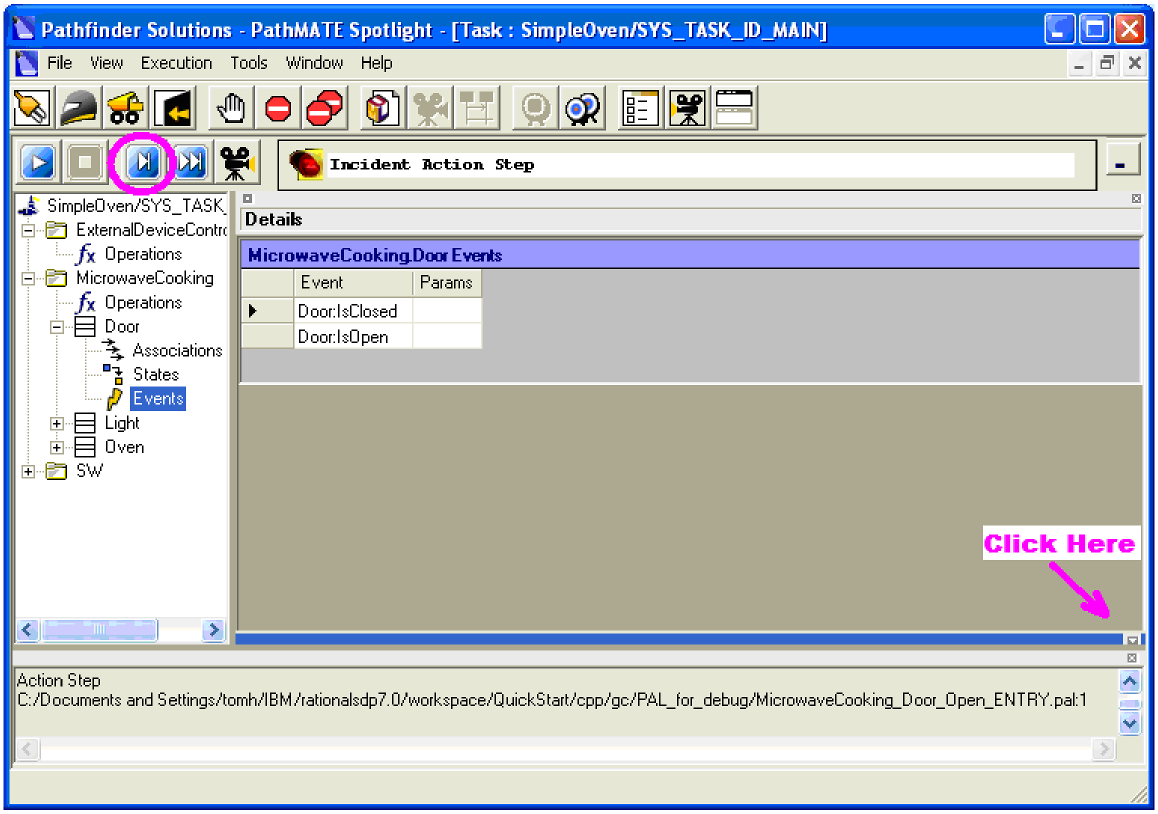
MicrowaveCooking.Door Events

Event	Params
Door:IsClosed	
Door:IsOpen	

Door:IsOpen To Door()

