An Introductory Guide to SpecTRM

SpecTRM (pronounced “spectrum” and standing for Specification Tools and Requirements Methodology) is a toolset to support the specification and development of safe systems and software. This system development environment supports assurance through inspections, formal validation tools, and simulation. SpecTRM emphasizes finding errors early in the development cycle so they can be fixed with the lowest impact on cost and schedule. The tool facilitates tracing not only requirements, but also design rationale (including safety constraints) throughout system design and documentation.

This tutorial is an introductory walkthrough. Following the tutorial, you will learn to:

- Create a new project
- Create a new intent specification
- Write requirements in the intent specification document
- Create hyperlinks to record traceability information
- Model component requirements
- Create a simulation of component behavior
- Use static analysis to examine the completeness of component requirements

The example used in the tutorial is a simple thermostat. Once you are comfortable with the ideas in this tutorial, you may wish to look at the altitude switch and cruise control examples included with SpecTRM. Although still quite simple, their system and component requirements are more sophisticated than the thermostat presented here. Additional information is also available in the other books in the online help system.

If you have any questions, please feel free to contact us at support@safeware-eng.com.

Run SpecTRM

Depending on what you chose during installation, SpecTRM may be run from your Start menu or by double clicking a desktop icon. Run SpecTRM now. SpecTRM will open a window like the one shown in figure 1, below.

