



PathMATE™ for Topcased Modeler version 4.3

Quick Start Guide

Version 1.42

June 29, 2011

PathMATE™ Series

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

Table of Contents

OVERVIEW	4
HOW TO USE THIS GUIDE	4
AUDIENCE	4
ADDITIONAL RESOURCES	4
ECLIPSE	5
CONVENTIONS	5
WHAT YOU’LL NEED	5
PATHMATE PI-MDD TOOLSET	1
DOWNLOAD AND INSTALL PATHMATE	3
ECLIPSE AND TOPCASED	4
CREATE A PLATFORM INDEPENDENT MODEL FOR SIMPLEOVEN	5
ECLIPSE WORKSPACES.....	5
TASK 1: CREATE A PATHMATE UML PROJECT USING THE PATHMATE PROJECT TEMPLATE	5
<i>PROCEDURE: Use PathMATE New Project Wizard in Eclipse</i>	5
<i>PROCEDURE: Use the Properties view to rename the System Model</i>	9
TASK 2: ADD THE MODELED DOMAIN <i>MICROWAVECOOKING</i>	10
<i>PROCEDURE: Use the PathMATE Domain Template Generator to Add the MicrowaveCooking Domain</i>	10
<i>PROCEDURE: Add a Domain Service to the MicrowaveCooking Domain</i>	12
<i>PROCEDURE: Specify ReportDoorStatus Interface Operation Action Language</i> ...	13
TASK 3: CREATE A REALIZED DOMAIN.....	15
<i>PROCEDURE: Use the PathMATE Domain Template Generator to Add the ExternalDeviceControl Domain</i>	15
<i>PROCEDURE: Add a UML Enumeration to ExternalDeviceControl's Public Types</i> ..	16
<i>PROCEDURE: Add a Domain Service to the ExternalDeviceControl Domain</i>	16
TASK 4: COMPLETE THE SYSTEM DOMAIN CHART	16
<i>PROCEDURE: Use System Domain Chart to Specify the Domain Hierarchy and their Dependencies</i>	17
<i>PROCEDURE: Specify System Package Imports</i>	18
<i>PROCEDURE: Capture a System-Level Scenario Model</i>	19
TASK 5: COMPLETE THE CLASS DIAGRAM FOR THE <i>MICROWAVECOOKING</i> DOMAIN	21
<i>PROCEDURE: Create the Oven, Door and Light Classes in the MicrowaveCooking Domain</i>	21
<i>PROCEDURE: Add Attributes to the Oven and Light Classes</i>	22
<i>PROCEDURE: Associate the Classes</i>	24
TASK 6: COMPLETE THE <i>MICROWAVECOOKING</i> DOMAIN INITIALIZATION ACTION	26
<i>PROCEDURE: Add MicrowaveCooking Initialization Action</i>	26
TASK 7: CREATE THE DOOR STATE MACHINE.....	26
<i>PROCEDURE: Create and Name the Door State Machine Diagram</i>	26
<i>PROCEDURE: Lay out Door States</i>	26
<i>PROCEDURE: Change State Machine Diagram Transition Preferences</i>	27
<i>PROCEDURE: Draw an Untriggered Transition</i>	28
<i>PROCEDURE: Create Signals, Triggers and Draw a Triggered Transition</i>	29
<i>PROCEDURE: Capture Door State Entry Actions</i>	31
TASK 8: CREATE THE LIGHT STATE MACHINE.....	32
<i>PROCEDURE: Create the State Machine Diagram and States</i>	32
<i>PROCEDURE: Draw Transitions</i>	32

<i>PROCEDURE: Create Signals, Triggers, and apply them</i>	33
<i>PROCEDURE: Create State Entry Actions</i>	33
PREPARE FOR TRANSFORMATION	35
TASK 1: REUSE ELEMENTS FROM EXAMPLE PROJECT	35
<i>PROCEDURE: Instantiate a Reference Project for SimpleOven</i>	35
<i>PROCEDURE: Use Eclipse Resource Perspective to Copy Files Needed for C++ and Java Systems</i>	36
TASK 2: CONNECT PATHMATE PROJECT TO YOUR NEW PIM.....	36
<i>PROCEDURE: Create a New PathMATE Project</i>	36
TASK 3: CHECK YOUR MODEL	37
<i>PROCEDURE: Use the PathMATE Project Editor to test Your Project</i>	37
GENERATE C++ CODE AND VISUAL STUDIO PROJECT FILES	40
TASK 1: CONFIGURE DEVELOPMENT ENVIRONMENT OPTIONS	40
<i>PROCEDURE: Use Eclipse Text Editor to Enable Spotlight Instrumentation by Changing "Markings" in properties.txt</i>	40
<i>PROCEDURE: Change Markings to select versions of Visual Studio</i>	41
TASK 2: TRANSFORM YOUR MODEL TO C++ CODE AND VISUAL STUDIO PROJECT FILES	41
<i>PROCEDURE: Generate C++ Implementation Code and a Visual Studio Project File</i>	41
TASK 3: BUILD AN EXECUTABLE SYSTEM	43
<i>PROCEDURE: Build SimpleOven.exe - Visual Studio Version 7</i>	44
<i>PROCEDURE: Build SimpleOven.exe - Visual Studio Version 6</i>	44
GENERATE JAVA AND BUILD WITH ECLIPSE JDT	45
TASK 1: SET UP JAVA AND GENERATE JAVA CODE.....	45
<i>PROCEDURE: Enable Java Development in you Eclipse Workbench</i>	45
<i>PROCEDURE: Generate Java Code</i>	46
TASK 2: CREATE A JAVA PROJECT READY THE SYSTEM FOR DEBUG	47
<i>PROCEDURE: Create a Java Project and Build</i>	47
<i>PROCEDURE: Change Markings and Defines to Enable Spotlight Instrumentation</i>	51
<i>PROCEDURE: Create a Debug Configuration and Smoke Test it</i>	52
RUN SIMPLEOVEN WITH SPOTLIGHT	53
TASK 1: START THE SIMPLE OVEN EXECUTABLE WITH SPOTLIGHT	53
<i>PROCEDURE: Run C++ SimpleOven from Visual Studio</i>	53
<i>PROCEDURE: Run Java SimpleOven from the Eclipse JDT</i>	54
<i>PROCEDURE: Use Eclipse to Launch the Spotlight Debugger</i>	54
<i>PROCEDURE: Connect Spotlight to the Running SimpleOven Program</i>	55
<i>PROCEDURE: Use Spotlight to Initialize the SimpleOven System</i>	56
<i>PROCEDURE: Use Spotlight to Browse the Door Class</i>	58
<i>PROCEDURE: Use Spotlight to step through SimpleOven's Action Language</i>	60
<i>PROCEDURE: Use Spotlight to Send Event (Signal) to SimpleOven</i>	63
GENERATE SYSTEM DOCUMENTATION	66
<i>PROCEDURE: Generate Documentation</i>	66
SUMMARY	68

NEXT STEPS.....	68
PATHFINDER SOLUTIONS.....	68
ACRONYMS	69

Overview

This document is provided as a first step in learning how to use PathMATE with the Topcased Modeler 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 MDD (PI-MDD) approach for building systems 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 portal at www.pathfindermda.com.

Continue to learn how PathMATE works by completing the walk-through 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 Topcased Modeler toolset.

Additional Resources

The Pathfinder Solutions website at www.pathfindermda.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@pathfindermda.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

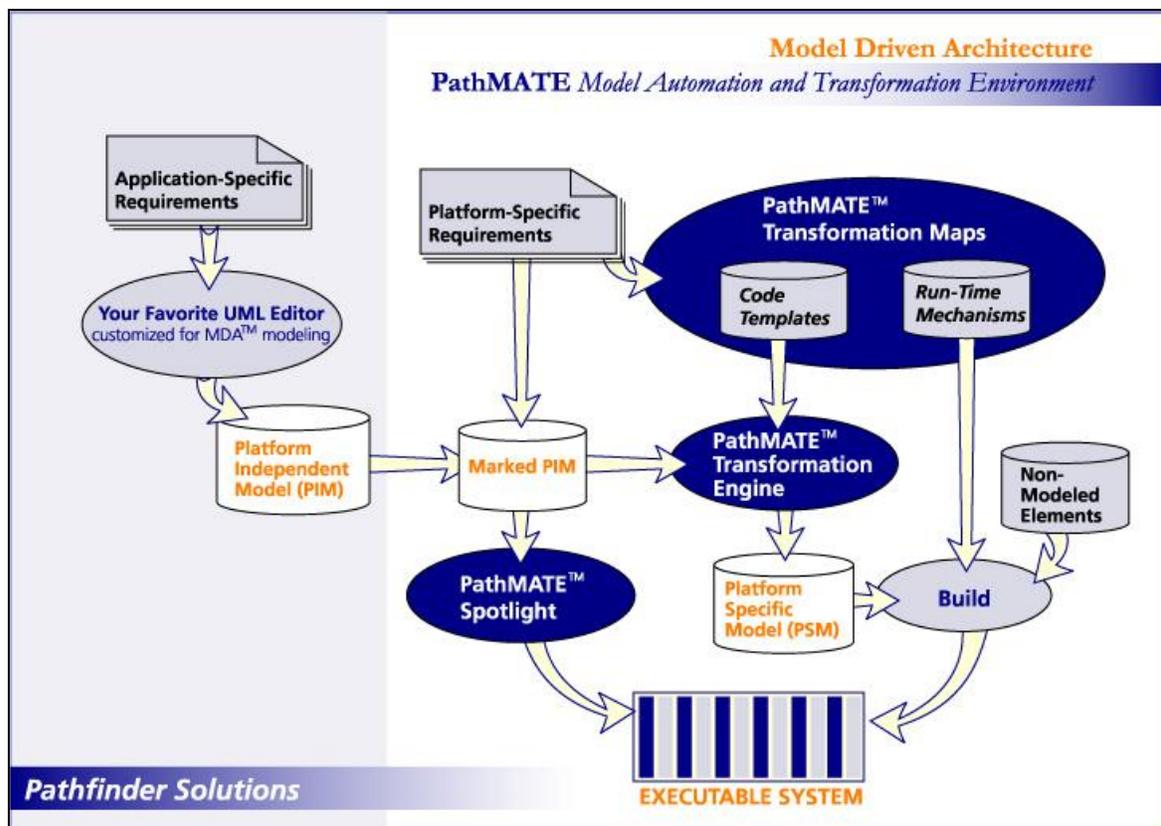
To complete the steps in this guide, you'll need the following software on your computer:

- PathMATE for Topcased software development toolkit
- Windows
 - Microsoft Windows 2000 or Windows XP Professional Edition or Windows Vista
 - Microsoft Visual C++ version 6+
- Linux
 - Make and a C, CPP, or JAVA compiler

PathMATE PI-MDD Toolset

Through its *Architectural Focus* and advanced automation, PathMATE is the leading MDD environment for embedded, real-time systems development. This powerful technology automates the verification, execution and deployment of Platform-Independent MDD 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:



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.

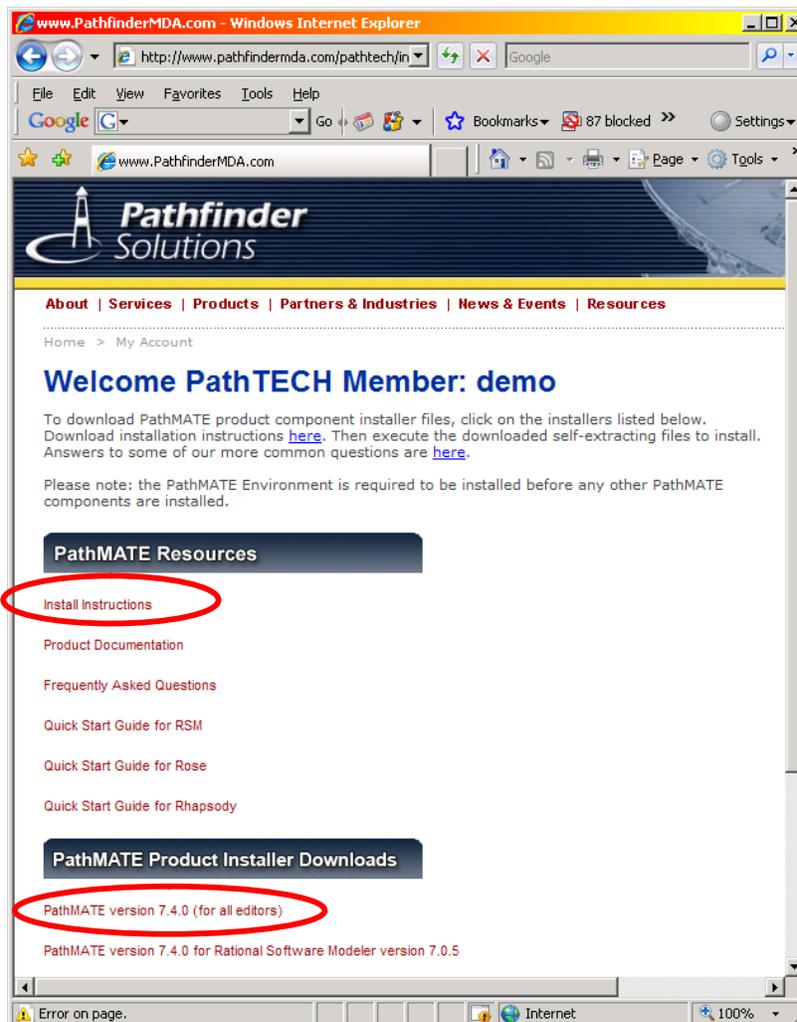
No other MDD transformation environment offers a more open or configurable set of development tools, designed to meet the requirements of embedded systems engineers.

Download and Install PathMATE

1. Navigate to <http://www.pathfindermda.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:



3. Download the following files:
 - *PathMATE, version 8.0.0 (for all editors)*, for the correct operating system.
 - *PathMATE Transformation Map for Java, version 8.00.000 (if you will be generating Java implementation code.)*
 - *Installation Instructions* – Follow these instructions to install the product and secure a key file.

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 if you plan to use this environment.

Eclipse and Topcased

The *PathMATE version 8.0 (for all editors)* installation includes a free preconfigured distribution of Eclipse v3.4 with Topcased 2.3 already configured.

Create a Platform Independent Model for SimpleOven

Eclipse Workspaces

When you launch Eclipse, 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.

To work with PathMATE, activate the Eclipse “Topcased” Perspective. This is done in the Eclipse Window Menu under Open Perspective, select other. When the dialog opens find and choose Topcased Modeling Perspective.

As with most leading software products, there are generally many ways in which a given action can be performed. We present a single, simple, and consistent way for maximum clarity. As you gain experience and comfort, explore the menus and dialogues. Try out some of the alternatives and develop the work style most comfortable to you.

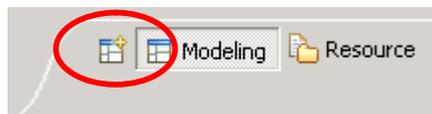
Task 1: Create a PathMATE UML Project Using the PathMATE Project Template

PROCEDURE: Use PathMATE New Project Wizard in Eclipse

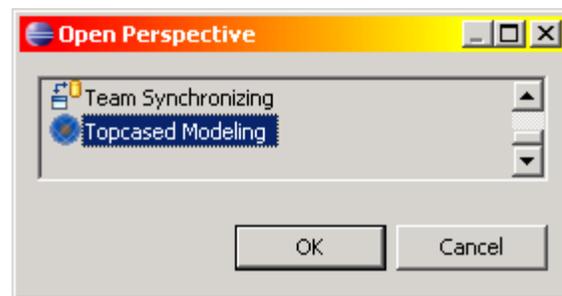
1. Launch Eclipse from the **Windows Start→All Programs** menu with **Pathfinder Solutions → Eclipse**:



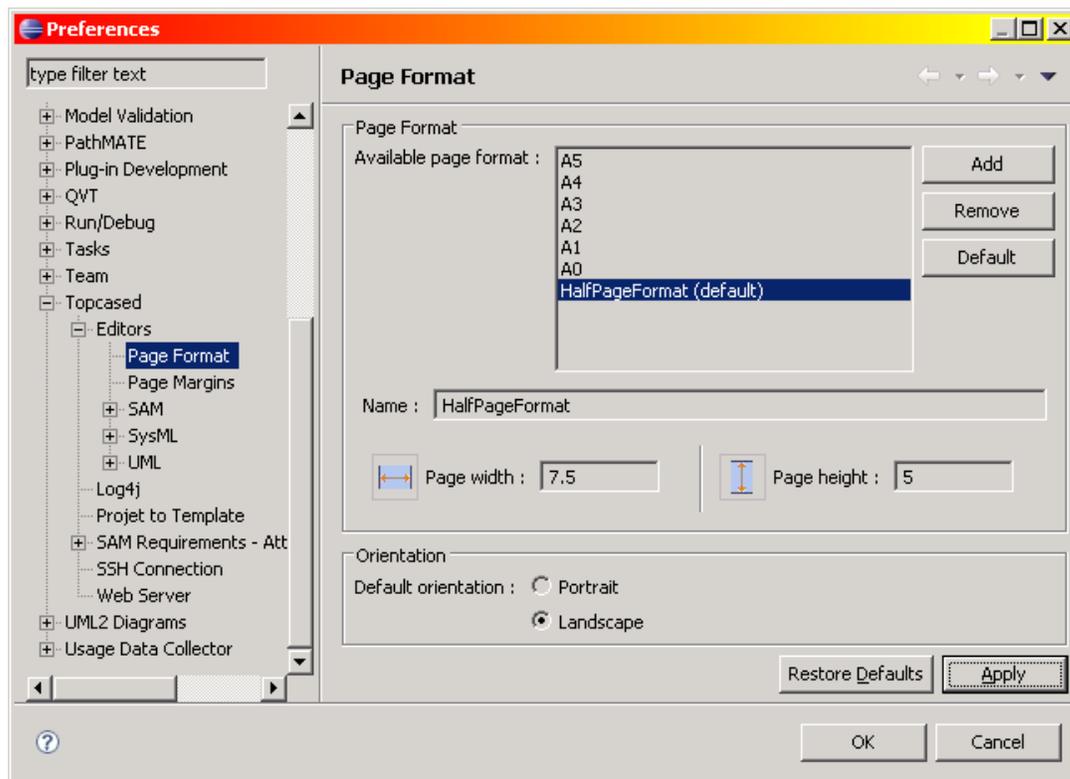
2. If it is not already selected, select the *Topcased Modeling* perspective: click on the Open Perspective button ...



3. ...click *Other* to get the complete set of perspectives, and select the *Topcased Modeling* perspective:

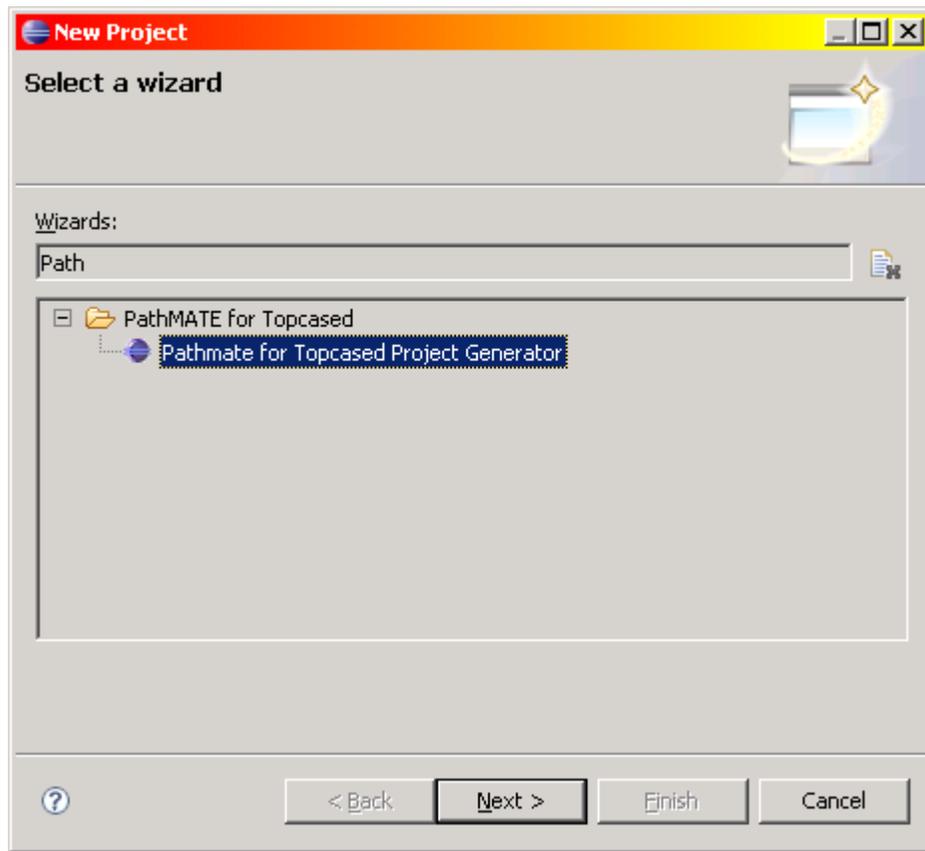


4. Click **OK**.
5. In Eclipse, select **Window** → **Preferences** and navigate to **Topcased** → **Editor**. Note what the default unit is (and if options are available change to unit of choice).
6. Now Navigate to **Topcased** → **Editors** → **Page Format**.
7. Click **Add** and name your new format *HalfPageFormat*. Select Landscape Default Orientation, enter 800 for the Page width, and 594 for the Page height. Select *HalfPageFormat* and press **Default**.