PROCEDURE: Use Spotlight to step through SimpleOven's Action Language

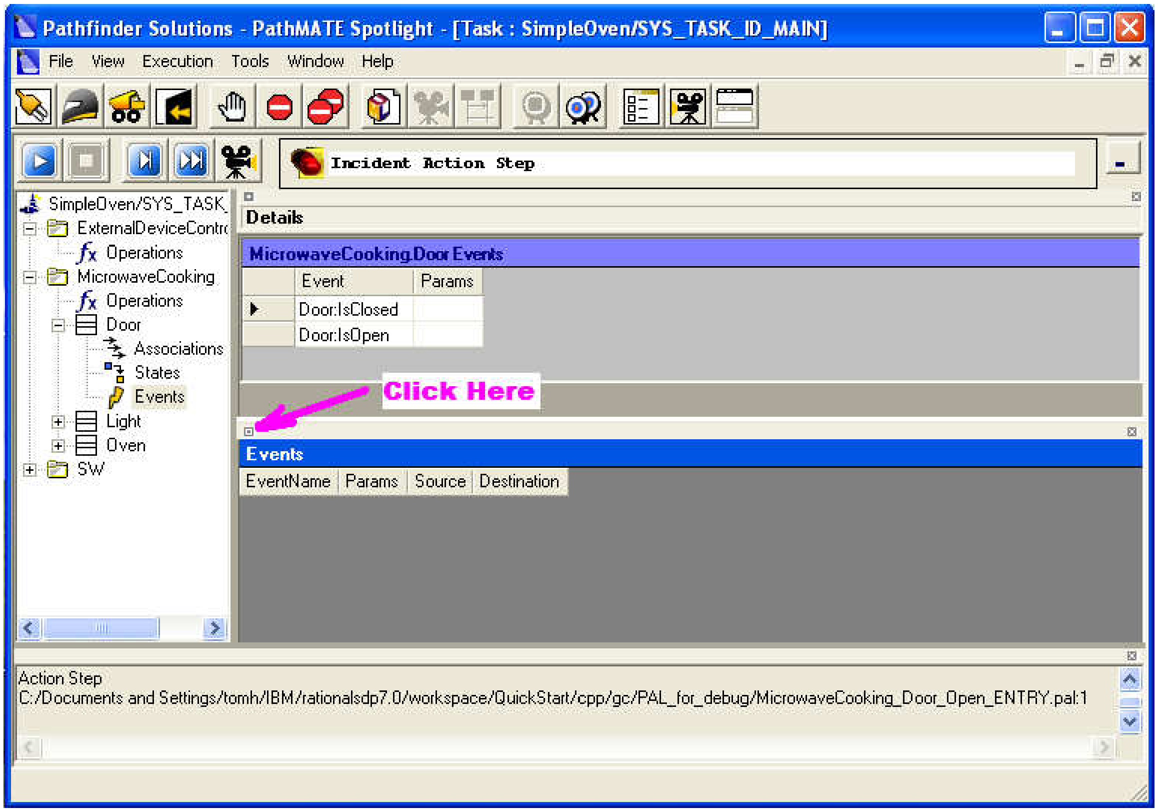
1. Click the **Action Step button** . *MicrowaveCooking_Door_Open.pal* appears in the state window.
2. Open the **State Window**. The figure below shows where to click in the lower right to open and close the action/event window.



Event	Params
Door:IsClosed	
Door:IsOpen	

```
Action Step
C:/Documents and Settings/tomh/IBM/rationalsdp7.0/workspace/QuickStart/cpp/gc/PAL_for_debug/MicrowaveCooking_Door_Open_ENTRY.pal:1
```

3. Select the **Action Language view**. The figure below shows how to toggle between action language and the event queue.