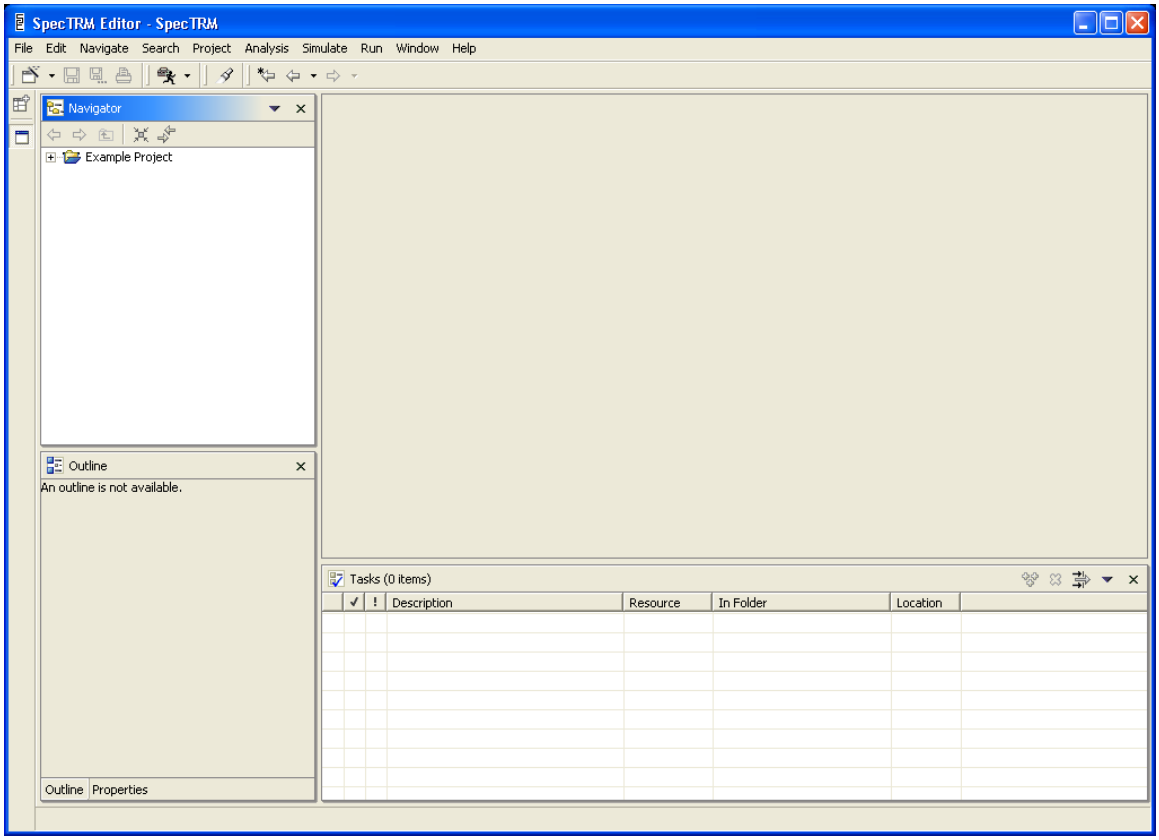

Figure 1 - SpecTRM application window

At the top of the window is a menu bar of commands. Beneath that is the tool bar. Vertically along the left side of the window is a bar of buttons for perspectives. A perspective is a collection of windows that work together to perform a task. The perspective you are in now is called the SpecTRM Editor Perspective, or just the editor perspective, for short. We'll come back to perspectives later.

The editor perspective is divided into four major sub-windows, called views. The view in the top-left corner of the perspective is the navigator view. The navigator view is used to browse through projects and open, cut, copy, paste, move, rename, and delete files and folders. The top right view, empty right now, is used for the editors used to create a specification. The bottom left view is for an outline of the contents of the editor. Because no files are open for editing, the outline view displays that an outline is not available. The bottom right view is the task list, which is used by SpecTRM to warn of illegal syntax in requirements models.

Creating a Project

1. Select the **File** > **New** > **Project…** command.

2. When the dialog box appears, select SpecTRM from the left side and SpecTRM Project from the right side, as shown in figure 2.