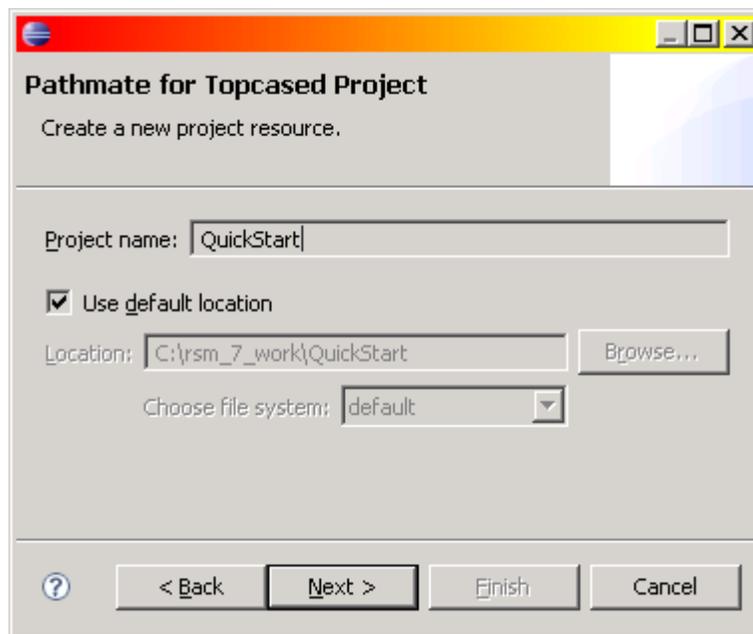


8. Click **Apply** and **OK**.

9. In Eclipse, select **File** → **New**, pick the *PathMATE for Topcased Project Generator* wizard and click **Next**:

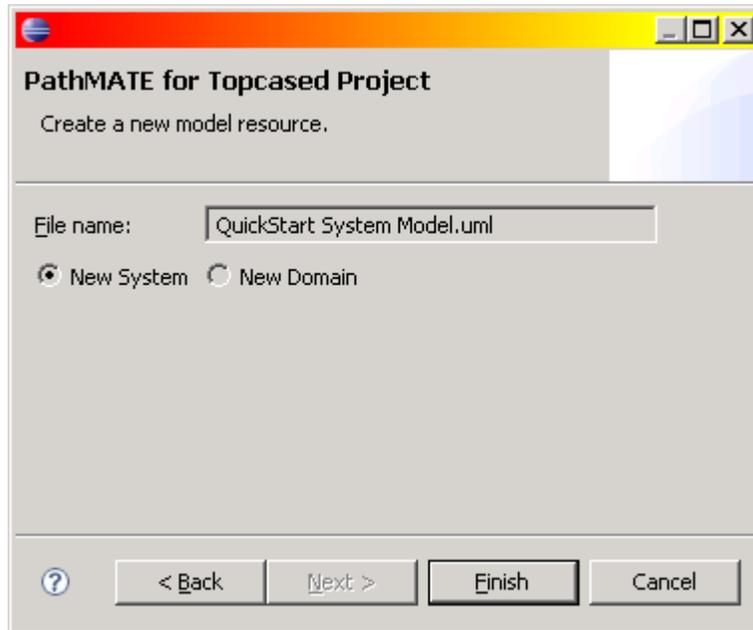


The UML Modeling Project wizard opens:



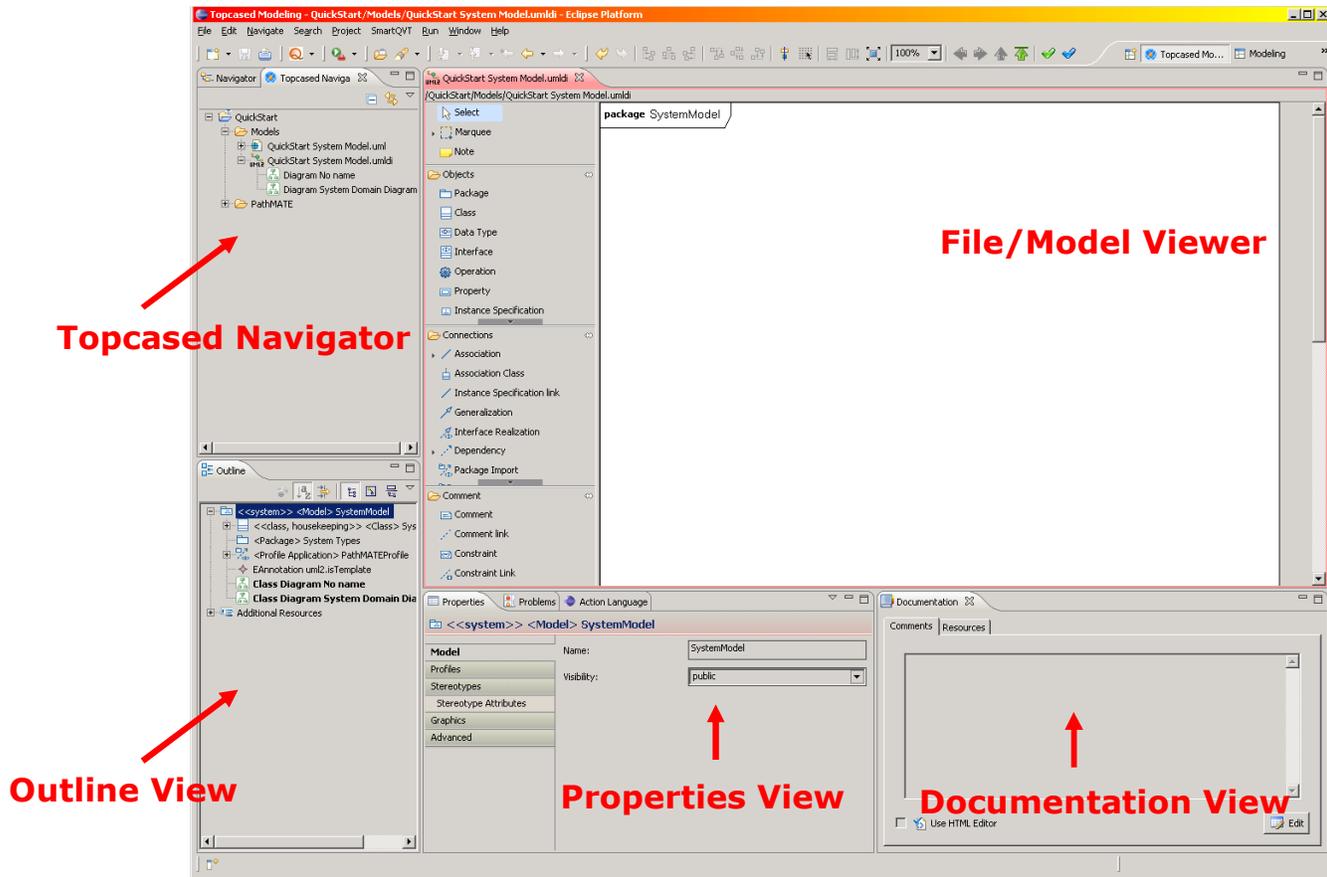
10. Type *QuickStart* in the Project name field and click **Next**.

The Create Model dialog opens:



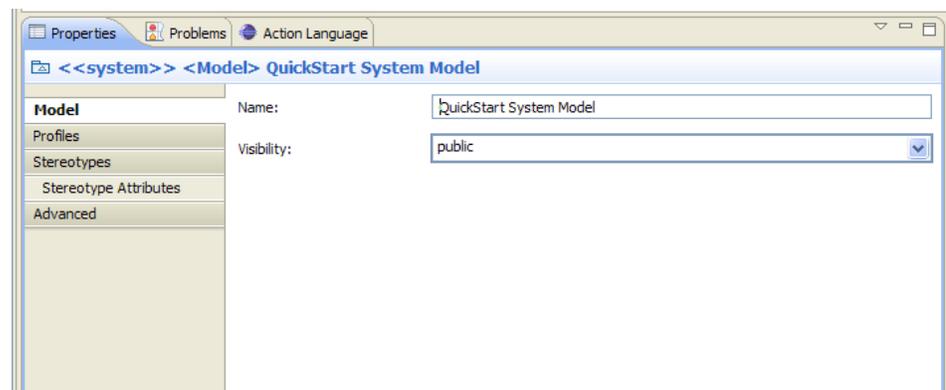
11. Select the **New System** button, in the File name field replace "new" with "QuickStart System Model", and click **Finish**.

The QuickStart Project appears in the Topcased Navigator, and the contents of the Quick Start System Model are shown in the Outline View:

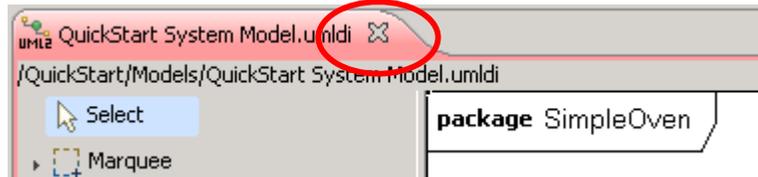


PROCEDURE: Use the Properties view to rename the System Model

1. If the QuickStart System Model.umldi file is not open, open it.
2. In the Outline View, select the Model **QuickStart System Model** – the Model now appears in the Properties view:



3. Using the Name field rename *Quick Start SystemModel* to *SimpleOven*.
4. Hit **Control-S** to save your model changes.
5. Click the "X" in the Editor Pane tab to close the *QuickStart System Model.umldi* file.

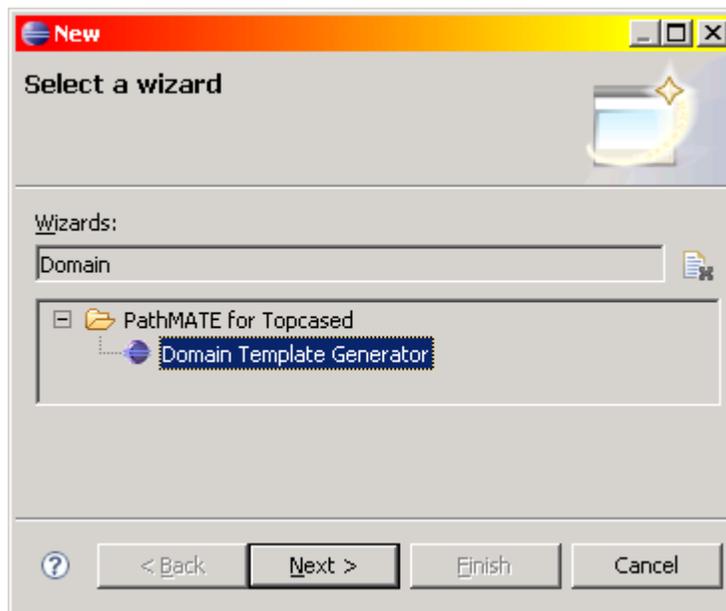


NOTE – Generally the .umldi (UML Diagram Interchange) file is the only modeling file we'll need to edit. Topcased has an important limitation in that only a single .umldi file can be open at one time if it references a common resource. We're closing *QuickStart System Model.umldi* so we can create (and open) a new umldi file for the MicrowaveCooking domain model.

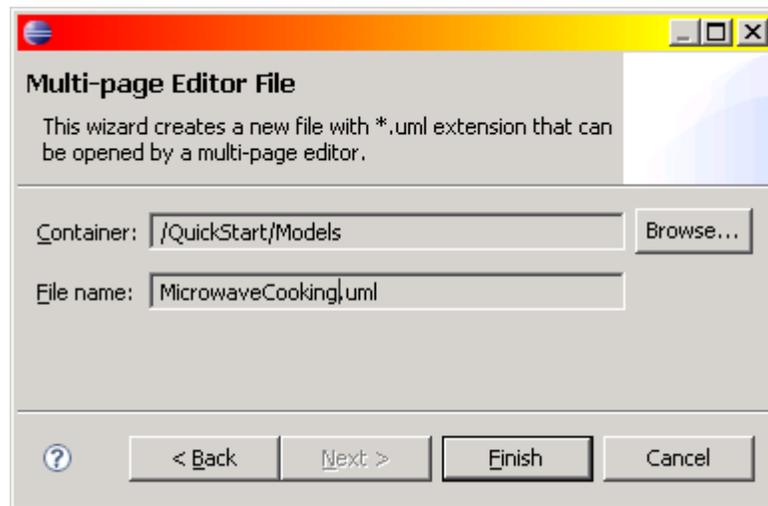
Task 2: Add the Modeled Domain *MicrowaveCooking*

PROCEDURE: Use the *PathMATE Domain Template Generator* to Add the *MicrowaveCooking* Domain

1. In the Topcased Navigator view, select the *Models* folder in the *QuickStart* project.
2. Select **File** → **New** → **Other** in the top menu bar. In the New File Dialog Wizards field type *Domain*:



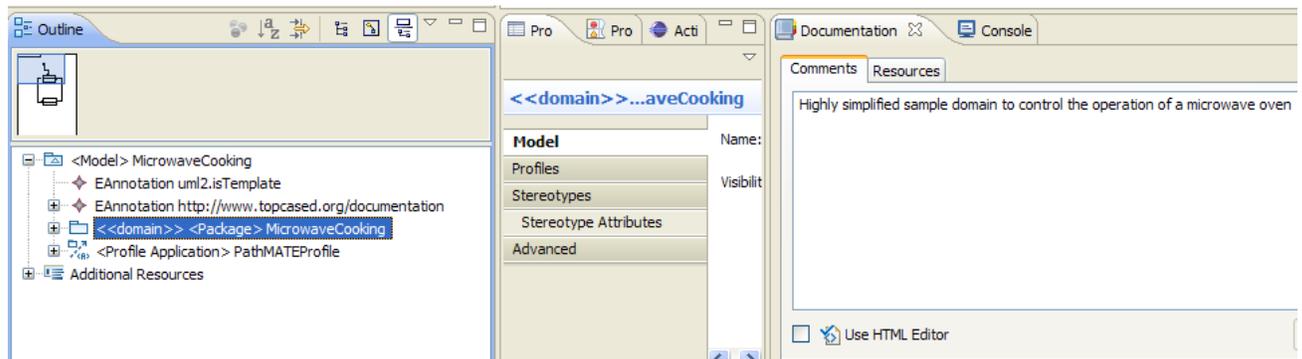
3. Under *PathMATE for Topcased* select **Domain Template Generator** and click **Next**. The New PathMATE Domain dialog opens:



4. Ensure the Container field shows */QuickStart/Models*, and in the File name field enter *MicrowaveCooking.uml*. Click **Finish**.

The new domain model appears in the Project Explorer.

5. In the Outline View select the domain *MicrowaveCooking*, and in the Documentation View enter *Highly simplified sample domain to control the operation of a microwave oven*.



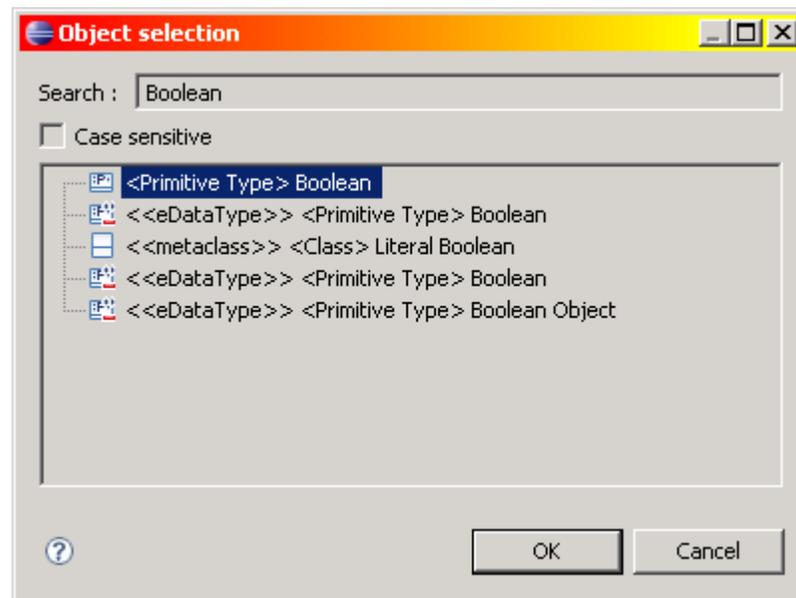
6. Click outside the document view. Note, in eclipse changes in a view aren't registered until it is no longer focused on.
7. Hit **Control-S** to save your model changes.

Expand *<Package> MicrowaveCooking* in the Outline View to see:

- A Domain Types package for user defined types within the domain.
- A Domain class diagram. The class diagram is open in your edit pane. See *Create a Class Diagram* for more information about the domain class diagram.
- A Domain Support class that contains the domain initialization() operation.
- A Domain Interface class for domain services. The class is a published interface.

PROCEDURE: Add a Domain Service to the MicrowaveCooking Domain

1. In the Outline View select the Domain *MicrowaveCooking* and expand it if necessary.
2. In the Outline View select the Interface *Domain Interface* and in the Propertive View use the Name field to rename it *MC services*. **NOTE** – *this will be helpful when building Scenario Models in later steps.*
3. Right-click on *MC services* and select **Create child → Owned Operation → Operation**.
4. Select the newly created Operation, and in the Properties View name it *ReportDoorStatus*.
5. In the Documentation View enter *Used by an external source (the door contact switch) to indicate whenever the door opens or closes.*
6. In the Properties View Parameters Tab click **Add**.
7. Go to the Details of the Parameter section, and in the Name field enter *is_open*.
8. At the right edge of the Type field select the **type browser button**, and enter *Boolean* in the Search field:

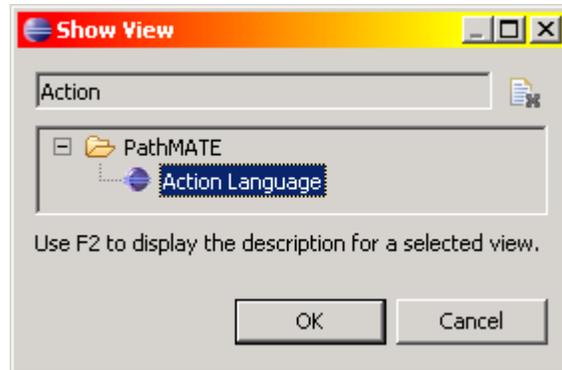


9. Click **OK**.
10. In the Outline View, under the Operation *ReportDoorStatus*, select the parameter *is_open*. In the Documentation View enter *TRUE indicates the door has opened, and FALSE indicates it has closed.*
11. Hit **Control-S** to save your model changes.

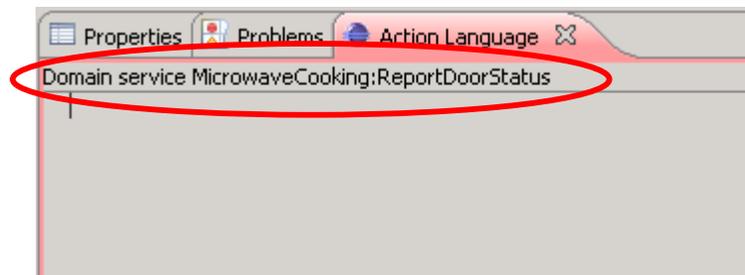
PROCEDURE: Specify ReportDoorStatus Interface Operation Action Language

To complete the *ReportDoorStatus* service, the PathMATE Action Language view is used to specify the platform-independent action language for the service.

1. If you do not have an Action Language View window open (typically grouped with the Properties View), at the Eclipse top-level menu select **Window** → **Show View** → **Other**.