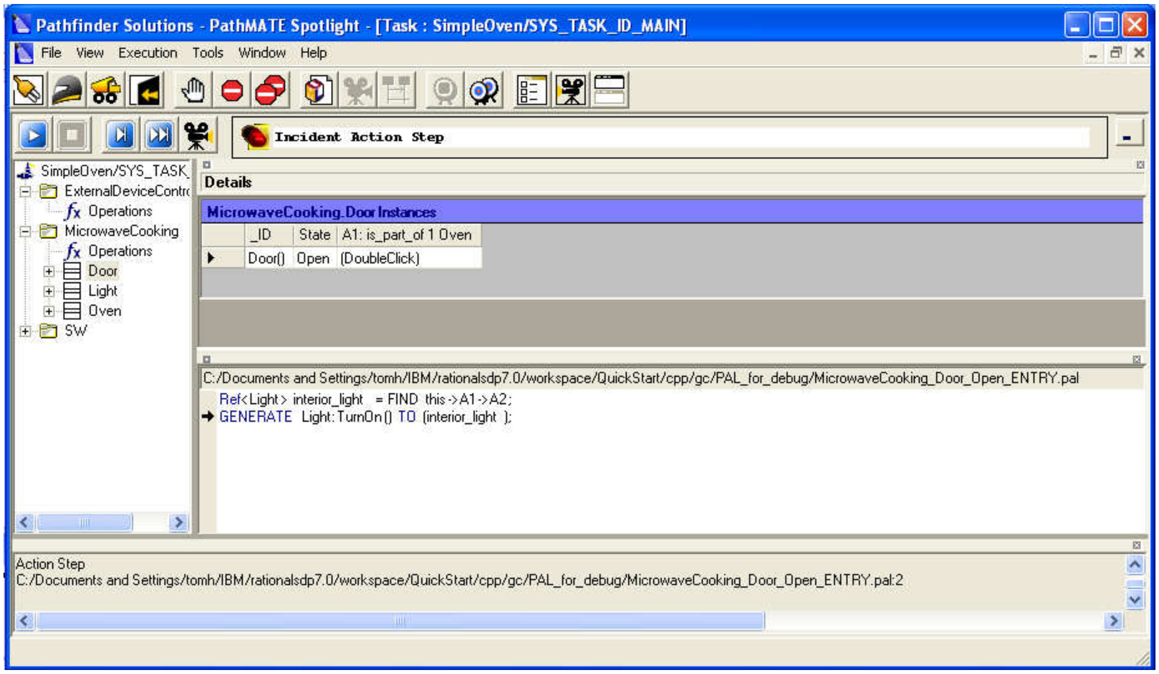
MicrowaveCooking.Door Events

	Event	Params
▶	Door:IsClosed	
	Door:IsOpen	

EventName	Params	Source	Destination

```
Action Step
C:/Documents and Settings/tomh/IBM/rationalsdp7.0/workspace/QuickStart/cpp/gc/PAL_for_debug/MicrowaveCooking_Door_Open_ENTRY.pal:1
```

4. Click **Action Step** again. The active step arrow advances to the next PAL statement.


Pathfinder Solutions - PathMATE Spotlight - [Task : SimpleOven/SYS_TASK_ID_MAIN]
File View Execution Tools Window Help
Incident Action Step
SimpleOven/SYS_TASK_
ExternalDeviceContr
Operations
MicrowaveCooking
Operations
Door
Light
Oven
SW
Details

MicrowaveCooking.Door Instances

	_ID	State	A1: is_part_of 1 Oven
▶	Door()	Open	(DoubleClick)

```
C:/Documents and Settings/tomh/IBM/rationalsdp7.0/workspace/QuickStart/cpp/gc/PAL_for_debug/MicrowaveCooking_Door_Open_ENTRY.pal
   Ref<Light> interior_light  = FIND this->A1->A2;
➔ GENERATE Light:TurnOn() TO (interior_light );
```

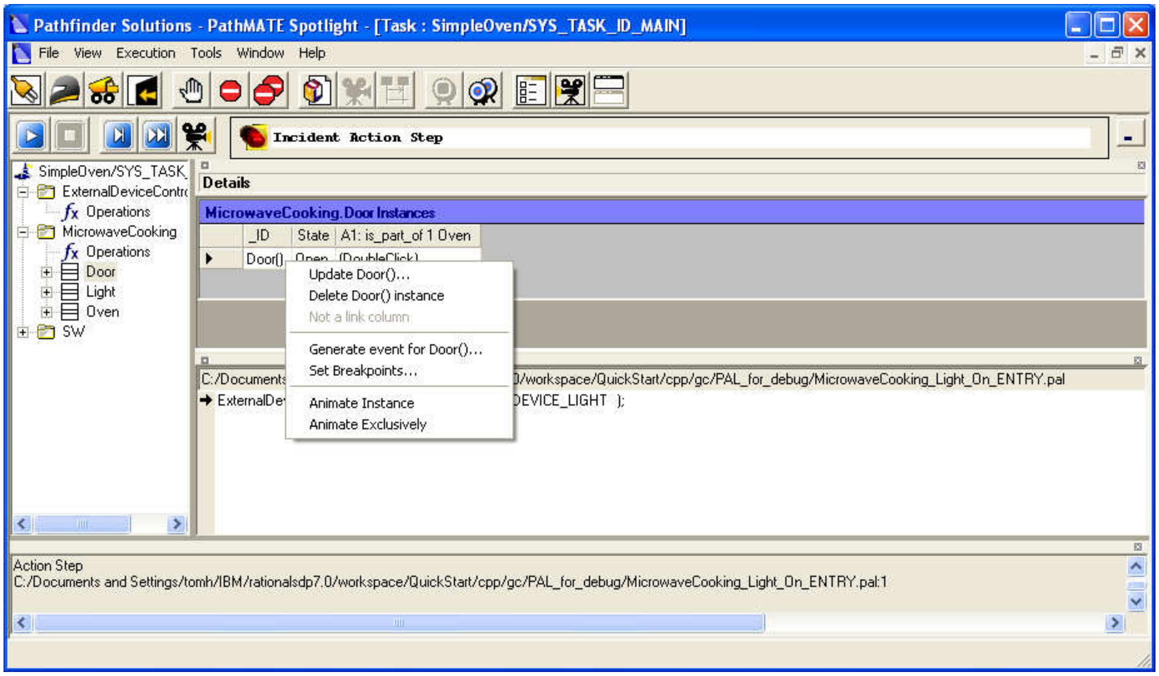
```
Action Step
C:/Documents and Settings/tomh/IBM/rationalsdp7.0/workspace/QuickStart/cpp/gc/PAL_for_debug/MicrowaveCooking_Door_Open_ENTRY.pal:2
```

5. Continue to click the **Action Step button** until *MicrowaveCooking_Light_On_entry.pal* appears in the PAL viewer.

(DO NOT CLICK AGAIN. If you do, you will queue up an action. Incident Action Step will change to Action Step Pending and the Red stoplight will change to Green. If this happens, all is not lost, but the Event Processing in the next procedure will proceed immediately on the event hitting the queue.)