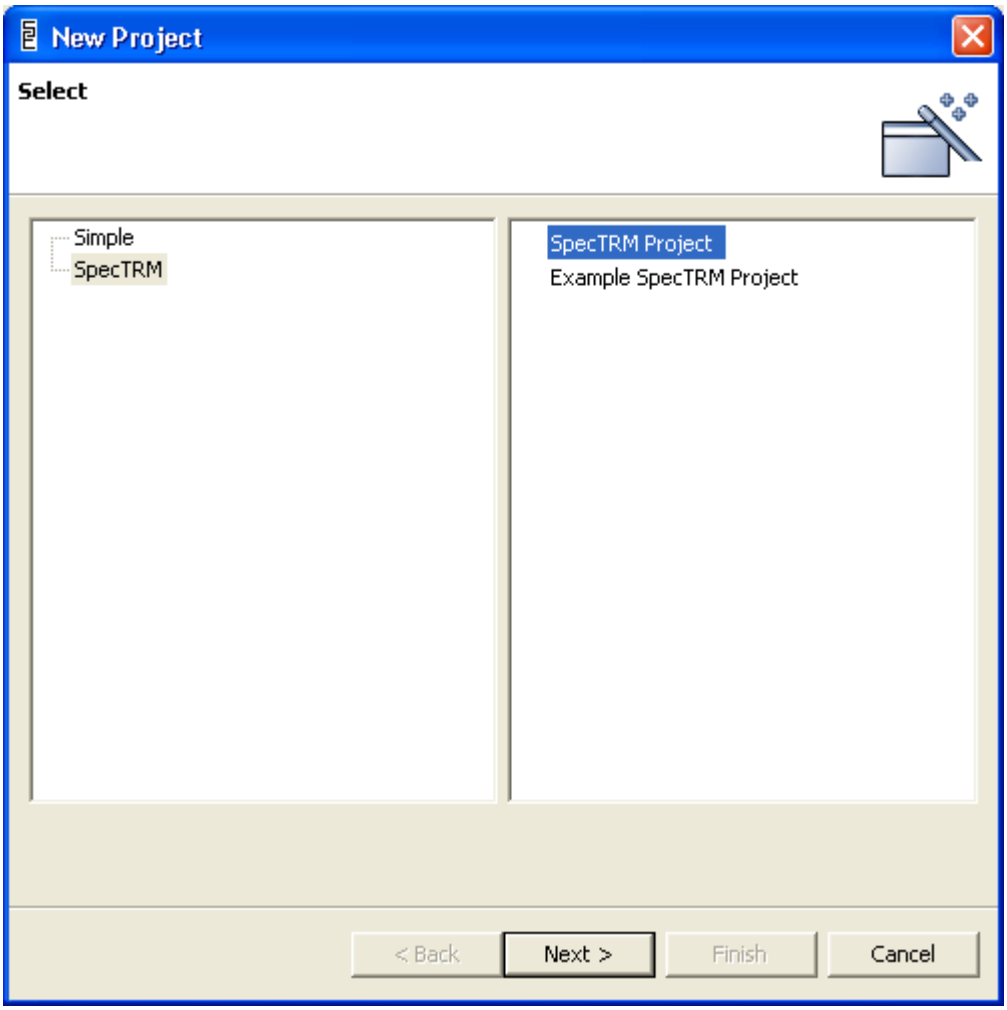

Figure 2 - Choose New Project Type

3. Click **Next**. Then on the next page, choose a name for this project, as shown in figure 3, below.

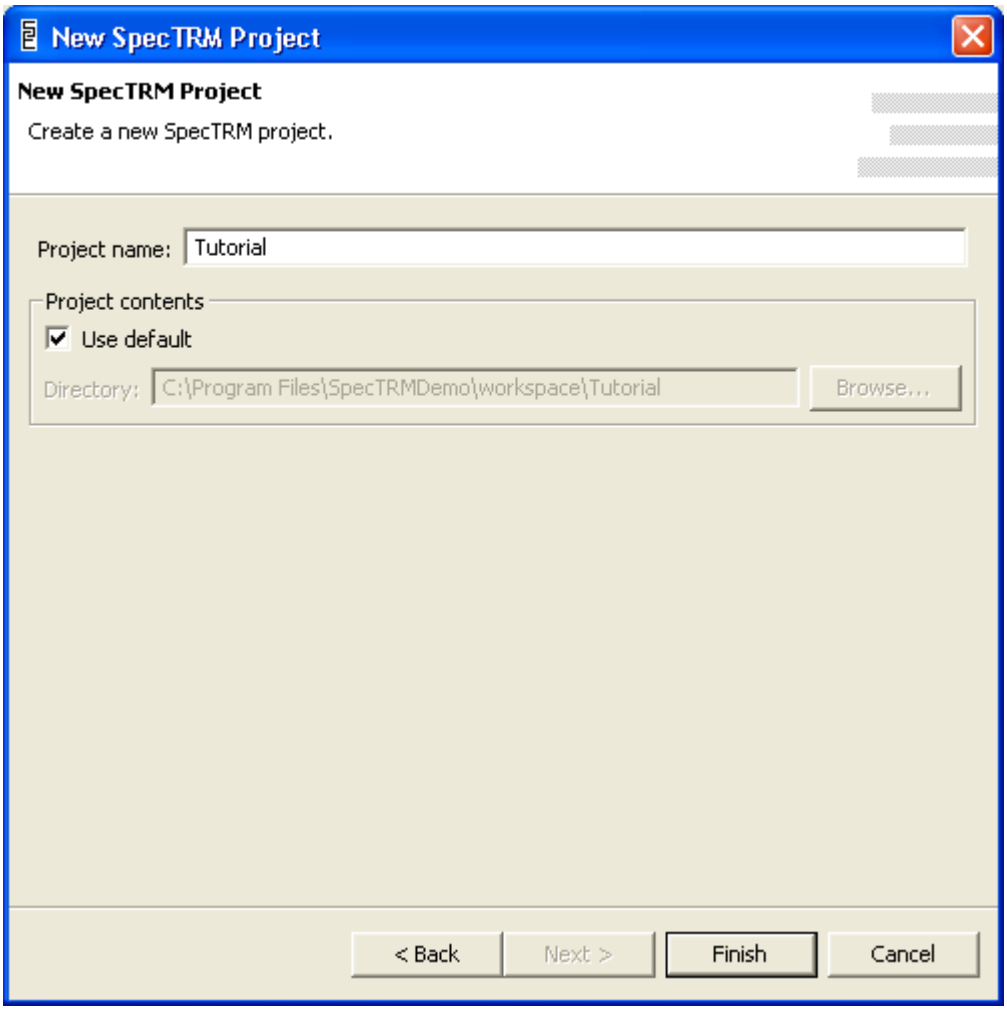

Figure 3 - Choose Project Name

4. Click **Finish** to create the project.

SpecTRM will create the new project in your workspace. The tutorial project will appear in the navigator view, as shown in figure 4, below.