2. Select the *Action Language* view and click **OK**.
3. In the Outline View, select the Operation *ReportDoorStatus*. Ensure the context specification indicates the Domain service *MicrowaveCooking:ReportDoorStatus* is in focus:



4. Copy the following PAL statements into the Action Language View:



```

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

```

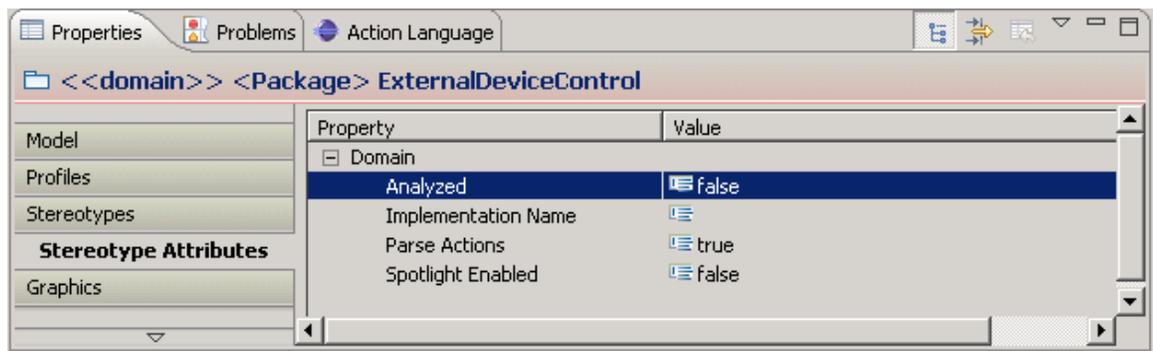
5. Hit **Control-S** to save your model changes.
6. Click the "X" in the Editor Pane tab to close the *MicrowaveCooking.uml* file.

Task 3: Create a Realized Domain

ExternalDeviceControl is a realized domain that specifies the model-level interface for some hand-written code that interfaces with external device hardware. This interface includes a published set of services specified via an Interface Class, and a user-defined type (enumeration) that supports it.

PROCEDURE: Use the PathMATE Domain Template Generator to Add the ExternalDeviceControl Domain

1. In the Topcased Navigator view, select the *Models* folder in the *QuickStart* project.
2. Select **File** → **New** → **Other** in the top menu bar.
3. Under *PathMATE for Topcased* select **Domain Template Generator** and click **Next**.
4. Ensure the Container field shows */QuickStart/Models*, and in the File name field enter *ExternalDeviceControl.uml*. Click **Finish**.
5. In the Outline View select the domain *ExternalDeviceControl*, and in the Documentation View enter *This domain provides a control interface to external devices (like a light)*.
6. Click on the **Stereotype Attributes** tab.
7. Expand the *Domain* properties group.
8. Select the *Analyzed* property from the Domain group and change its value from **true** to **false**:



9. Hit **Control-S** to save your model changes.

PROCEDURE: Add a UML Enumeration to ExternalDeviceControl's Public Types

1. In the Outline View expand the Package *ExternalDeviceControl*, right-click the Package *Domain Types* and select **Create Child** → **Owned Type** → **Enumeration**.
2. Expand the package *Domain Types*, select the newly created Enumeration, and in the Properties View Model Tab name the enumeration *sys_device_e*.
3. In the Documentation View enter *External device identifiers*.
4. Right-click the new Enumeration and select **Create Child** → **Owned Literal** → **Enumeration Literal**.
5. Expand the enumeration, select the newly created Literal and using the Properties View name it *SYS_DEVICE_LIGHT*.
6. Hit **Control-S** to save your model changes.

PROCEDURE: Add a Domain Service to the ExternalDeviceControl Domain

1. In the Outline View select the Interface *Domain Interface* and in the Properties View use the Name field to rename it *EDC services*. **NOTE** – *this will be helpful when building Scenario Models in later steps.*
2. In the Outline View select the Interface *EDC services*, right-click and select **Create child** → **Owned Operation** → **Operation**.
3. Select the newly created Operation, and in the Properties View name it *ActivateDevice*.
4. In the Documentation View enter *Activate the specified device*.
5. In the Properties View Parameters Tab click **Add**.
6. In the Name field enter *device_id*.
7. At the right edge of the Type field select the **type browser button**, and enter *sys_device_e* in the Search field:
8. Click **OK**.
9. In the Outline View, under the Operation *ActivateDevice*, select the parameter *device_id*. In the Documentation View enter *Specifies the id of the target device*.
10. Repeat steps 1-8 to add another Domain Service *DeactivateDevice* with a similar description and parameter.
11. Hit **Control-S** to save your model changes.
12. Click the "X" in the Editor Pane tab to close the *ExternalDeviceControl.uml* file.

Task 4: Complete the System Domain Chart

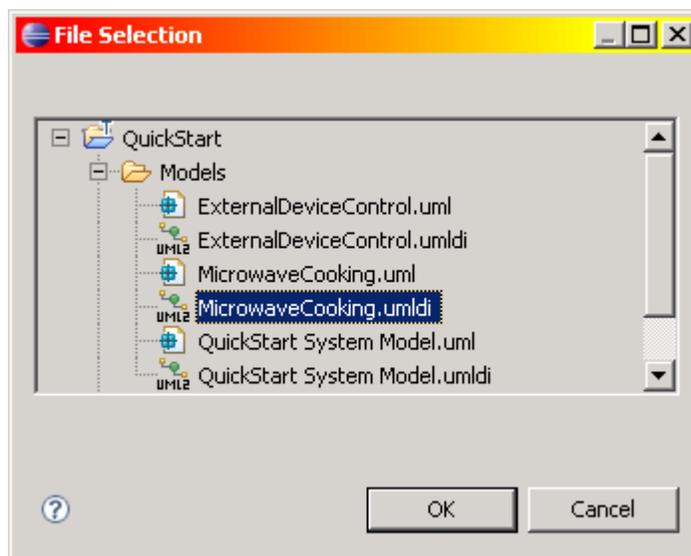
In this task you will specify the Logical Architecture of the system in a Domain Chart.

PROCEDURE: Use System Domain Chart to Specify the Domain Hierarchy and their Dependencies

1. In the Topcased Navigator View double-click on the **QuickStart System Model.uml** file to open it.
2. In the Outline View expand the system.

The diagram open in the Editor Pane is the System Domain Chart Diagram.

3. In the Outline View right click on *Additional Resources*, select **Load Resource** and click the **Browse Workspace** button.

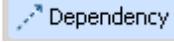


4. Under *QuickStart/Models* select *MicrowaveCooking.uml* and click **OK** and **OK**.
5. Expand *Additional Resources*, *MicrowaveCooking.uml*, *Diagrams*, *Model MicrowaveCooking*, and drag the package *MicrowaveCooking* onto the Domain Diagram, near the top.
6. On the Domain Diagram select *MicrowaveCooking* and drag the Change Bounds box to resize the package symbol so only the name portion shows.

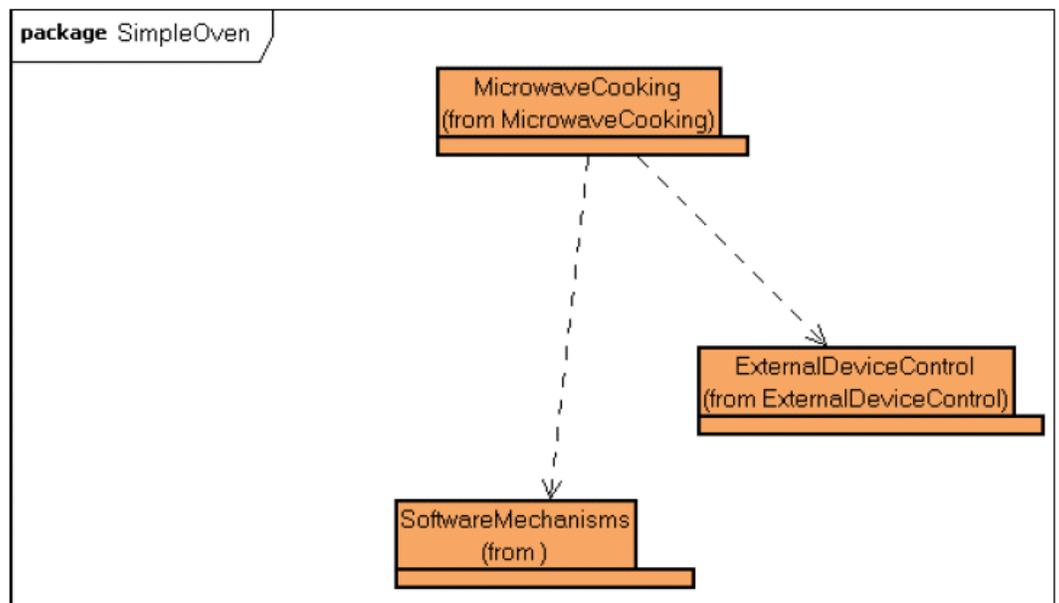


7. Right click on any diagram elements that do not belong and select *Delete from Diagram*. Press yes to the following prompt.
8. Repeat steps 3-7 to load *ExternalDeviceControl.uml* from the *QuickStart/Models* folder and add its domain Package Symbol

to the Domain Diagram, below and to the right of *MicrowaveCooking*.

9. Repeat steps 3-7 to load *SoftwareMechanisms.uml* from the QuickStart/PathMATE folder, and add its domain Package Symbol to the Domain Diagram, below and to the left of *ExternalDeviceControl*.
10. In the Edit Pane tool palette select the **Dependency tool** . Click in the *MicrowaveCooking* Package, and then in the *ExternalDeviceControl* Package.
11. Repeat step 9 to create a dependency between *MicrowaveCooking* and *SoftwareMechanisms*.
12. Click on any white space in the Diagram to bring up the diagram properties in the Properties View.
13. Set the Page Format, to *HalfPageFormat*.

Your Domain Diagram may look something like this:



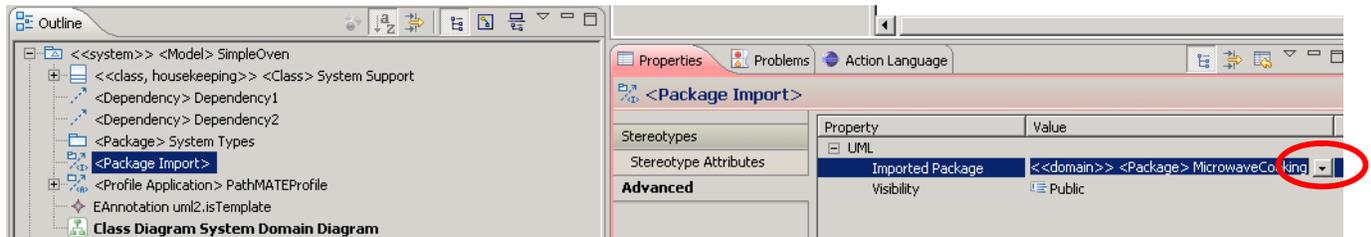
14. Hit **Control-S** to save your model changes.

PROCEDURE: Specify System Package Imports

In addition to specifying the Domain hierarchy, the file structure of the system also needs to be identified. Specifically all domain model files used in the system need to be imported.

1. In the Outline View right click the Model *SimpleOven* and select **Create Child → Package Import → Package Import**.
2. Select the newly created Package Import.
3. In the Properties View select the Advanced Tab.

- On the right edge of the Imported Package field click the package selector button:



- Choose the domain Package *MicrowaveCooking*.
- Repeat steps 1-5 to add Package Imports for *ExternalDeviceControl* and *SoftwareMechanisms*.
- Hit **Control-S** to save your model changes.

PROCEDURE: Capture a System-Level Scenario Model

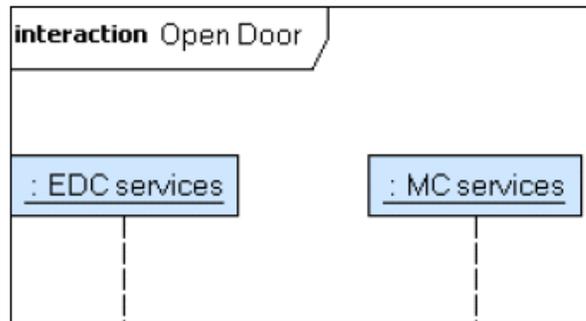
Although a typical system constructed with PathMATE will have many System-Level Scenarios, we will capture a single scenario to expose the core concepts.

- In the Outline View right click the Model *SimpleOven* and select **Create child** → **Nested Package** → **Package**. In the Properties View Model tab, name the new package *System-Level Scenarios*.
- Right click on the package *System-Level Scenarios* and select **Add Diagram** → **Sequence Diagram**.
- In the Properties View select the new Collaboration and use the Model Tab Name field to name it *Open Door*.
- In the Properties View select the new Interaction and use the Model Tab Name field to name it *Open Door* also.
- In the Documentation View add:

PRECONDITION - The system is running, and the door is closed.

POSTCONDITION - The light has been turned on.
- Select the new Sequence Diagram. In the Properties view on the Advanced tab, use the Name field to name the diagram *Open Door*.
- In the Editor palette pick the **Lifeline tool**  Lifeline .
- Click in the upper left of the Editor Pane. Delete the default name and hit **Enter** (this creates a new lifeline with no name.)
- In the Properties View click the **Object selector** (...) to the right of the Represents field. Type *EDC services*. The Interface *EDC services* should be selected. Click **OK**.
- Repeat steps 7-9 to create a lifeline for *MC services* just to the right of the *EDC services* lifeline.

At this point your sequence diagram may look something like this:

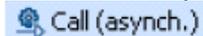


11. In the Editor palette pick the **Execution extent tool**



12. Click on the *EDC services* lifeline just below the title box.

13. In the Editor palette pick the **Call asynchronous tool**



14. Click once in the Execution extent box on the *EDC services* lifeline, and again directly across from this on the *MC services* lifeline.

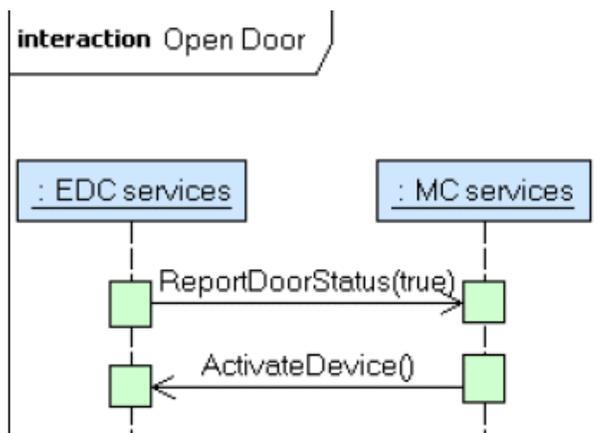
15. Select the new Message line on the diagram, and in the Properties View on the Model tab, click the **Object selector** (...) to the right of the Operation field.

16. Pick *ReportDoorStatus* and click OK.

17. In the Properties View, in the Value column for the *is_open* attribute, select **true**. Click **OK**.

18. Repeat steps 11-15 to place a call from *MC services* to the *EDC services ActivateDevice* operation.

At this point your sequence diagram may look something like this:



19. Hit **Control-S** to save your model changes.

20. Click the "X" in the Editor Pane tab to close the QuickStart System Model.umlIdi file.

Task 5: Complete the Class Diagram for the MicrowaveCooking Domain

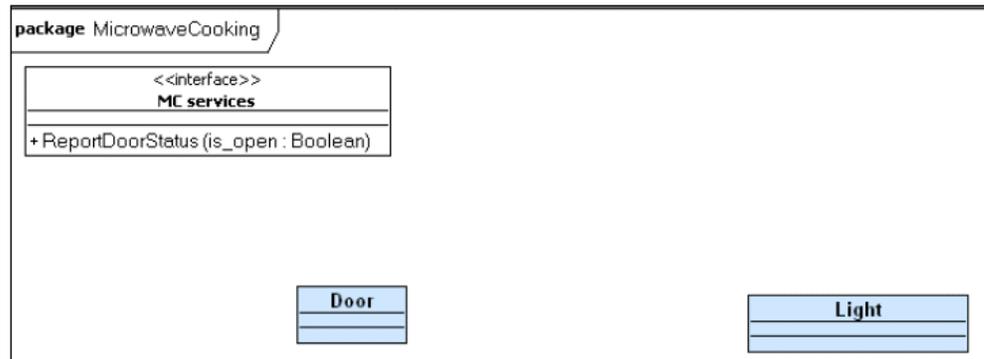
The MicrowaveCooking Domain is the heart of the application. In this task you will complete the MicrowaveCooking's Domain Class Diagram by adding Classes, Attributes, and Associations.

PROCEDURE: Create the Oven, Door and Light Classes in the MicrowaveCooking Domain

Topcased supports the creation of model element both from the Editor Pane and the Outline View. We will do both.

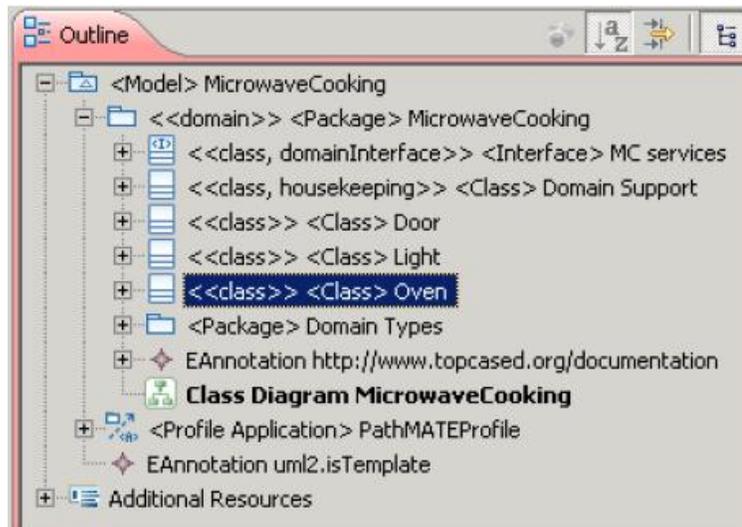
1. In the Topcased Navigator View double-click on the **MicrowaveCooking.uml** file to open it. The Domain Class Diagram opens in the Editor Pane.
2. In the Outline View expand the Package *MicrowaveCooking*. Select the diagram *Class Diagram Main*.
3. In the Properties view on the Advanced tab, use the Name field to name the diagram *MicrowaveCooking*.
4. In the Editor pane, right-click on Domain and select *Delete from Diagram*.
5. Click on any white space in the Diagram to bring up the diagram properties in the Properties View.
6. Set the Page Format, to *HalfPageFormat*.
7. In the Editor Pane palette, select the **Class tool** .
8. Click on the left side of the diagram. At this point the newly create class symbol appears, and the default name is selected. Type *Door* and hit **Enter**.
9. In the Outline View select the new *Door* class. In the documentation view enter the description *Microwave oven door to cooking area*.
10. Repeat steps 5-7 to create class named *Light* to the right of *Door* with the description *Light to illuminate cooking area*.
11. In the Outline View right-click on the Package *MicrowaveCooking* and select **Create child** → **Owned Type** → **Class**.
12. Select the newly created class. In the Properties View on the Model tab use the Name field to name it *Oven*.
13. In the documentation view enter the description *Microwave oven - central controller*.

At this point your class diagram may look something like this:



Note: If the Domain Interfaces do not appear in the diagram, they can be dragged and dropped from the outline view unto the MC services box to be added to diagram.

Your outline view may look like:

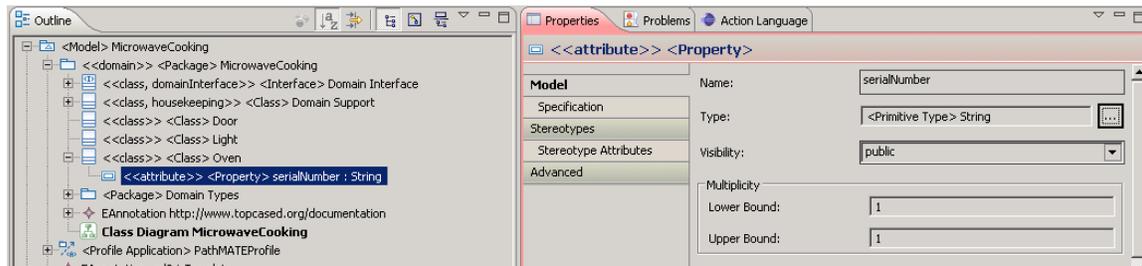


21. Hit **Control-S** to save your model changes.

PROCEDURE: Add Attributes to the Oven and Light Classes

1. In the Outline View right-click the *Oven* class and select **Create child** → **Owned Attribute** → **Property**.
2. Expand the *Oven* class, select the new attribute, and in the Properties view use the Name field to name it *serialNumber*.
3. Select the Type browser and enter *String*.