PROCEDURE: Use Spotlight to Send Event (Signal) to SimpleOven

1. Select **Door** in the browser. Right-click the **Door***()* field in the details pane to bring up a context-sensitive menu:



Incident Action Step

MicrowaveCooking.Door Instances

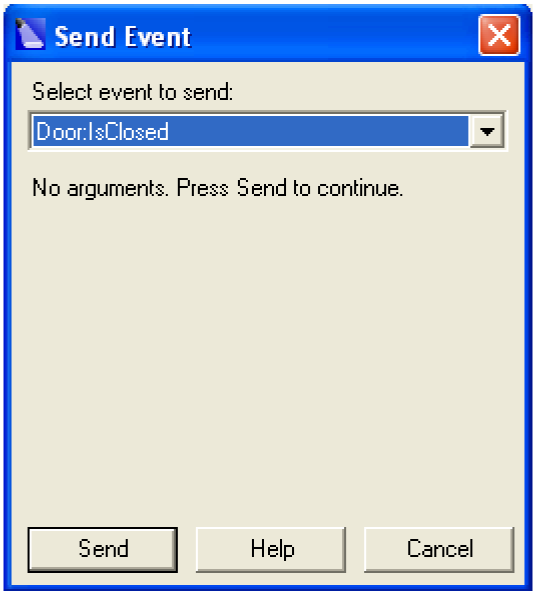
	_ID	State	A1: is_part_of 1 Oven
▶	Door()	Open	(DoubleClick)

C:/Documents … 0/workspace/QuickStart/cpp/gc/PAL_for_debug/MicrowaveCooking_Light_On_ENTRY.pal

➔ ExternalDe… DEVICE_LIGHT);

Action Step
C:/Documents and Settings/tomh/IBM/rationalsdp7.0/workspace/QuickStart/cpp/gc/PAL_for_debug/MicrowaveCooking_Light_On_ENTRY.pal:1

2. Select **Generate event for Door()** in the pop-up menu. The Send Event dialog opens. Select **Door:IsClosed** in the drop-down list.