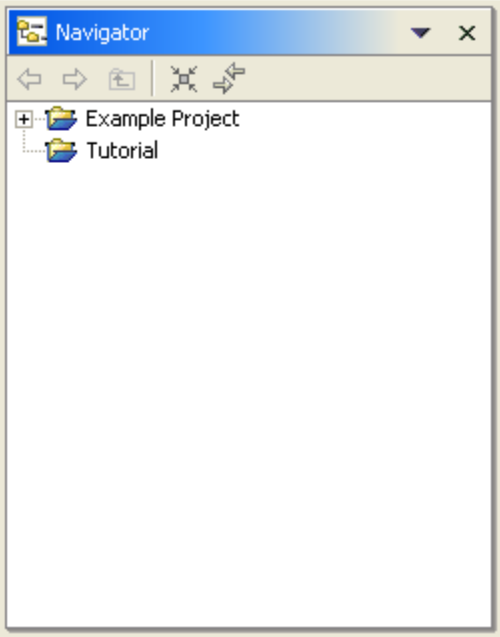

Figure 4 - Tutorial Appears in the Navigator View

This new project will contain all the files associated with the tutorial project.

Creating an Intent Specification

SpecTRM works with documents called intent specifications. An intent specification organizes information in a way that supports system and safety engineering. For more information on intent specifications, consult the papers available at http://www.safeware-eng.com/index.php/publications

1. Select the tutorial project in the navigator view and right-click. Choose **New** > **Intent Specification** from the menu, as shown in figure 5, below.