The Properties view for the attribute may look like:



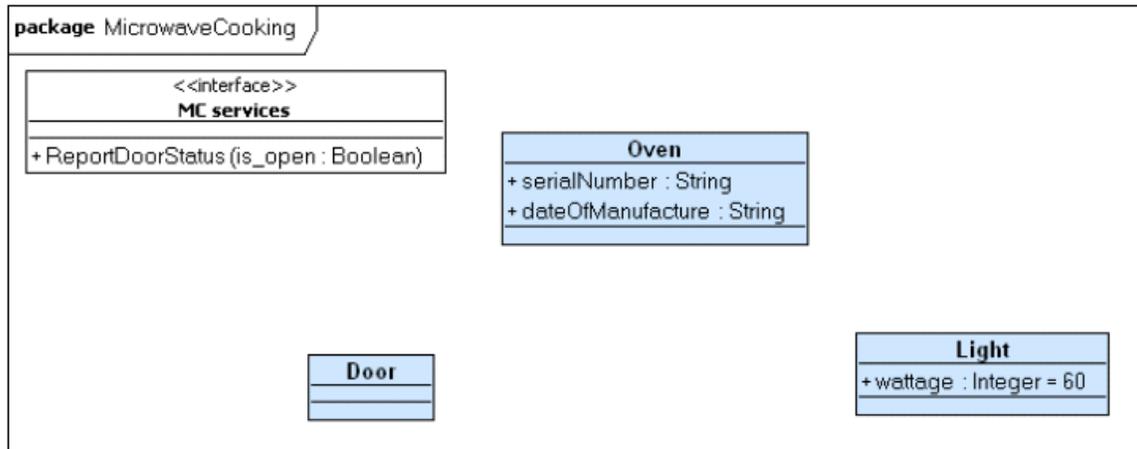
4. In the Documentation View enter *Serial number for the overall oven unit.*
5. Repeat steps 1-4 to add to the class *Oven* a new attribute *dateOfManufacture* of type *String* with the description *The date the oven unit was assembled, and ready to package and ship.*
6. Repeat steps 1-4 to add to the class *Light* a new attribute *wattage* of type *Integer* with the description *The wattage of the bulb in the light fixture.*
7. In the Outline View, right-click the attribute *wattage* and select **Create child** → **Default Value** → **Literal Integer**.
8. In the Properties View Advanced Tab type *60* in the Value field.

Your Outline view for the *Light* class may look like:



9. Drag the Class *Oven* from the Outline View onto the top center of the Class Diagram.
10. Drag the *Light* attribute *wattage* from the Outline View into the center compartment of the *Light* class symbol on the Class Diagram.
11. Right-click the *Light* class symbol on the Class Diagram and select **Graphical Properties** → **Auto Resize**.

At this point your class diagram may look like:

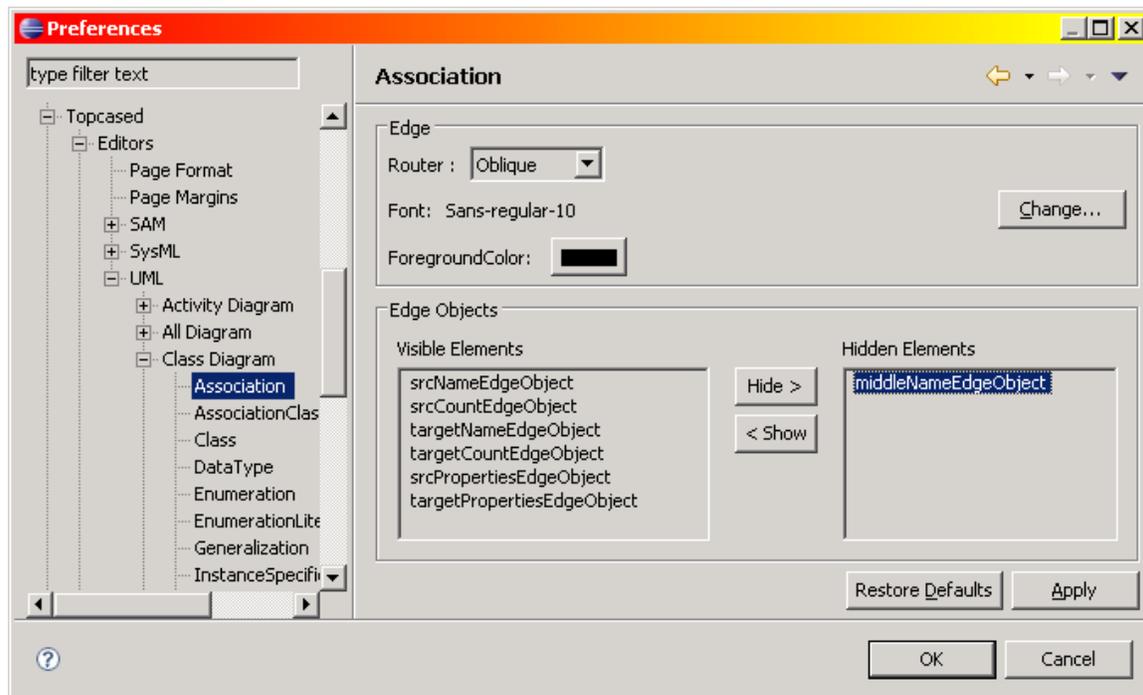


22. Hit **Control-S** to save your model changes.

PROCEDURE: Associate the Classes

First we will adjust the display preferences for associations, then we will create two associations in the MicrowaveCooking domain.

1. At the Eclipse top-level menu select **Window** → **Preferences** and navigate to **Topcased** → **Editors** → **UML** → **Class Diagram** → **Association**.
2. Select **middleNameEdgeObject** in the Hidden Elements box:



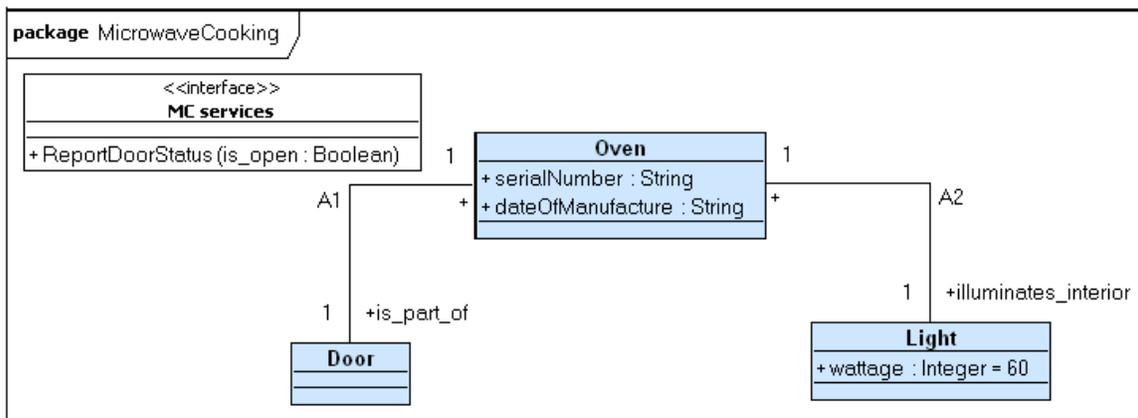
3. Click **Show** and **OK**.

4. In the Editor Pane palette select the **Association tool**
 - Association
5. Click once in the *Oven* class symbol, and then again in the *Door* class symbol.
6. In the Outline View select the new association. In the Properties View on the Model tab, use the Name field to name it *A1*.
7. In the Documentation View enter *The oven has a door to seal the cooking area during microwave emission*.
8. On the First End tab, delete the text in the Role field.
9. On the Second End tab, enter *is_part_of* in the Role field.
10. In the Editor Pane right-click the association and select **Graphical Properties → Change Router:**



11. Select Rectilinear and click **OK**.
12. Drag the line and label elements of the association to arrange them in a clear manner.
13. Repeat steps 4-12 to create association *A2* between *Oven* and *Light* with a role phrase *illuminates_interior* on the *Light* end, and a description *The light illuminated the oven cooking area*.

At this point your class diagram may look like:



23. Hit **Control-S** to save your model changes.

Task 6: Complete the MicrowaveCooking Domain Initialization Action

PROCEDURE: Add MicrowaveCooking Initialization Action

The domain initialization action creates startup-time class instances in the MicrowaveCooking domain.

1. In the Outline View select the housekeeping class *Domain Support*.
2. Ensure the context is shown as *MicrowaveCooking domain initialization* and click the Action Language View tab.
3. Copy the following PAL statements into the Action Language View:



```
// 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);
```

4. Hit **Control-S** to save your model changes.

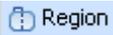
Task 7: Create the Door State Machine

The door State Machine shows transitions between closed and open states.

PROCEDURE: Create and Name the Door State Machine Diagram

1. In the Outline View right-click the *Door* class and select **Create child** → **Owned Behavior** → **State Machine**.
2. Expand the *Door* class and right-click the newly created State Machine and select **Add Diagram** → **State Machine Diagram**. A new state machine diagram opens in the Editor Pane.
3. In the Outline View select the new State Machine. In the Properties View Model tab enter in the Name field *Door State Machine*.

PROCEDURE: Lay out Door States

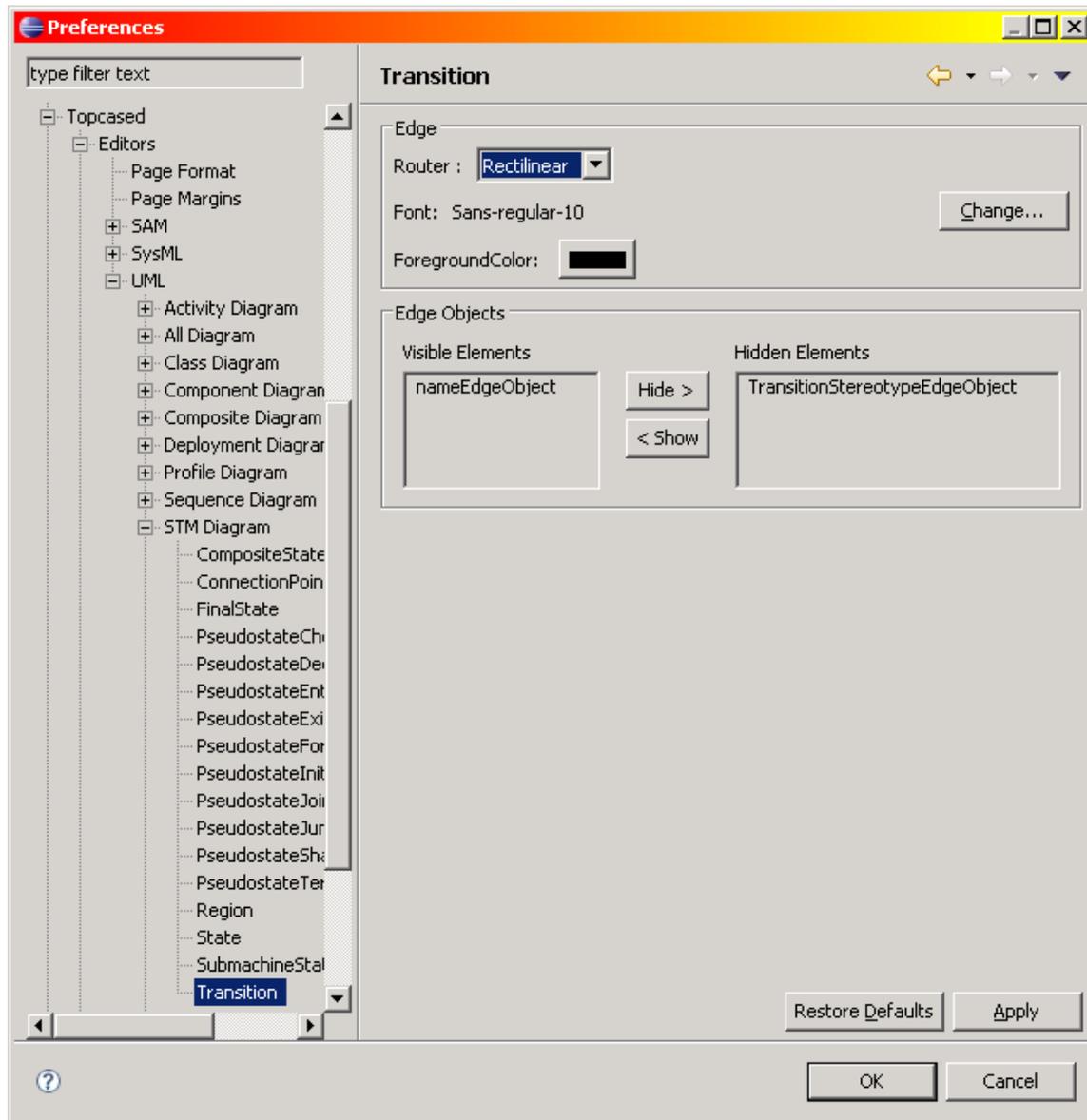
1. In the Editor Pane palette, select the **Region tool** .
2. Click on the Editor Pane to place a region. (It is a box that lines up with the original state machine diagram borders so you really don't see anything new.)
3. In the Editor Pane palette, select the **Initial pseudostate tool** .

4. Click in the top area of the diagram to place the Initial pseudostate. Type *initial* to name it.
5. In the Editor Pane palette, select the **State tool**  .
6. Click just below the Initial pseudostate to place the State. Type *Closed* to name it.
7. Repeat steps 5-6 to create a state *Open* just below the *Closed* state.

PROCEDURE: Change State Machine Diagram Transition Preferences

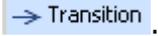
1. At the Eclipse top-level menu select **Window** → **Preferences** and navigate to **Topcased** → **Editors** → **UML** → **STM Diagram** → **Transition**.

2. Select the Router choice **Rectilinear**:



3. Click **OK**.

PROCEDURE: Draw an Untriggered Transition

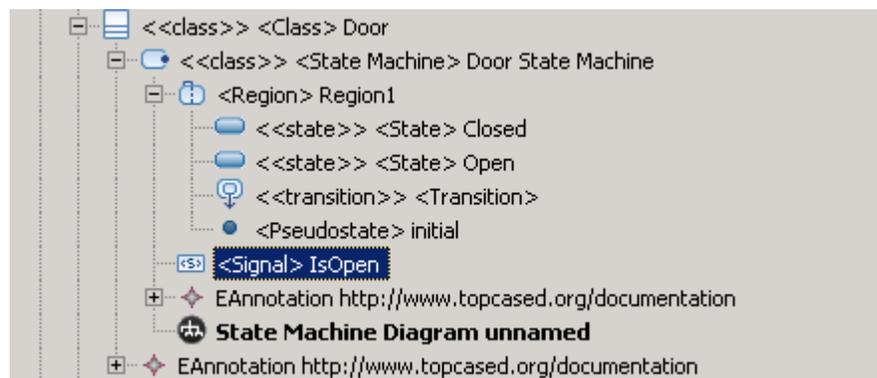
1. In the Editor Pane palette, select the **Transition tool** .
2. Click once in the *initial* pseudostate, and again in the *Closed* state.
3. In the Outline View select the new transition. In the Properties View on the Model tab, delete the name (typically *Transition1*) from the Name field.

PROCEDURE: Create Signals, Triggers and Draw a Triggered Transition

Transitions can be triggered by Signals which can be defined with a State Machine. However in Topcased a Signal must be tied to a Trigger with a Signal Event, which can only be defined under a package. So we will use a separate package *Signal Events* for these event elements.

1. In the Outline View right-click on Package *MicrowaveCooking* and select **Create child** → **Nested Package** → **Package**. Name the new Package *Signal Events*.
2. In the Outline View right-click on the State Machine *Door State Machine* and select **Create child** → **Nested Classifier** → **Signal**.
3. Select the new Signal and name it *IsOpen*. In the Documentation View enter *Indicates the Door has opened*.

Your Outline View may look like:



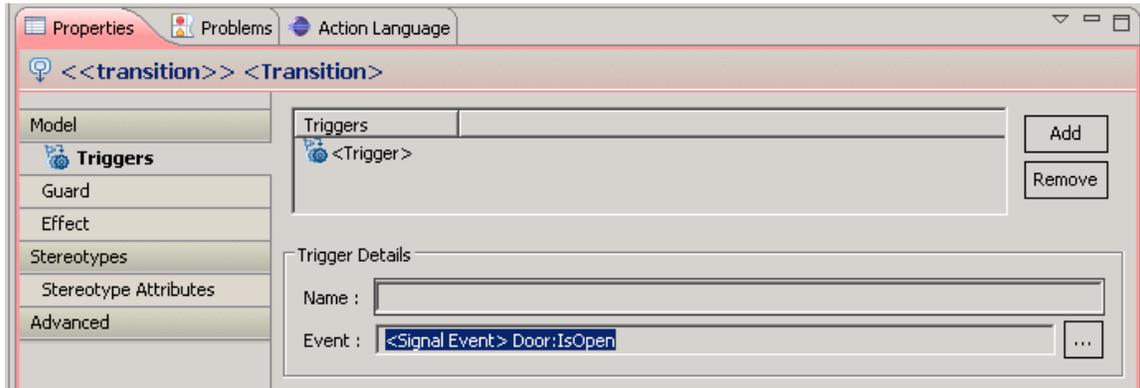
4. In the Outline View right-click on Package *Signal Events* and select **Create child** → **Packaged Element** → **Signal Event** (you may need to scroll down the list to find it.). Name it *Door:IsOpen*.
5. In the Advanced Tab Signal field select the **IsOpen** signal.

Your Outline View may look like:

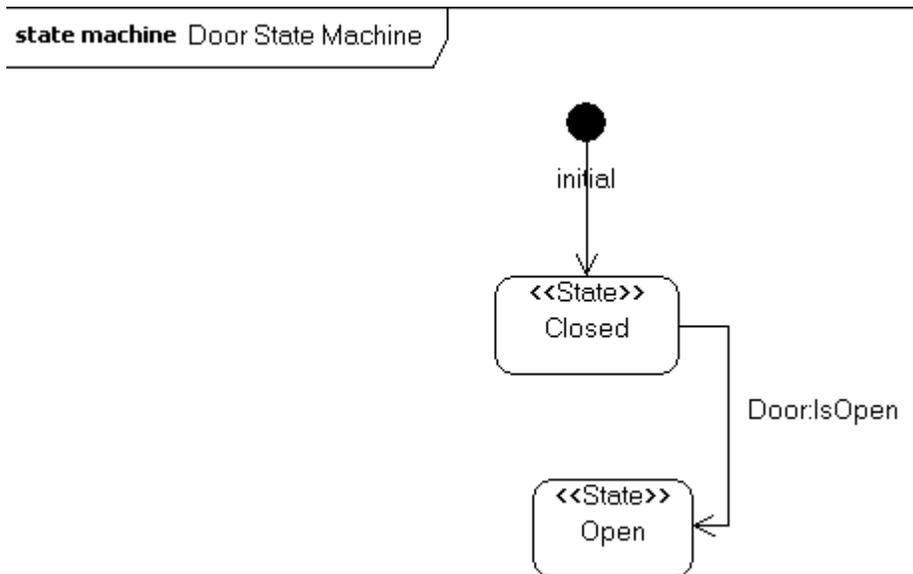


6. In the Editor Pane palette, select the **Transition tool** → **Transition**.
7. Click once in the *Closed* state, and again in the *Open* state.
8. In the Outline View select the new Transition.

9. In the Properties View Model tab, delete the name from the Name field.
10. On the Triggers tab, click **Add**. Select the new Trigger in the Triggers list.
11. Click the Event selector, and select the Signal Event *Door:IsOpen*:



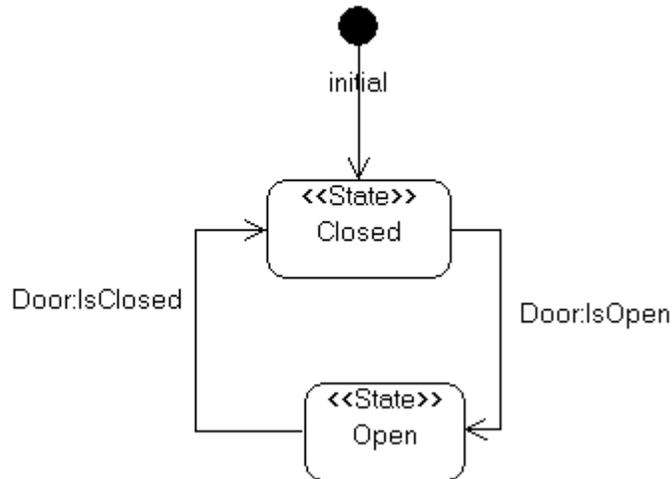
Your State Machine diagram may look something like:



12. Hit **Control-S** to save your model changes.
13. Repeat steps 2-11 to add a new Signal *IsClosed* with the description *Indicates the Door has closed*, a corresponding Signal Event. Have it trigger a transition from the state *Open* to the state *Closed*.