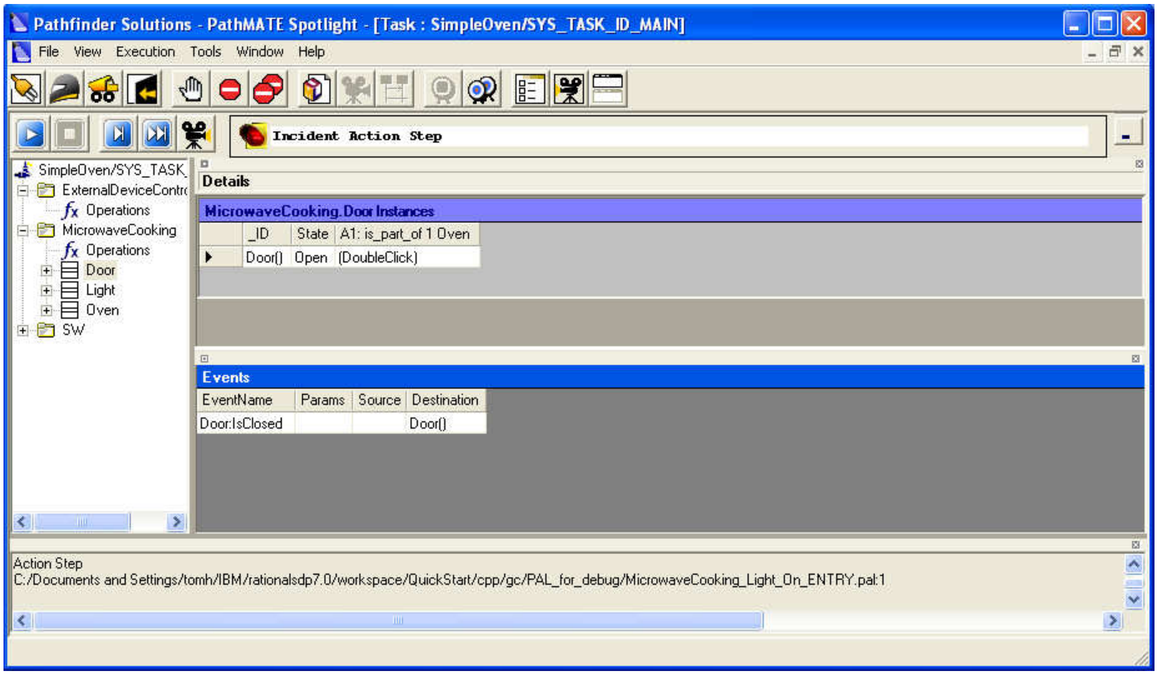


Select event to send:

Door:IsClosed

No arguments. Press Send to continue.

3. Click **Send**.
4. Toggle from the Action Language viewer to the event queue in the lower pane. The event *Door:IsClosed* re queued to be sent to Door(). The event appears in the queue under Events in the lower pane.



MicrowaveCooking.Door Instances

	_ID	State	A1: is_part_of 1 Oven
▶	Door()	Open	(DoubleClick)

Events

EventName	Params	Source	Destination
Door:IsClosed			Door()

Action Step
C:/Documents and Settings/tomh/IBM/rationalsdp7.0/workspace/QuickStart/cpp/gc/PAL_for_debug/MicrowaveCooking_Light_On_ENTRY.pal:1

5. Press the Spotlight **Go button** ▶ to resume execution, and allow *SimpleOven* to process the new event.

6. You may choose to continue execution by resending *Door:IsOpen* and *Door:IsClosed* events.

Generate System Documentation

PROCEDURE: Generate Documentation

Return to PathMATE and transform:

1. Select the **All Reports** deployment:
2. Select **Transform**.

From the resource perspective explore the generated reports in the reports folder. (Note: a benign warning “Properties file properties.txt not found.” will be written to the Eclipse Console View.)

Generated documentation files:

Report Title	Filename
Domain Report for SimpleOven	SimpleOven_summary.rtf
Full Analysis Report for SimpleOven	SimpleOven.rtf
Class Modeling Report for SimpleOven.MicrowaveCooking	SimpleOven_MicrowaveCooking_summary.rtf
State Modeling report for SimpleOven.MicrowaveCooking	SimpleOven_MicrowaveCooking.rtf

3. To view any of the generated files, right-click on the file in the Project Explorer and select **Open With → System Editor**.

Congratulations!
You have documentation and reports for your SimpleOven system.

You have now completed the Quick Start.

Summary

PathMATE separates the feature logic of a system from the details of its implementation on a specific platform. That separation yields simplicity, facilitating the solution of each aspect of the system in relative isolation from the others. The simplicity of platform independent models yields substantial benefits all across the development cycle. With PathMATE you will:

- Improve productivity in your initial development effort.
- React quickly to changing software and hardware requirements.
- Test integration of all system components at the model level, much earlier in the development cycle.
- Substantially reduce the time required for debugging.
- Improve reliability and performance.
- Consistently meet your deadlines.

Additionally, the use of platform-independent action language gives PathMATE a complete semantic awareness of everything that your system will do, allowing it to do Self Optimization of your system as it generates your implementation code. This yields substantial performance gains over code generated from code-in-the-model approaches.

Pathfinder's tools help you transform your models into executable code, predictably and accurately. Deploy faster, highly reliable embedded systems much more quickly than you thought possible.

Next Steps

The white papers at www.pathfindermdd.com contain additional information. Most importantly, try the PathMATE toolset with real code, as outlined in this *Guide*. We can help you get started.

If you are interested in the use of modeling for Systems Engineering, be sure to visit www.pathfindersystemsmodeling.com.

Pathfinder Solutions

Headquartered in Foxboro, Massachusetts, Pathfinder Solutions provides embedded software engineers with the tools, methods and services needed to reduce development costs and improve quality.

Please contact us at:

Pathfinder Solutions
Phone: 888-662-7284 (+1 508-568-0068)
Email: info@pathfindermdd.com

Acronyms

Acronym	**Definition**
JDT	Java Developer Toolkit
MDA	Model Driven Architecture
MDD	Model Driven Development
OMG	Object Management Group
PAL	Platform-independent Action Language
PathMATE	Pathfinder Model Automation and Transformation Environment
PIM	Platform Independent Model
UML	Unified Modeling Language