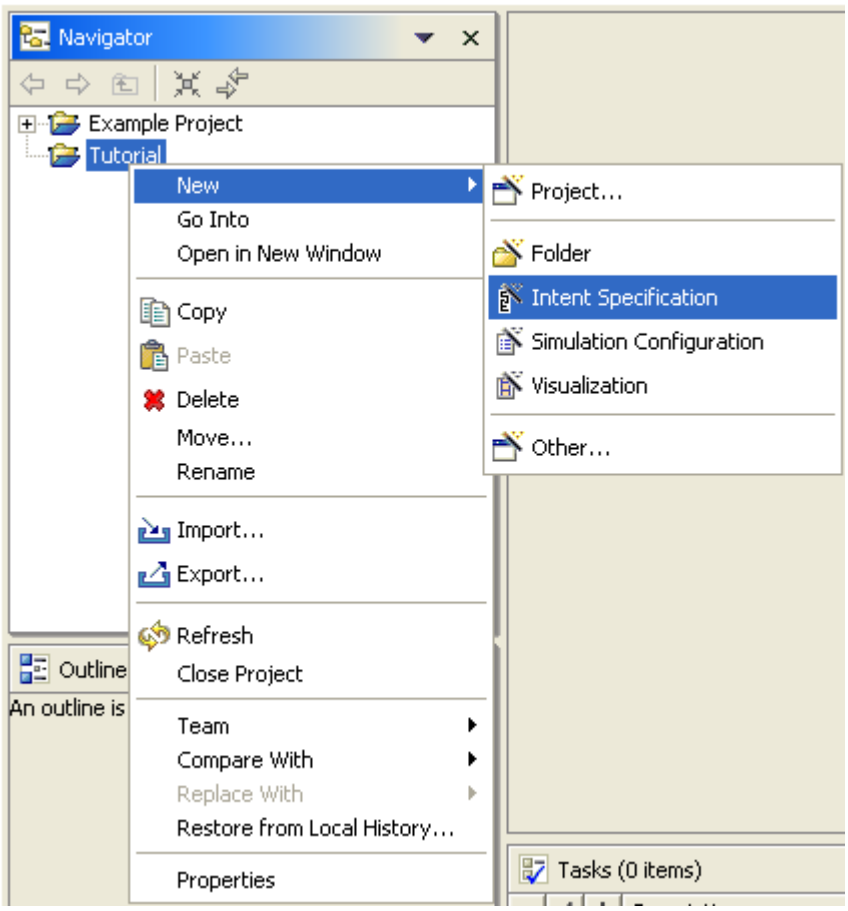

Figure 5 - New Specification Command

2. Enter a name for the specification file, as shown in figure 6, below.

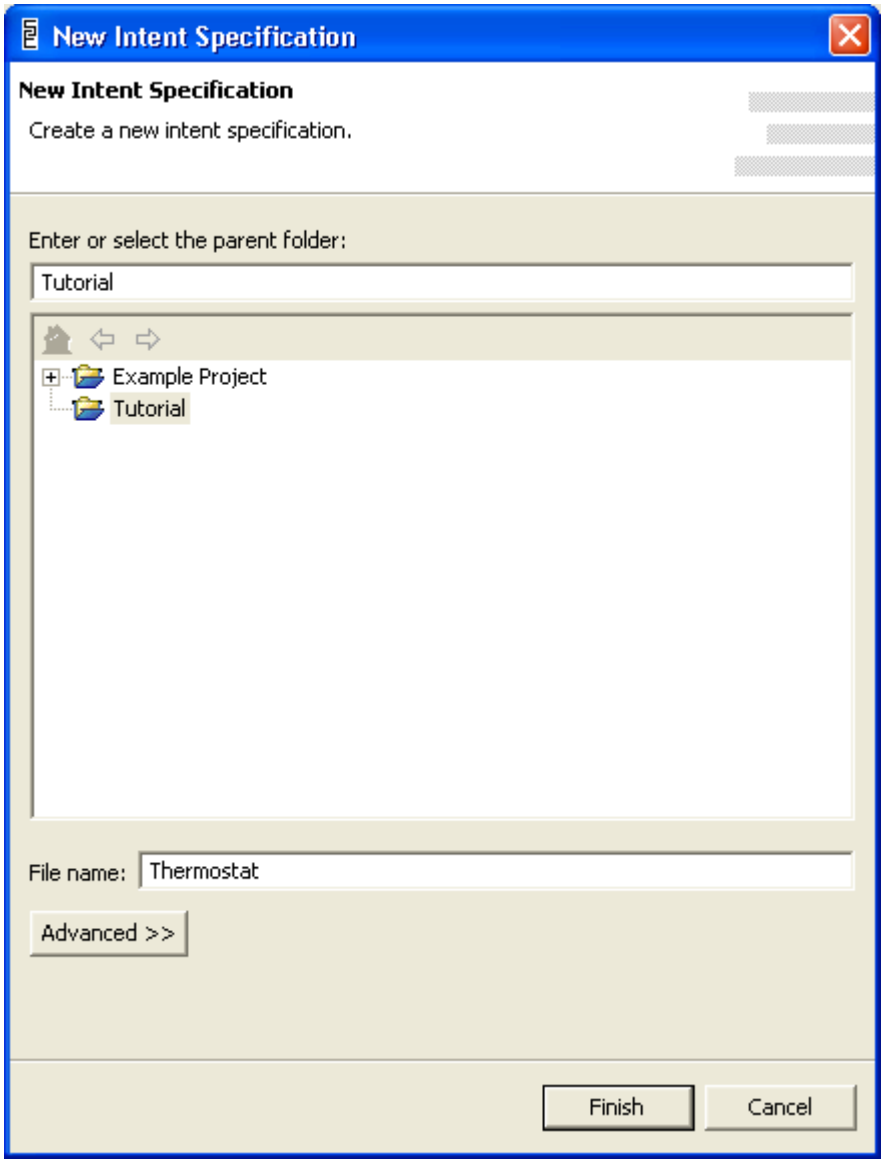

Figure 6 - New Specification Name

SpecTRM will create a new intent specification file and open it in an editor window, as shown in figure 7, below.