Now your State Machine diagram may look something like:

state machine Door State Machine



14. Hit **Control-S** to save your model changes.

PROCEDURE: Capture Door State Entry Actions

1. In the Outline View right-click on the *Door* State Machine *Closed* state and select **Create child** → **Entry** → **Activity**.
2. Select the new Activity and in the Properties View on the Model tab in the Name field enter the Action Summary *Generate TurnOff to the light*.
3. In the Action Language View copy the following PAL statements:



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

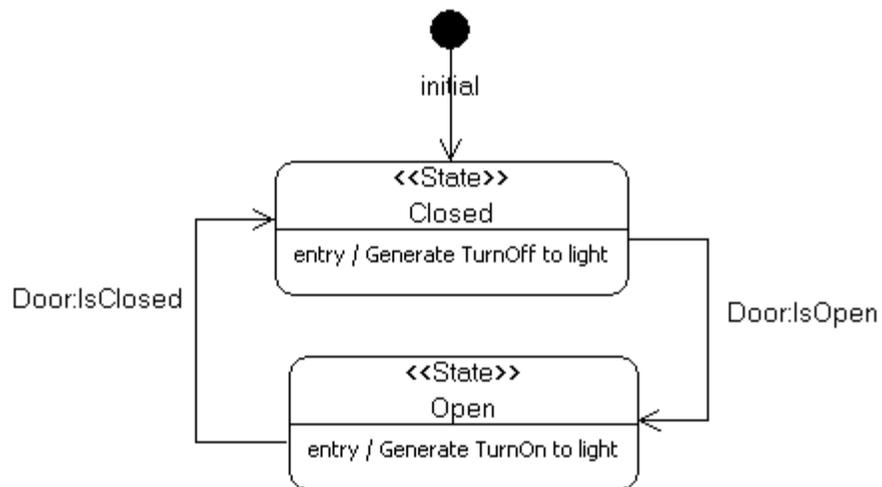
4. Right-click the *Closed* state symbol on the State Machine Diagram and select **Graphical Properties** → **Auto Resize**.
5. Repeat steps 1-4 for the *Open* state entry action, using the following PAL statements:



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

Now your State Machine diagram may look something like:

state machine Door State Machine



6. Hit **Control-S** to save your model changes.

Task 8: Create the Light State Machine

This section will briefly recap the steps needed to capture a state machine as you build the *Light*. If you are unsure exactly how to proceed, please review the detailed Procedures in Task 7 above.

PROCEDURE: Create the State Machine Diagram and States

1. In the Outline View right-click the *Light* class and select **Create child** → **Owned Behavior** → **State Machine**.
2. Right-click the State Machine and select **Add Diagram** → **State Machine Diagram**. Name it *Light state machine*.
3. Place a region with  **Region**.
4. Place an Initial pseudostate  and name it *initial*.
5. Place states *Off* and *On* with .

PROCEDURE: Draw Transitions

1. Create a transition from *initial* to *Off* with  **Transition**. Delete its name.
2. Create additional transitions from *Off* to *On*, and from *On* to *Off*.
3. Click once in the *initial* pseudostate, and again in the *Closed* state.

- Hit **Control-S** to save your model changes.

PROCEDURE: Create Signals, Triggers, and apply them

- In the Outline View right-click on the State Machine *Light State Machine* and select **Create child** → **Nested Classifier** → **Signal**. Name it *TurnOff*, and give it the description *Indicates the light is to be deactivated*.
- Repeat step 1 for the Signal *TurnOn* with the description *Indicates the light is to be activated*.
- Right-click on Package *Signal Events* and select **Create child** → **Packaged Element** → **Signal Event** twice. Name them *Light:TurnOn* and *Light:TurnOff*.
- In the Advanced Tab Signal field select the *TurnOff* and *TurnOn* Signals appropriately.
- On the Editor Pane select the Transition from *Off* to *On*. In the Properties View Model tab, delete the name from the Name field. On the Triggers tab, click **Add** and select *Light:TurnOn* in the Event selector.
- Repeat step 5 for the Transition from *On* to *Off*, selecting the Event *Light:TurnOff*.
- Hit **Control-S** to save your model changes.

PROCEDURE: Create State Entry Actions

- In the Outline View use **Create child** → **Entry** → **Activity** to add an Entry Action Activity to the *Off* state. Name it *Deactivate device with ExternalDeviceControl*. In the Action Language View copy the following PAL statement:



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

- Add an Entry Action Activity to the *On* state and name it *Activate device with ExternalDeviceControl*. Copy the following PAL statement:

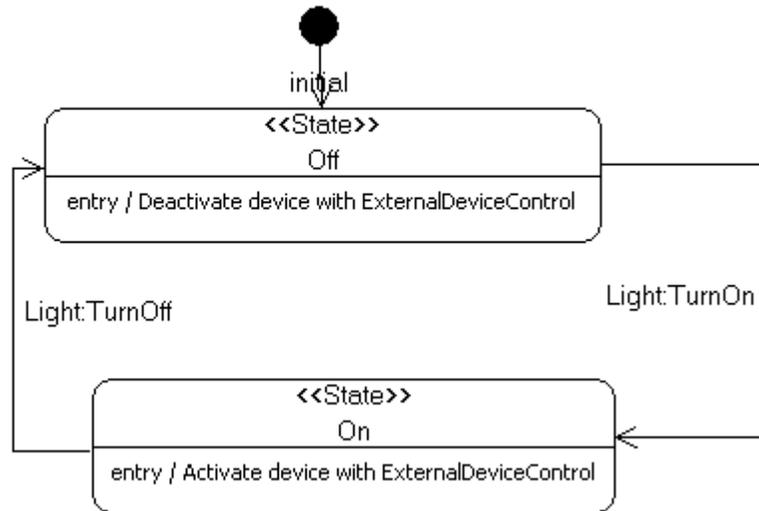


```
ExternalDeviceControl:ActivateDevice(SYS_DEVICE_LIGHT);
```

- On the State Machine Diagram do **Graphical Properties** → **Auto Resize** on both states.

Your *Light* State Machine diagram may look something like:

`state machine` Light state machine



4. Hit **Control-S** to save your model changes.
5. Close MicrowaveCooking.uml.di.

Congratulations! Your QuickStart project has a Real Model.

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:

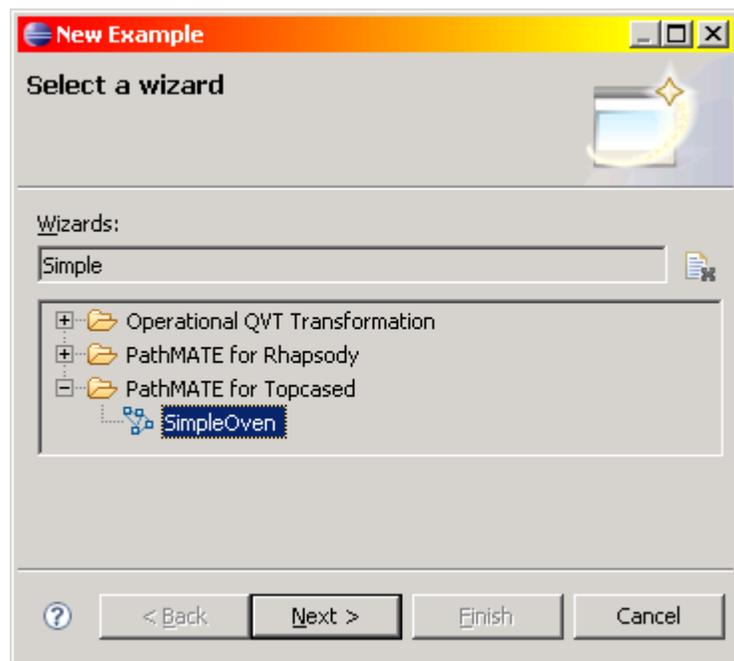
- Navigate to the complete sample SimpleOven project and use it as a reference, copy C++ and Java transformation properties and realized code from it.
- Add a PathMATE Project to the QuickStart project which will control transformation and deployment.
- "Smoke Test" your model by generating documentation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

Task 1: Reuse elements from Example Project

SimpleOven exists as a sample model in the PathMATE profile. We will use the existing SimpleOven project as a reference project to copy the realized ExternalDeviceControl implementation code and transformation markings/properties that guide the automatic generation of a C++ and Java implementation for this system.

PROCEDURE: Instantiate a Reference Project for SimpleOven

1. From the Eclipse base menu select **File** → **New** → **Example**.
2. In the Wizards field start to type *SimpleOven*.



3. Select **Next**.

4. Name the project *ReferenceSimpleOven* and click **Finish**.
5. Press **Finish**.

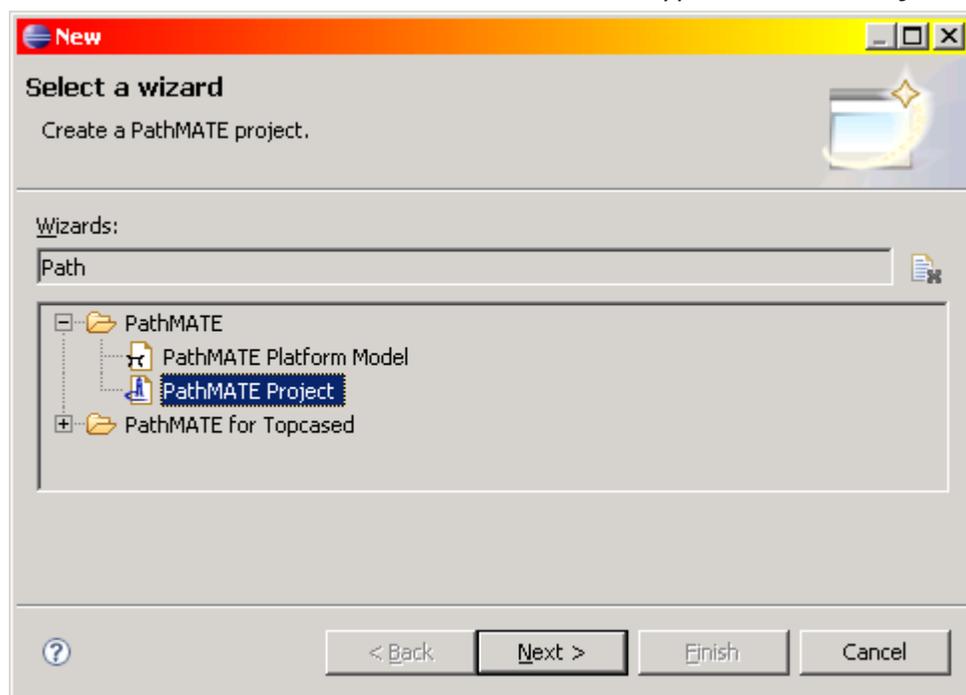
PROCEDURE: Use Eclipse Resource Perspective to Copy Files Needed for C++ and Java Systems

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 *ReferenceSimpleOven* project *Models* folder, select the **cpp** folder, right click and select **Copy**.
3. Select the *QuickStart* project *Models* folder, right click and select **Paste**.
4. Repeat steps 2-3 for the **java** folder.

Task 2: Connect PathMATE Project to your new PIM

PROCEDURE: Create a New PathMATE Project

1. In the *QuickStart* project *Models* folder select the *QuickStart System Model.uml* file.
2. From the Eclipse base menu select **File** → **New** → **Other**.
3. In the Wizards field start to type *PathMATE Project*:



4. Select **PathMATE Project** and click **Next**.

5. Under the *QuickStart* project *Models* folder select the *QuickStart System Model.uml* file and click **Finish**.
6. The PathMATE project file opens in the Editor Pane.

Task 3: Check Your Model

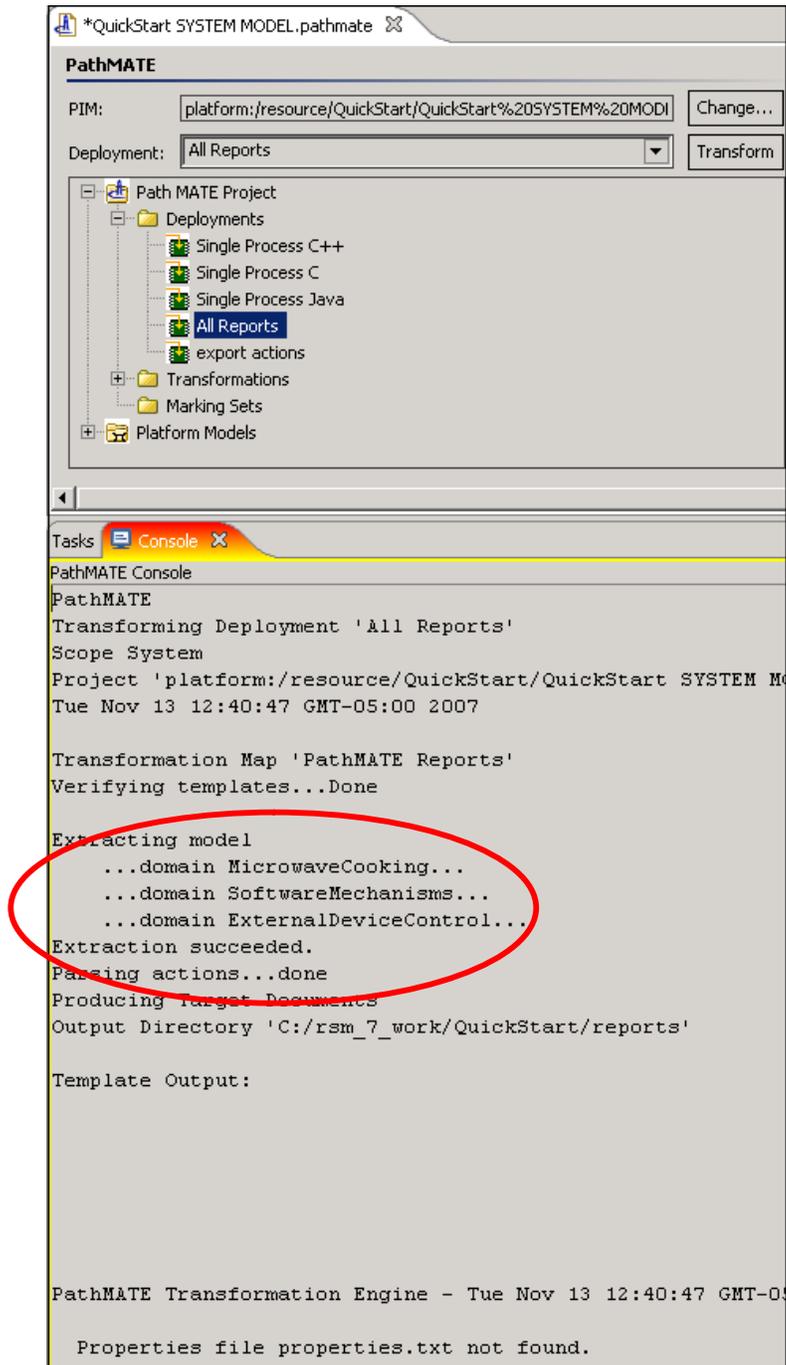
Before going on, this is a good point to check for any errors that may have occurred in entering your model.

We will use the PathMATE documentation Transformation Map *All Reports* and have 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.

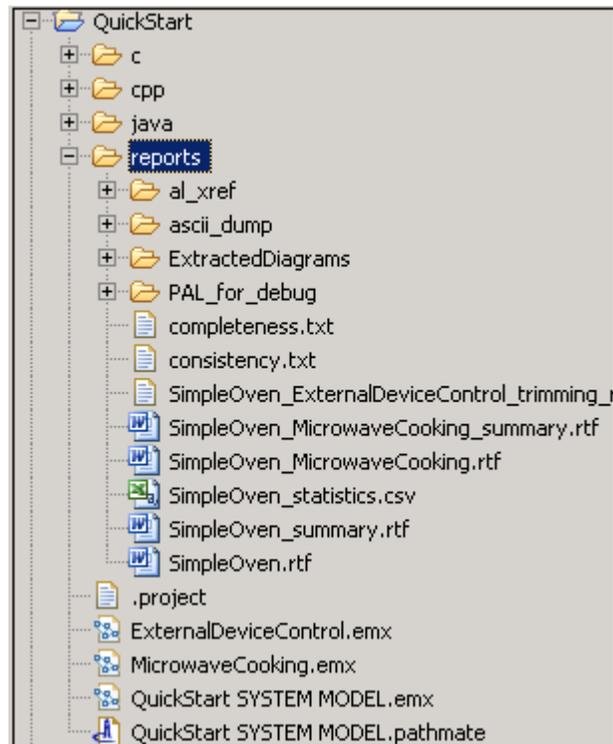
PROCEDURE: Use the PathMATE Project Editor to test Your Project

1. In the Editor Pane select the deployment, **All Reports** from the Deployment pick list.
2. Click **Transform**. A progress dialog will appear. When it finishes, your Console window will show your results.

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



4. Check in the Project Explorer that you have the generated documentation files. If not, you may have a problem with your model.



5. 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.
6. Correct any problems identified in the messages, then return to QuickStart System Model.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 `QuickStart/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.



2. Add the following as the top line:

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

3. Add the following as the bottom line:

```
System,SimpleOven,Defines,_CRT_SECURE_NO_DEPRECATED;_AFX_NOFORCE_LIBS
```

4. If you are building on Linux also add the following as the bottom line.

```
System,SimpleOven,TargetOS,Linux
```

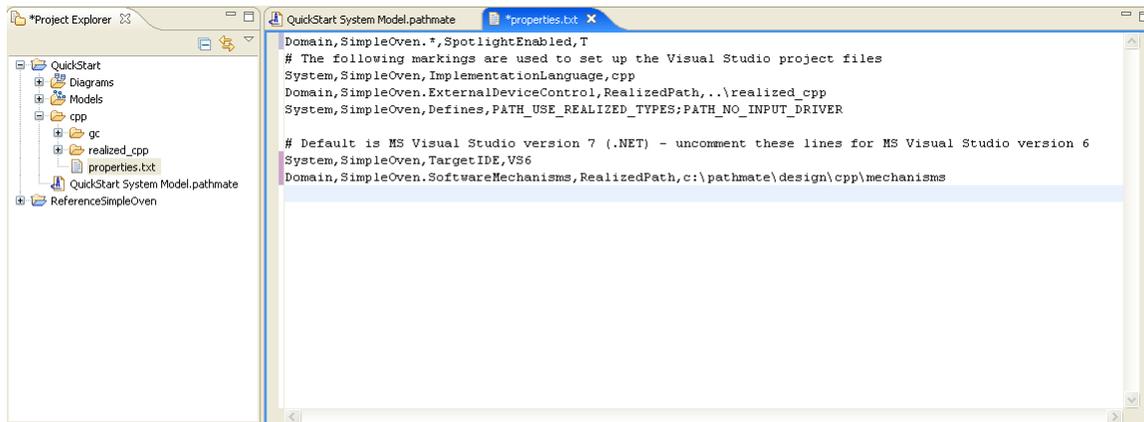
5. Select **File** → **Save** to save the changes.