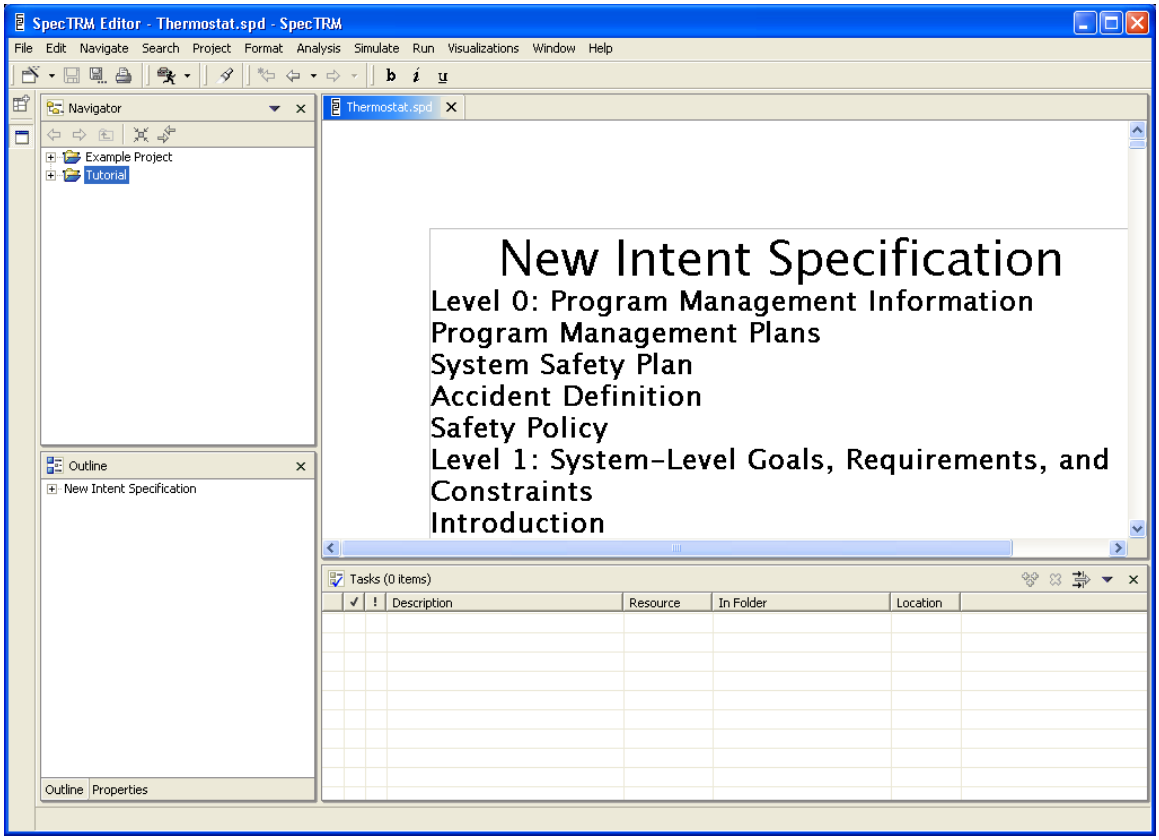

Figure 7 - New Intent Specification

Notice that the type right portion of the perspective is now taken up by the specification editor. In the bottom left, there is an outline of the document. Expanding that outline shows the major sections and subsections of the intent specification (see figure 8, below).