PROCEDURE: Change Markings to select versions of Visual Studio

The default target when generating IDE project files is to support Visual Studio version 7 (SimpleOven.vcproj). To generate Visual Studio version 6, 8 or 9 project files, 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 second-to last line in the file.
2. Leave the value *VS6*, or change it to *VS8* or *VS9* as appropriate.
3. If you set TargetIDE to *VS6*, also uncomment the last line in the file:



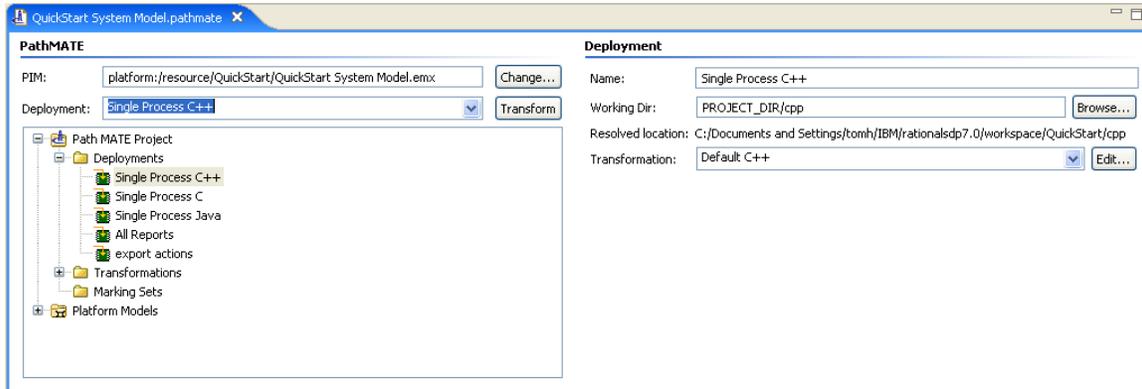
4. Select **File** → **Save** to save the changes.

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

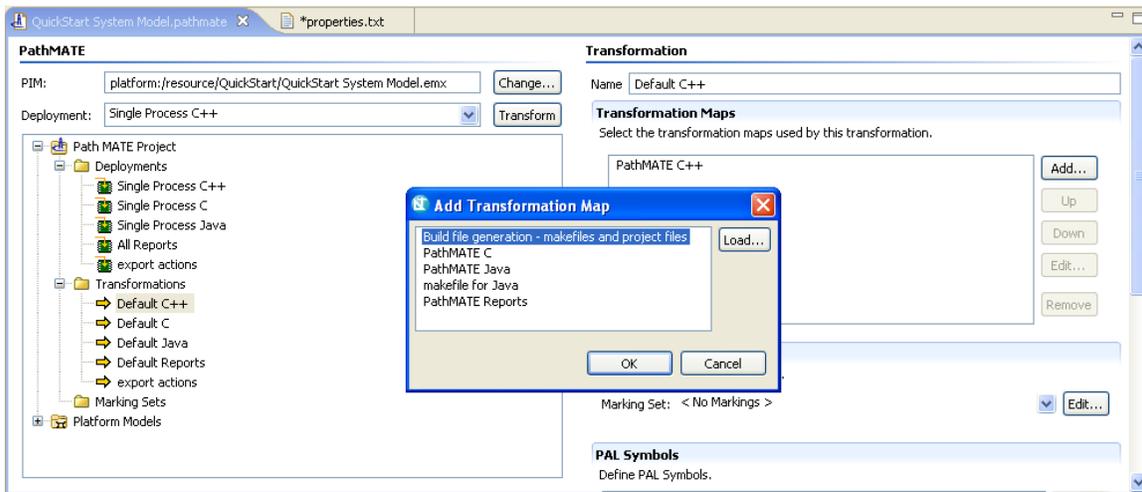
PROCEDURE: Generate C++ Implementation Code and a Visual Studio Project File/Makefiles

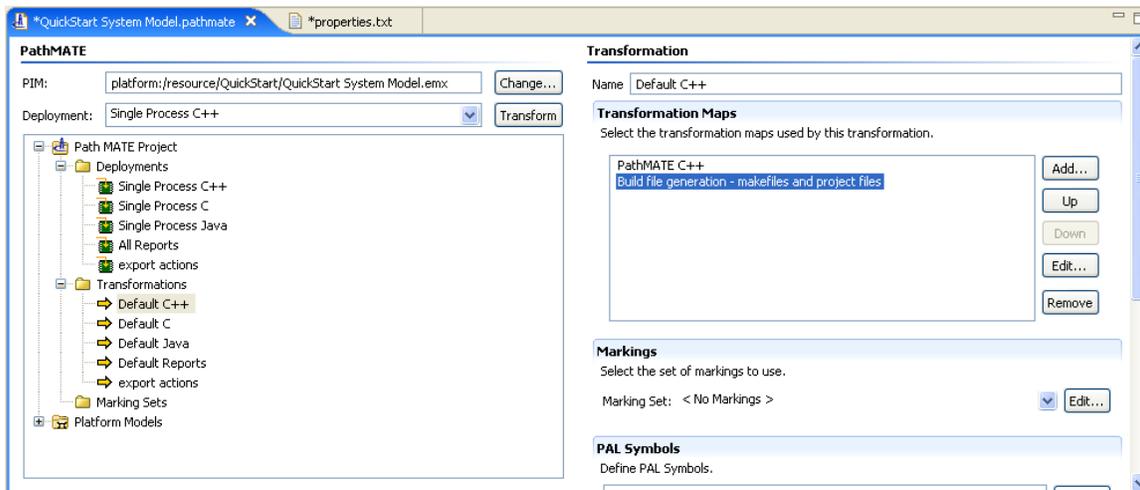
1. Open *QuickStart.pathmate* in the editor pane.

- Verify that the deployment *Single Process C++* is selected in the Deployment pick list.

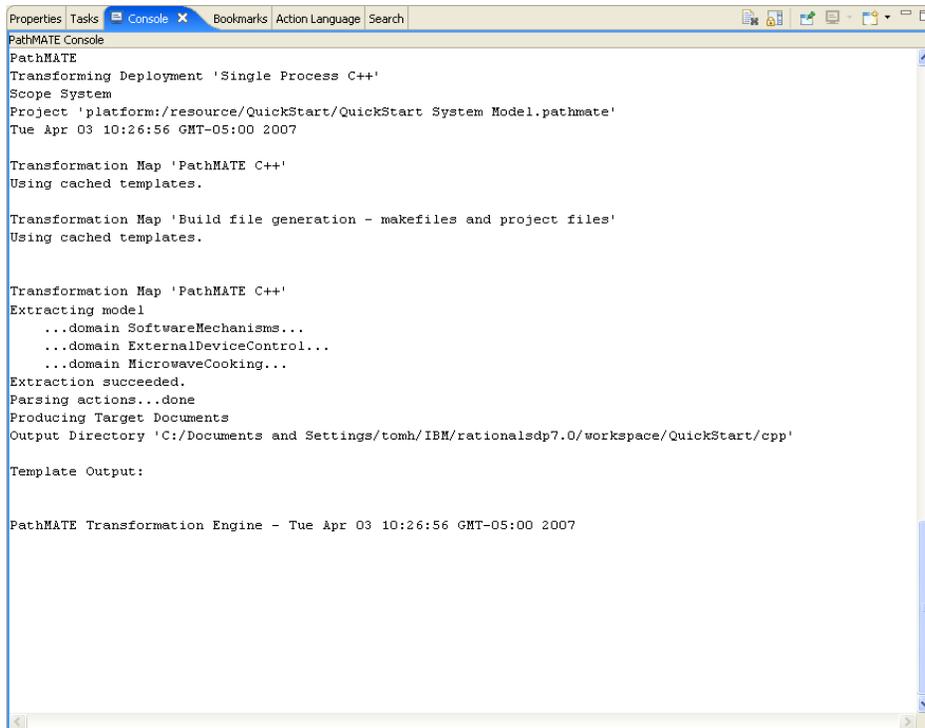


- Click the **Edit...** button to the right of the Transformation drop down. In the Transformation Maps window ensure the *PathMATE C++* and *Build file generation* Maps appear in order. If not, use the Add and/or Up/Down buttons to get both maps to appear in the proper order.





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.)



```

PathMATE Console
PathMATE
Transforming Deployment 'Single Process C++'
Scope System
Project 'platform:/resource/QuickStart/QuickStart System Model.pathmate'
Tue Apr 03 10:26:56 GMT-05:00 2007

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 QuickStart\cpp\MAIN. To launch the Visual C++ environment and build SimpleOven.exe:

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

1. Right-click **SimpleOven.vcproj** under QuickStart, cpp, MAIN in the Project Explorer and select **Open With → System Editor**.
2. Build the SimpleOven system in Visual Studio 7.

PROCEDURE: Build SimpleOven.exe - Visual Studio Version 6

1. Right-click **SimpleOven.dsw** under QuickStart, cpp, MAIN in the Project Explorer and select **Open With → System Editor**.
2. Build the SimpleOven system in Visual Studio 6.

PROCEDURE: Build SimpleOven.exe - GCC

1. In a terminal window navigate to the cpp/MAIN folder.
2. Run the command `make -f SimpleOven.mak`

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

NOTE – you should skip this section if you just completed the **Generate C++ Code and Visual Studio Project Files** section and do not want to build the Java implementation of your system.

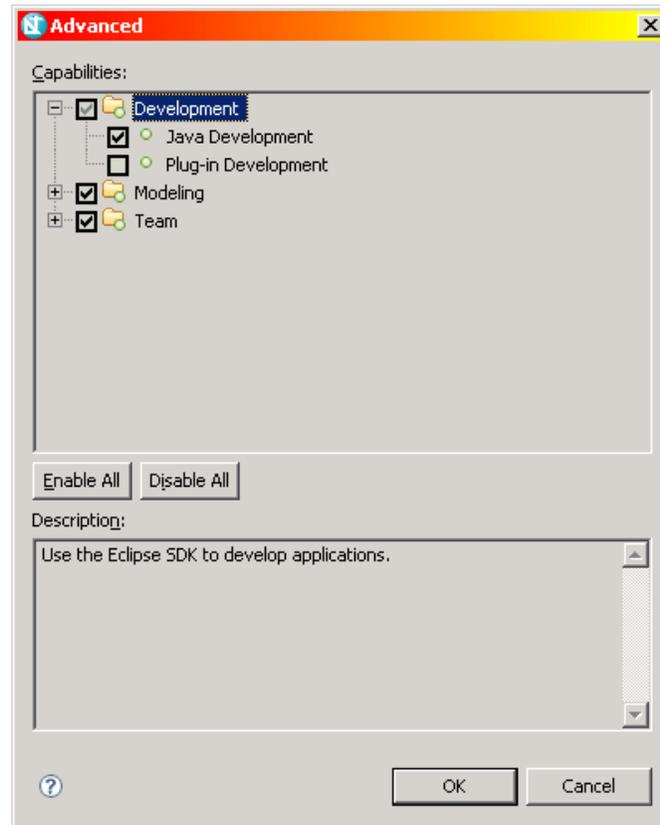
Task 1: Set Up Java and Generate Java Code

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. *If you do not see a **Capabilities** option, then skip this step.* Click the **Advanced...** button.

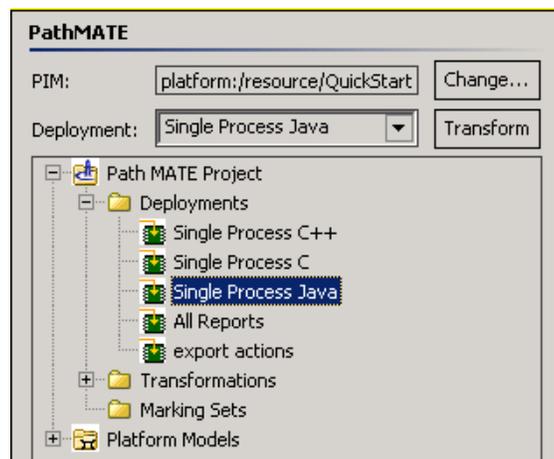
- Expand **Development** and ensure **Java Development** is checked.



- Click **OK** on both dialogs.

PROCEDURE: Generate Java Code

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



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

```
PathMATE
Transforming Deployment 'Single Process Java'
Scope System
Project 'platform:/resource/QuickStart/QuickStart SYSTEM MODEL.pathmate'
Tue Nov 13 12:58:15 GMT-05:00 2007

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:/rsm_7_work/QuickStart/java/gc'

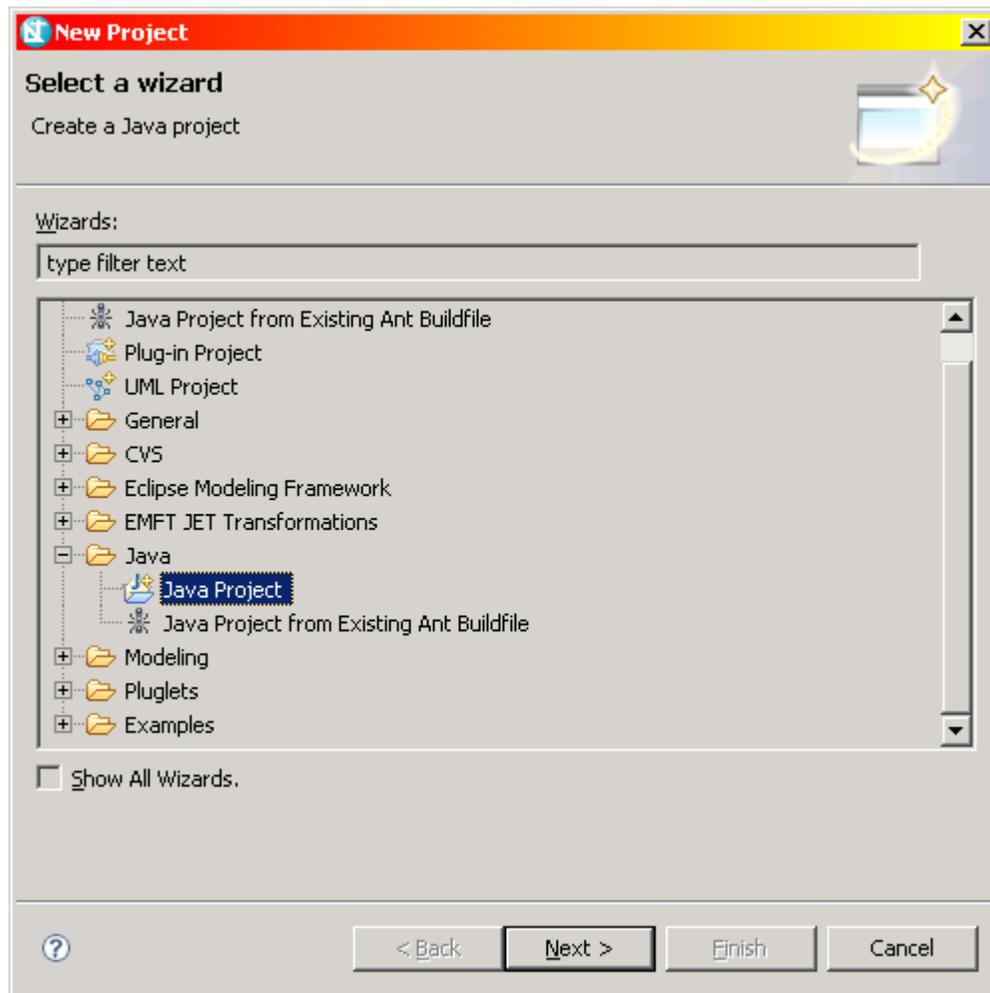
Template Output:
PathMATE Transformation Map for Java code templates version 5.02.004
PathMATE Transformation Engine - Tue Nov 13 12:58:15 GMT-05:00 2007
```

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.

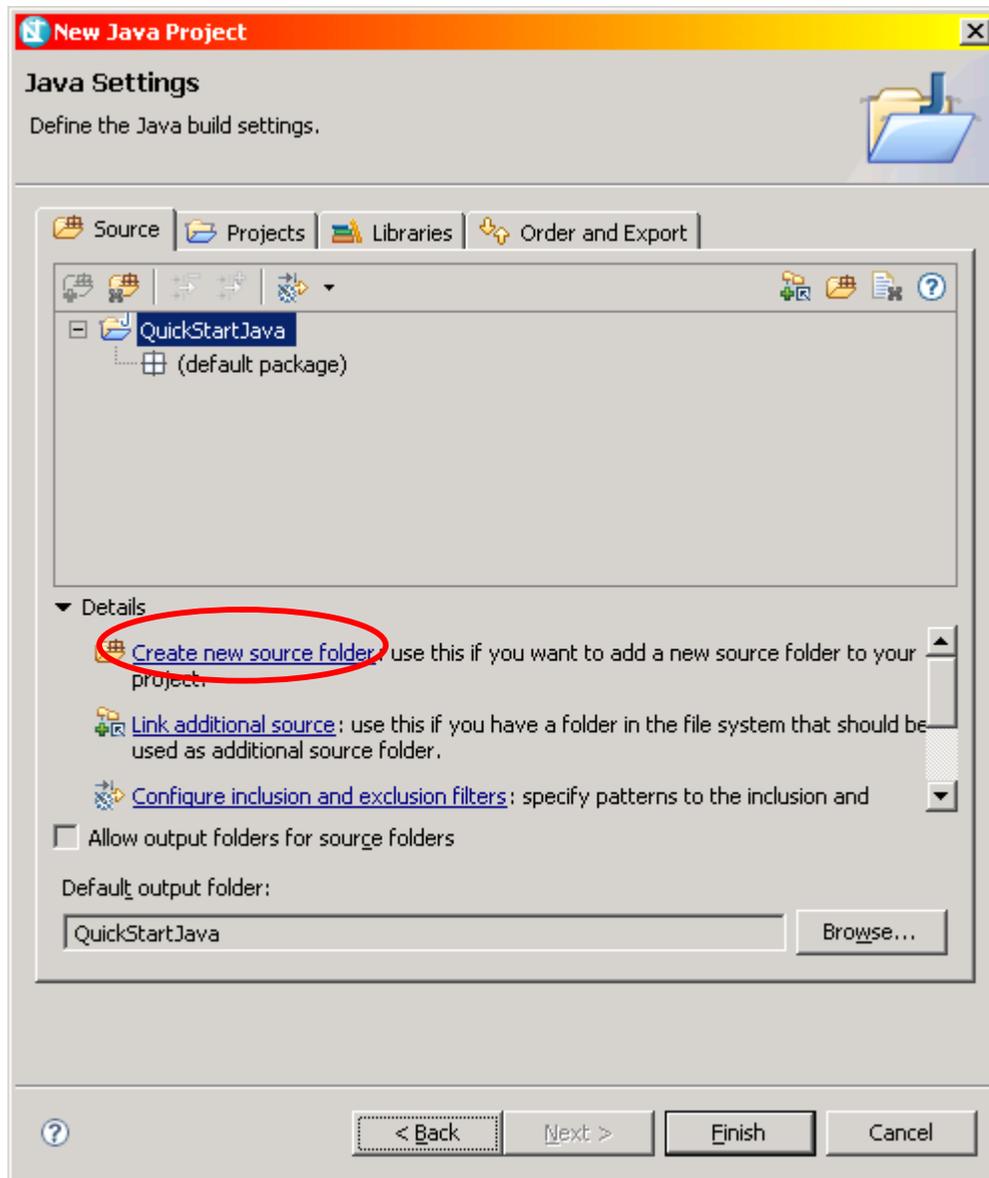
3. Select **Java / Java Project** from Wizards.



4. Click **Next**. The Create a Java Project page appears.

5. Enter the name *SimpleOvenJava*. Click **Next**.

6. If there is no `src` folder under the top level folder, click **Create new source folder**. Enter the Folder name `src`. Click **Finish** on both dialogs.

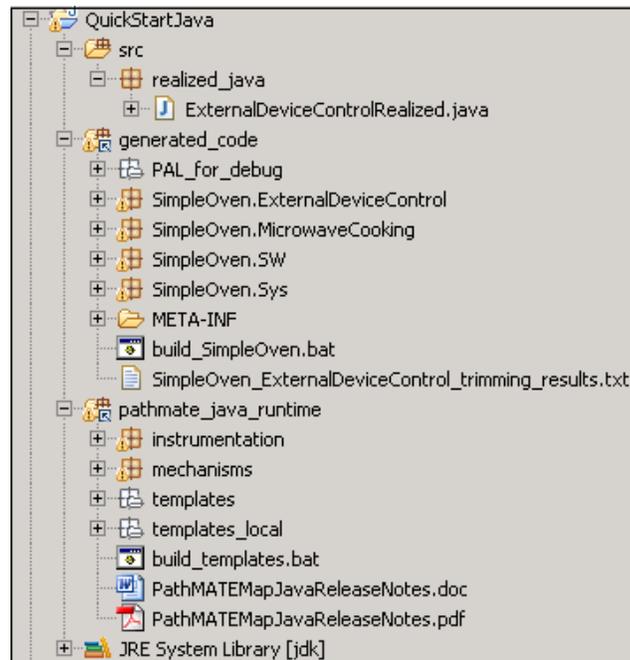


7. In the Package Explorer browser in your `QuickStartJava` project select the `src` source folder, right click and select **New** → **Package**.
8. Enter `realized_java` for the package name, and click **Finish**.
9. In the Package Explorer browser, navigate in the `QuickStart` project `java/realized_java` folder, select `ExternalDeviceControlRealized.java`, and hit control-C to copy it.
10. Back in your `QuickStartJava` project select **src/realized_java**, and hit control-V to paste in `ExternalDeviceControlRealized.java`.

11. In the Package Explorer browser, select your *QuickStartJava* project, right click, and select **Build Path** → **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.



3. Click **Finish**.
4. Select **File** → **Switch Workspace** → **Other**. Hit **control-C** to copy the pathname of your current workspace from **Workspace** field. Click **Cancel**.
5. Go back to your *QuickStartJava* project, right click, and select **Build Path** → **Link Source** again.
6. Paste your workspace pathname into the **Linked folder location** field, click **Browse...** and navigate to the generated code in your *QuickStart* project: `<your Eclipse workspace>/QuickStart/Models/java/gc`. Name this folder `generated_code`.
7. Click **Finish**.
8. Verify that your *QuickStartJava* project has these contents:



- 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

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



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

- 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.

- In addition to generating instrumentation code (controlled by the SpotlightEnabled marking) you will need to enable Spotlight instrumentation in the runtime layer. In the Package Explorer view, expand **QuickStartJava** → **pathmate_java_runtime** → **mechanisms**.
- Double click to open **PfdDefine.java**. The file opens in the Editor pane. Set the *NO_PATH_IE* to *false* as shown below:

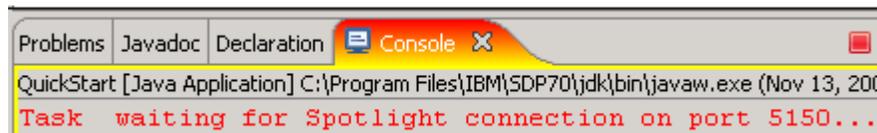


```
public static final boolean NO_PATH_IE = false;
```

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

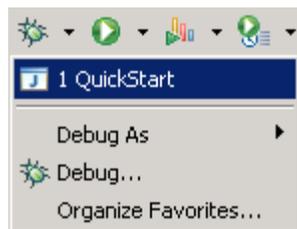
PROCEDURE: Create a Debug Configuration and Smoke Test it.

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



11. 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 QuickStart*:



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.

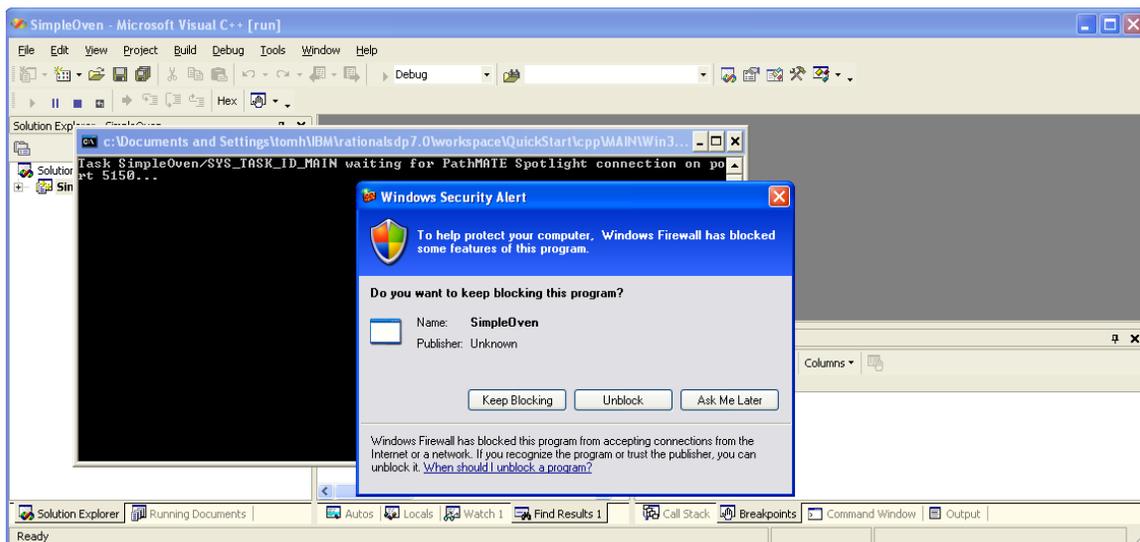
A command window opens to say that the application is running and waiting for a connection to Spotlight.

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.

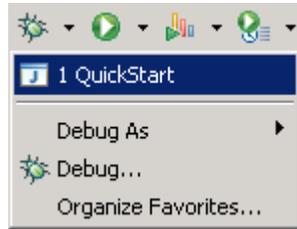


2. If a Windows Security Alert appears, click **Unblock**.
3. Skip ahead to PROCEDURE: Use Eclipse to Launch the Spotlight Debugger.

PROCEDURE: Run Java SimpleOven from the Eclipse JDT

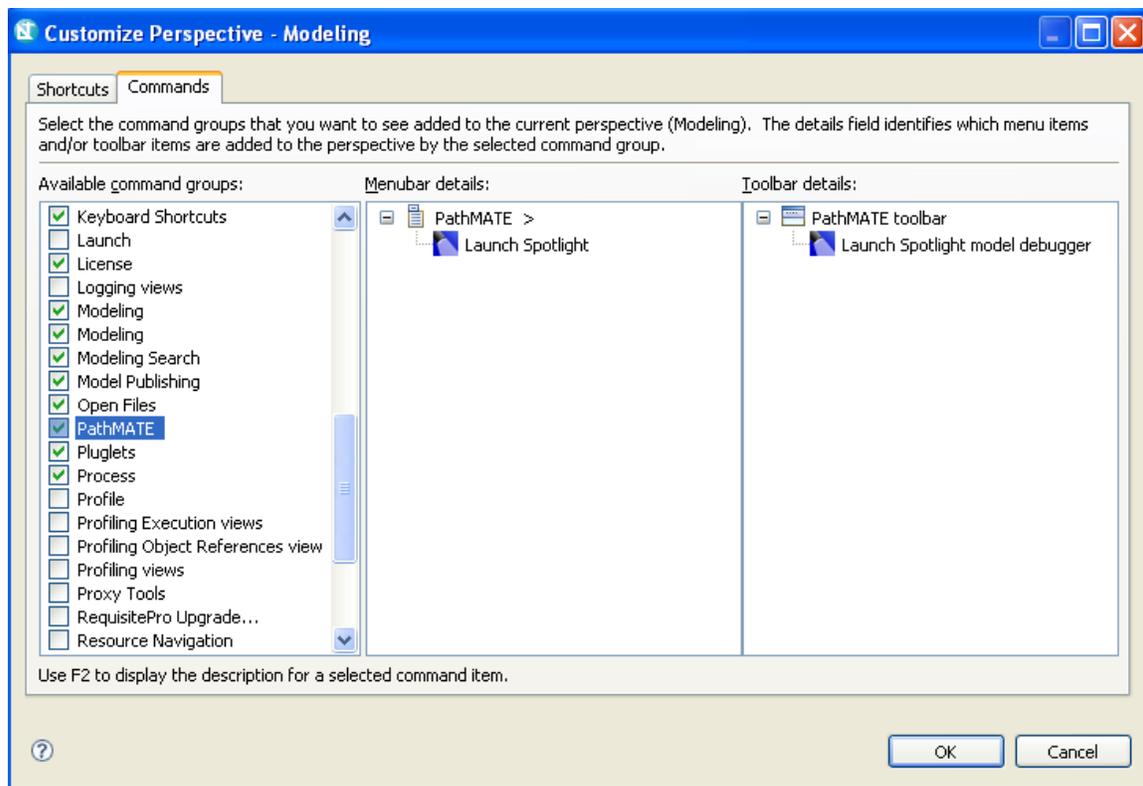
If you built C++, please skip this procedure.

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



PROCEDURE: Use Eclipse to Launch the Spotlight Debugger

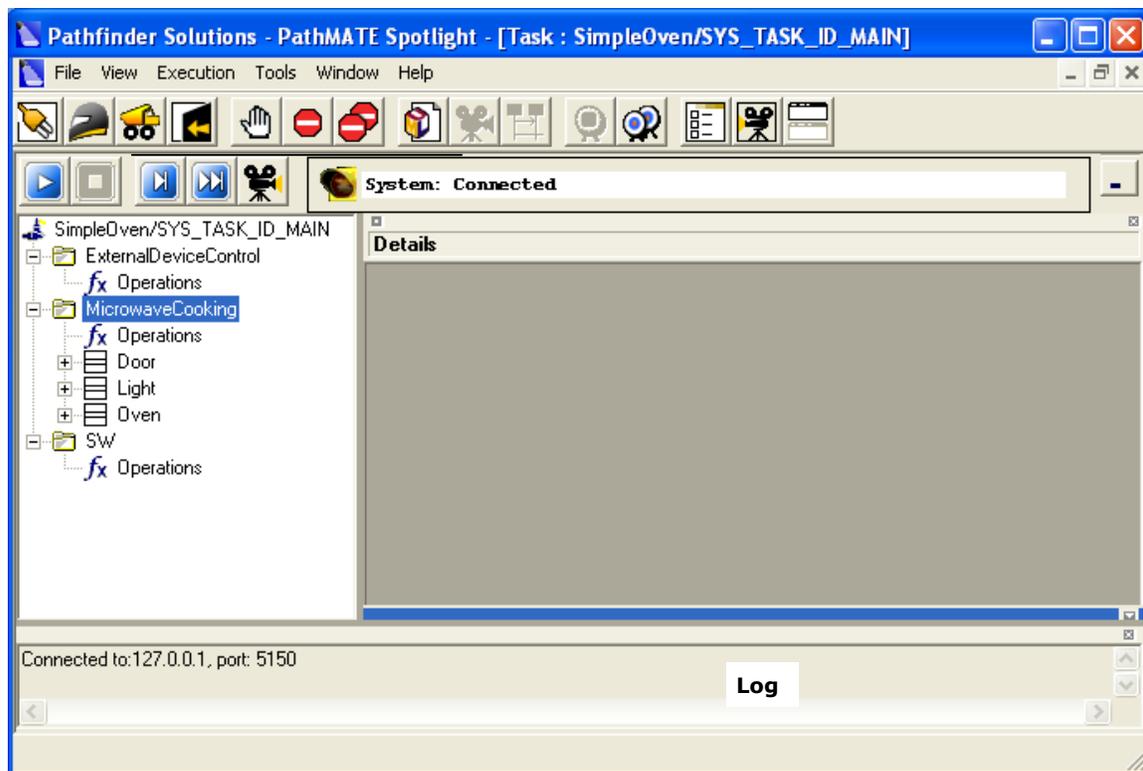
1. If a PathMATE menu does not appear on the top menu bar, open the **Customize Perspective...** option in the Window menu. Select the **Commands** tab and check the **PathMATE** command group on the Commands tab:



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

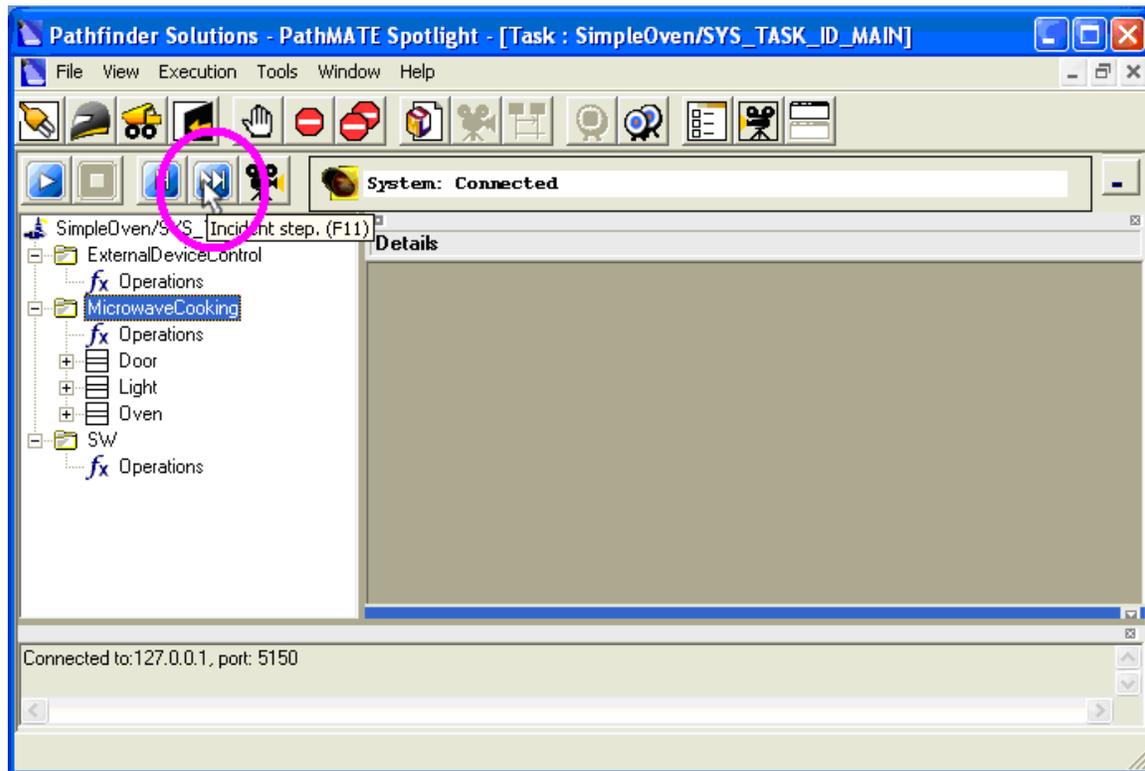
PROCEDURE: Connect Spotlight to the Running SimpleOven Program

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:

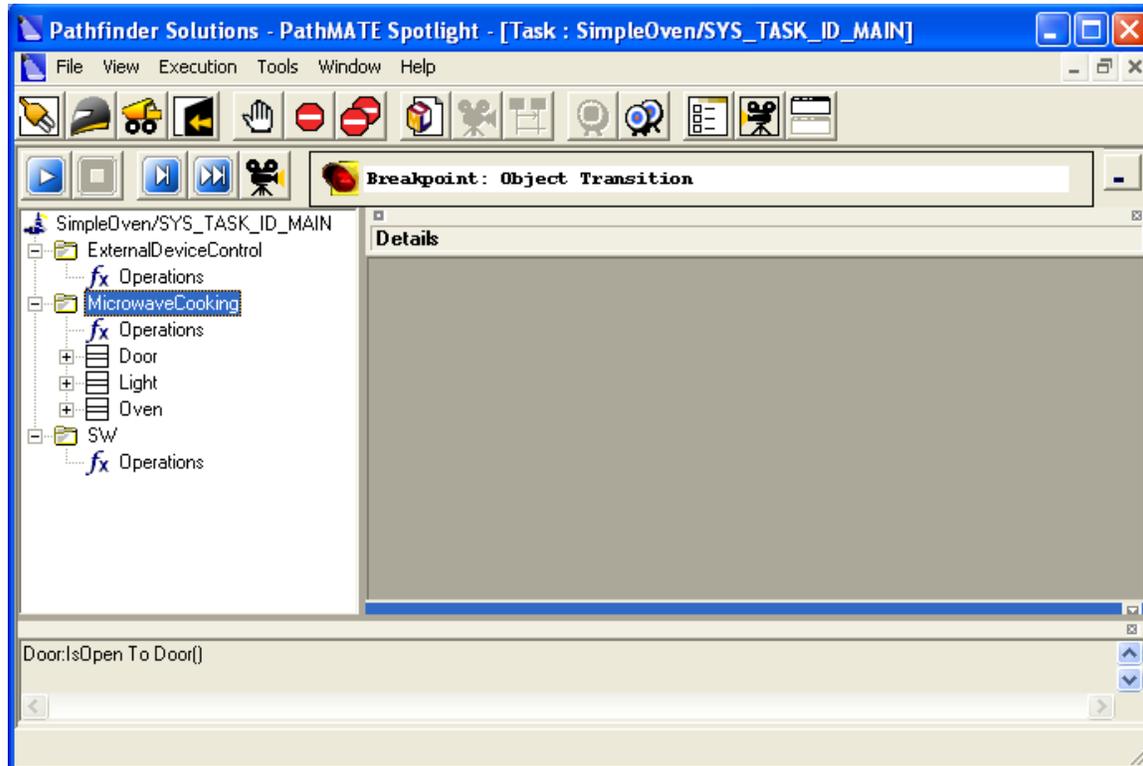


PROCEDURE: Use Spotlight to Initialize the SimpleOven System

1. Locate the Incident Step button .



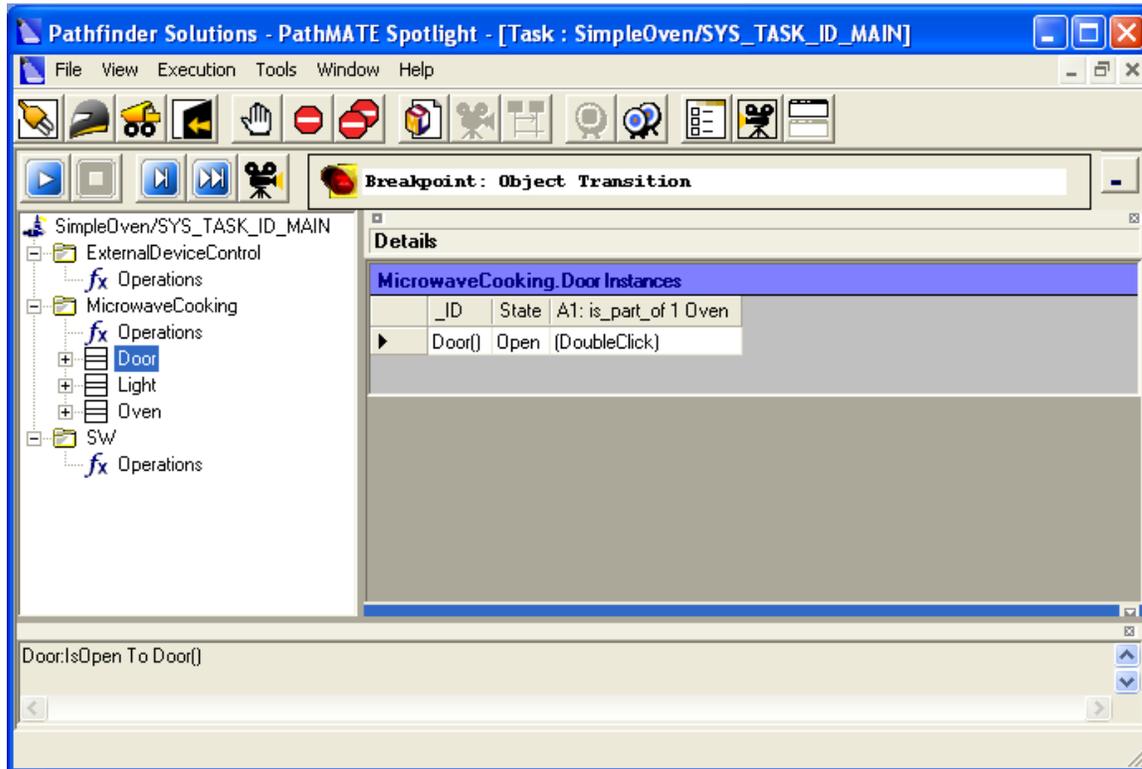
2. Click the **Incident Step button** . The status indicator changes from *Connected* to *Object Transition*.



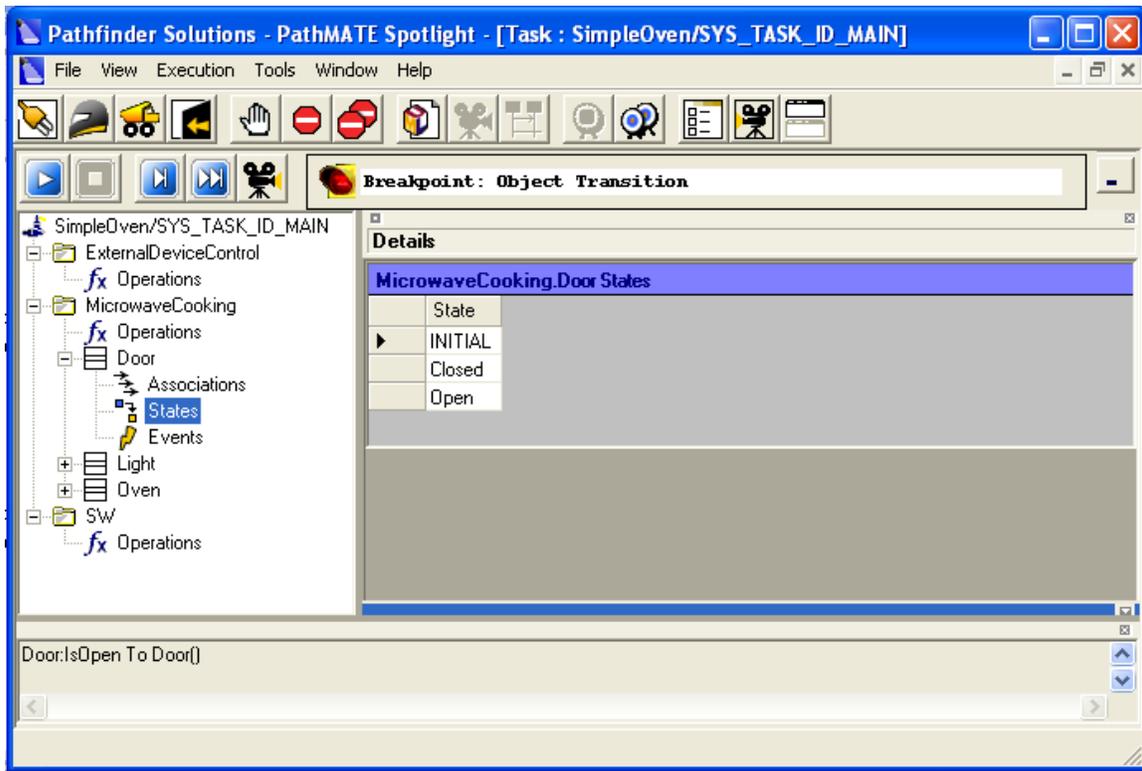
PROCEDURE: Use Spotlight to Browse the Door Class

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

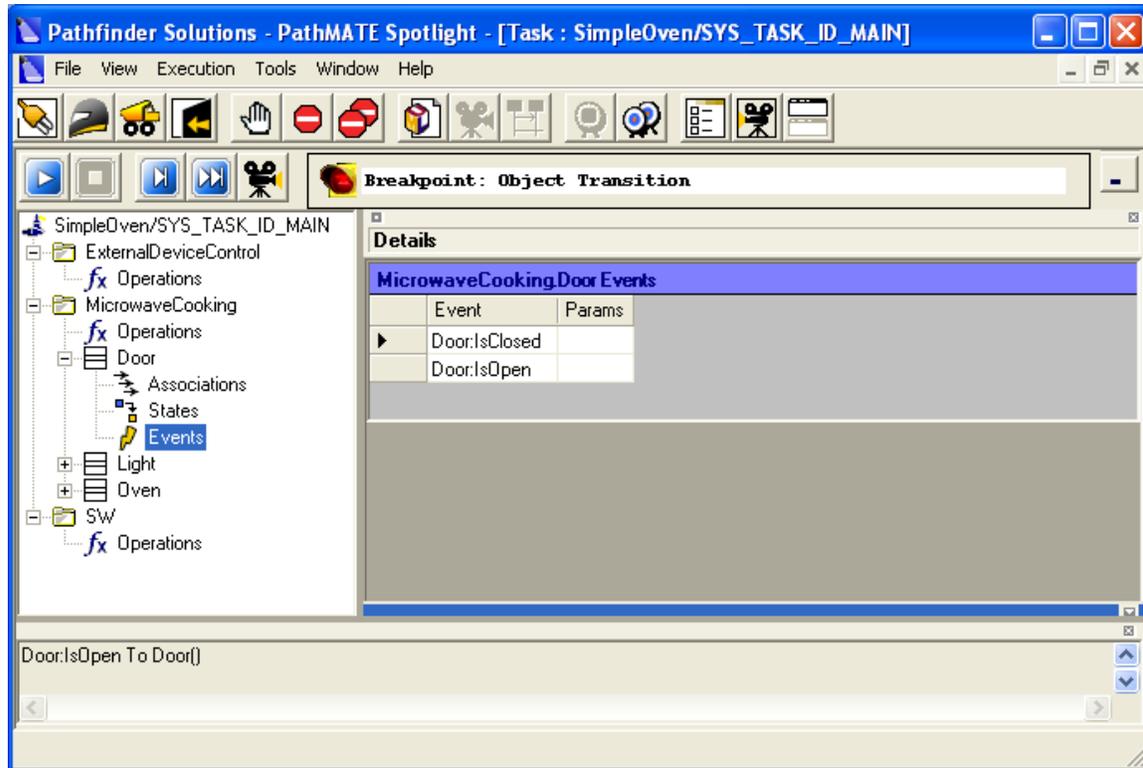
The details pane shows one instance of the Door class. The door object is in the Open state, and it is part of one oven (association A1).



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 under <<Housekeeping>> Domain Support for <<Domain>>MicrowaveCooking in the Project Explorer.

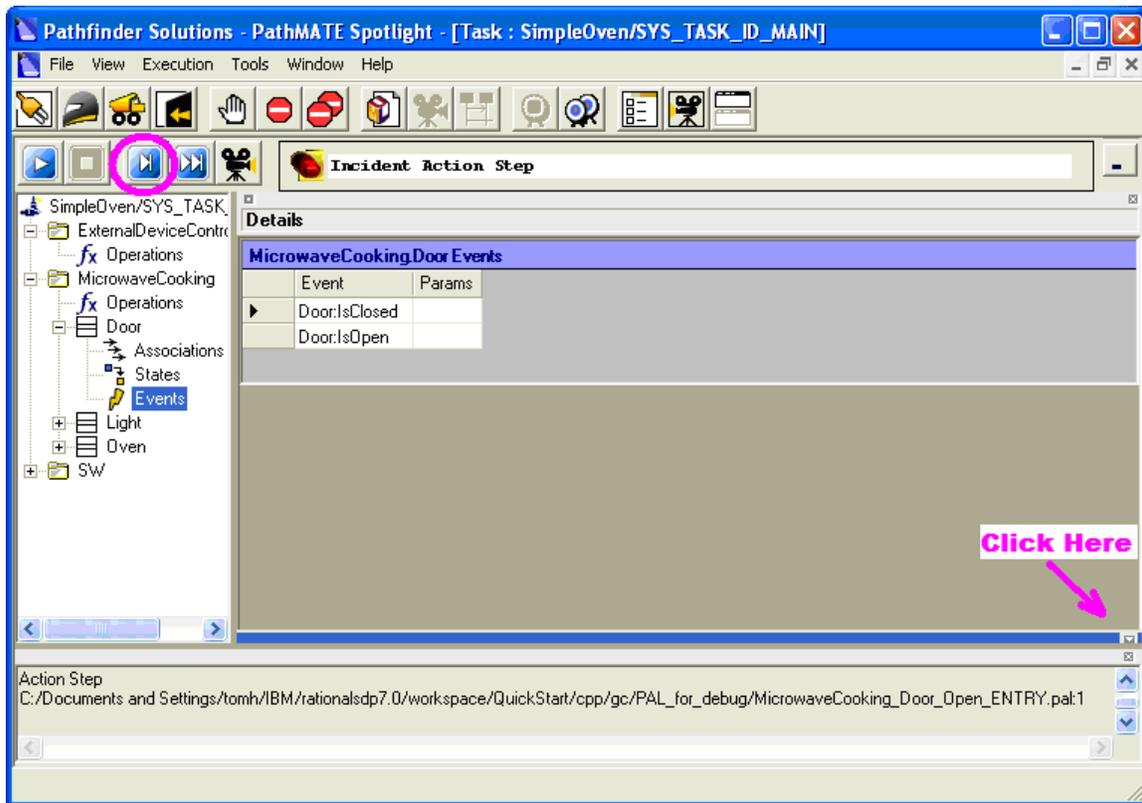


4. Select **Events** in the browser. The details pane shows the events that you can send to the *Door* class.

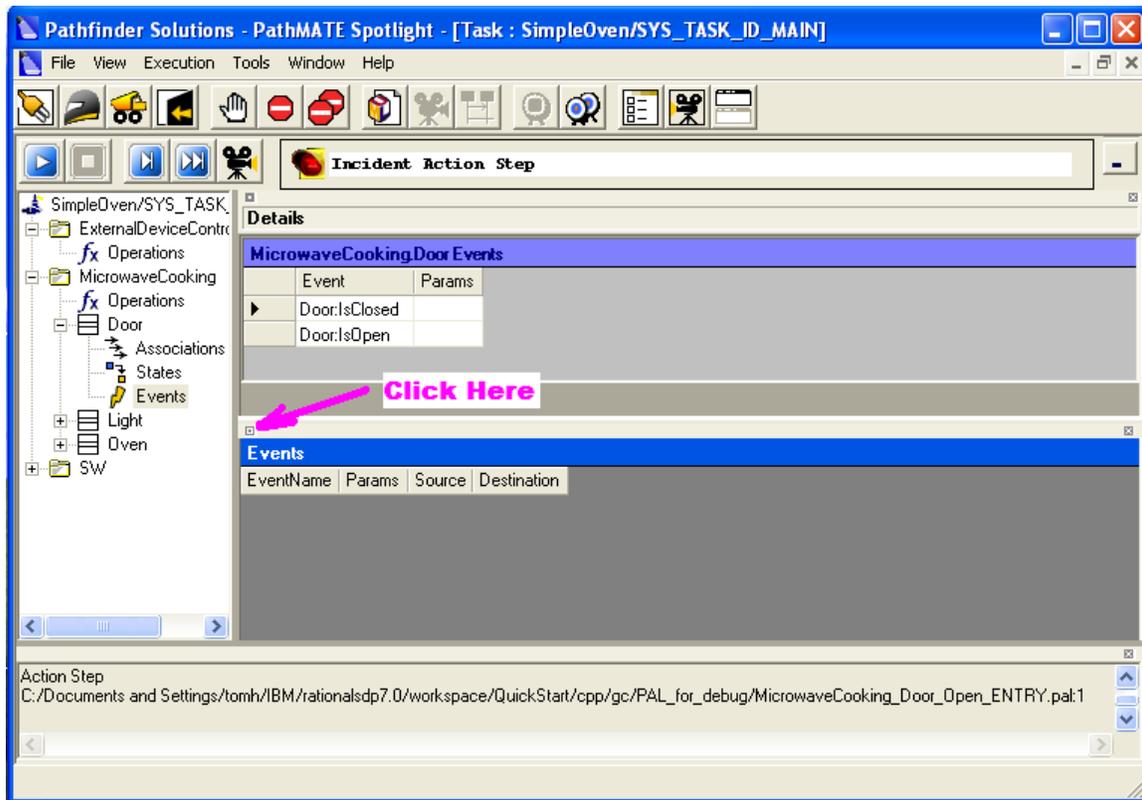


PROCEDURE: Use Spotlight to step through SimpleOven's Action Language

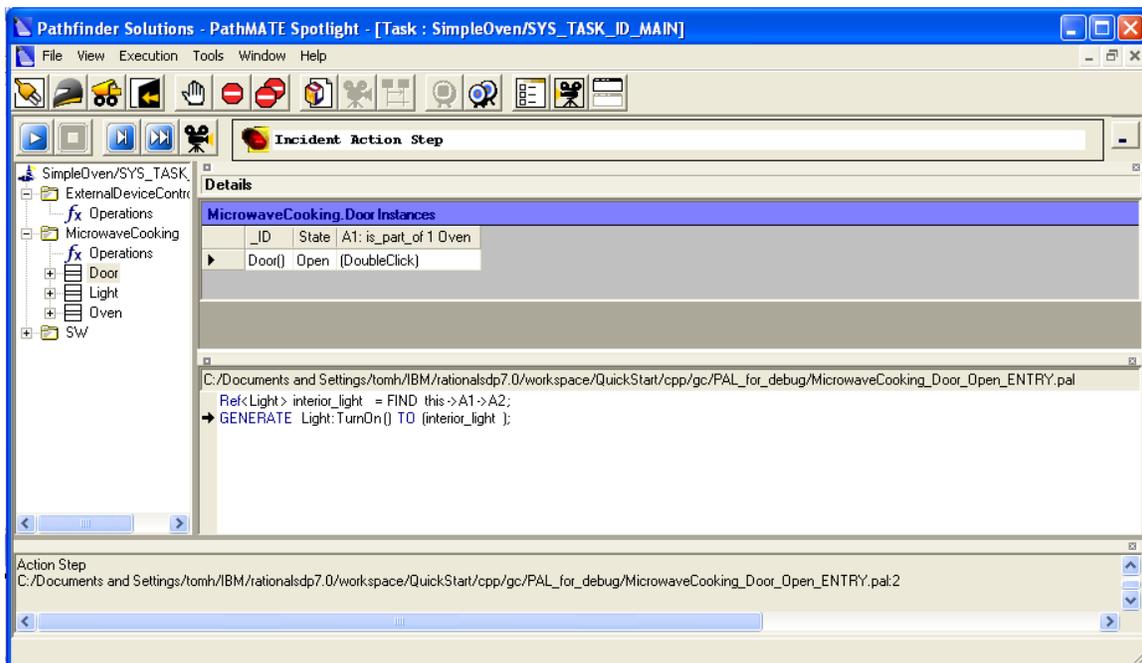
1. Click the **Action Step** button . *MicrowaveCooking_Door_Open.pal* appears in the state window.
2. Open the **State Window**. The figure below shows where to click in the lower right to open and close the action/event window.



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



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

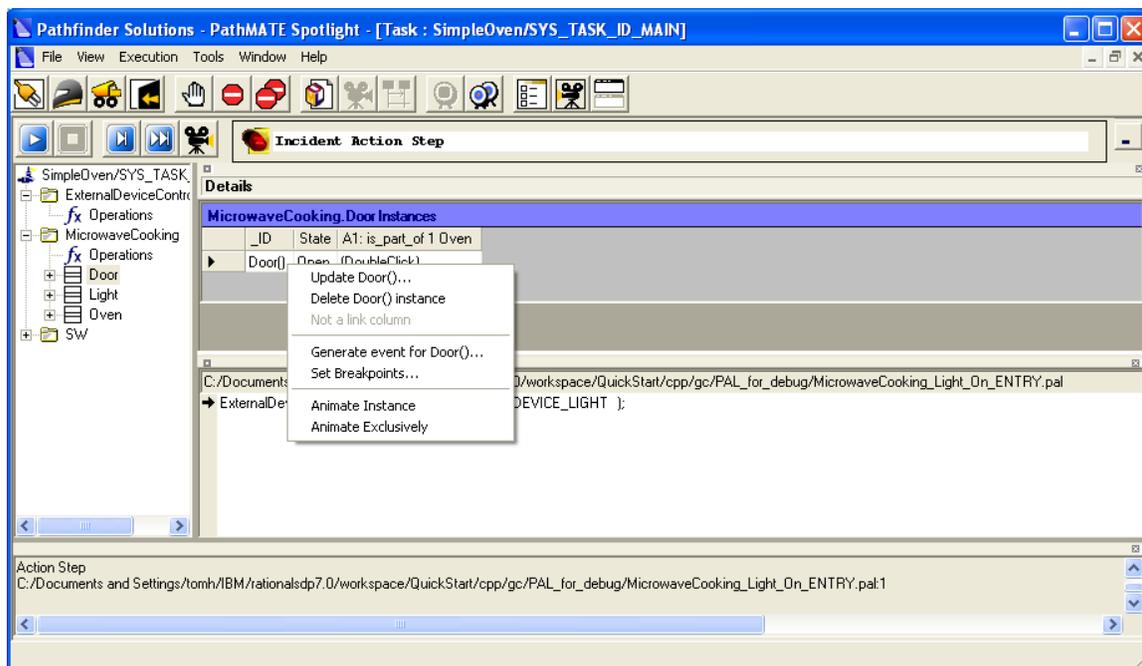


- Continue to click the **Action Step button**  until *MicrowaveCooking_Light_On_entry.pal* appears in the PAL viewer.

(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

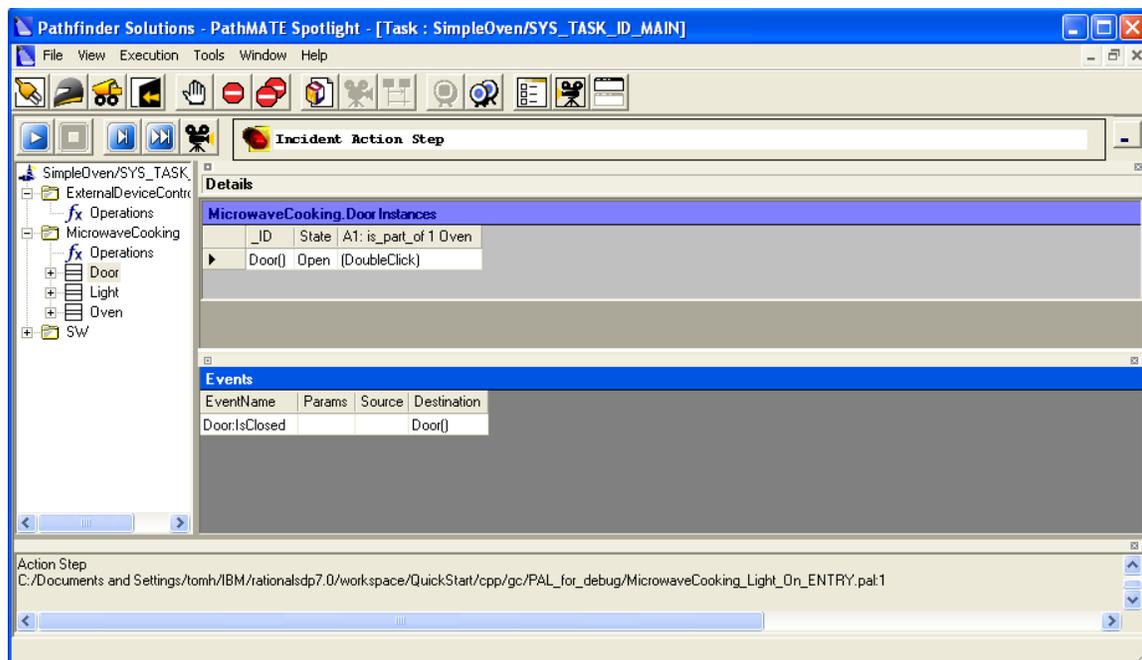
- Select **Door** in the browser. Right-click the **Door()** field in the details pane to bring up a context-sensitive menu:



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



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



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.

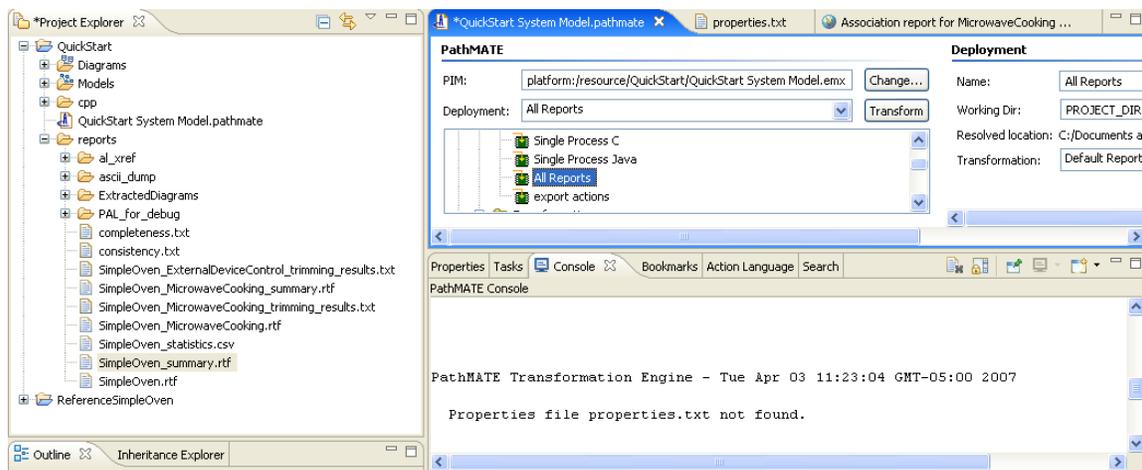
Generate System Documentation

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 QuickStart 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.pathfindermda.com contain additional information. Most importantly, try the PathMATE toolset with real code, as outlined in this *Guide*. We can help you get started.

Pathfinder Solutions

Headquartered in Foxboro, Massachusetts, Pathfinder Solutions provides embedded software engineers with the tools, methods and services needed to boost developer productivity and improve quality. Pathfinder Solutions is Platform Member of the Object Management Group (OMG).

If you would like to learn more, please contact us at:

Pathfinder Solutions
33 Commercial Street, Suite 2
Foxboro, MA 02035 USA
Phone: 888-662-7284
Email: info@pathfindermda.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