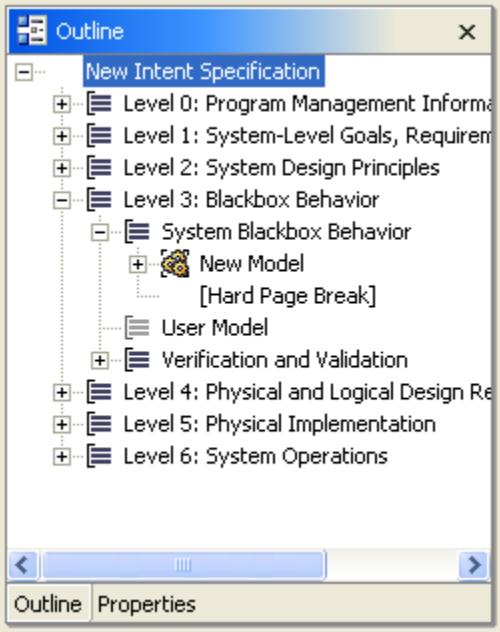

Figure 8 - Outline of a New Specification

Editing the Specification

Once the new file is created and opened, it is possible to edit the specification. To change the name of the specification, place the cursor and type, just as you would with a word processor.

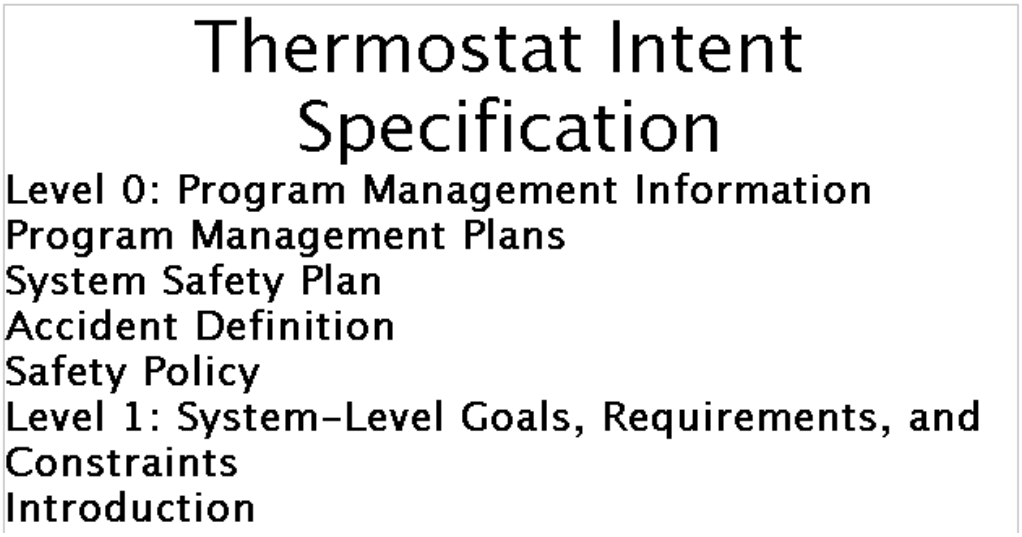

Figure 9 - Editor with Specification Title Changed

Next, we'll add some text to the introduction section of the document.

1. Select the introduction in the outline view.

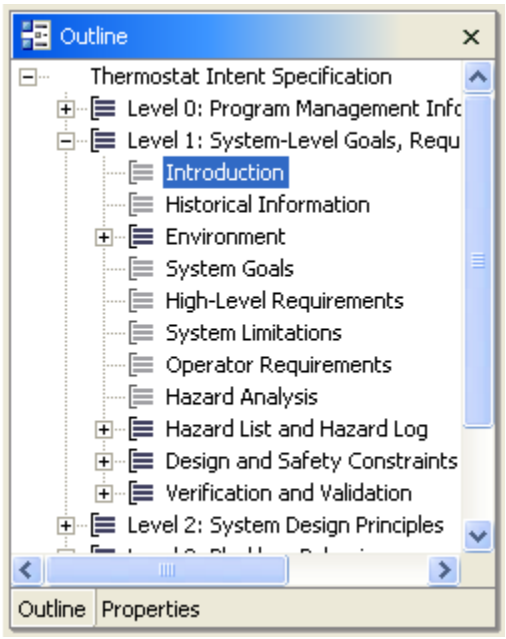

Figure 10 - Selection in Outline View

2. Right-click and select the **New** > **Paragraph** command. The new menu also has options for adding other things to specifications such as new subsections, component models, etc.

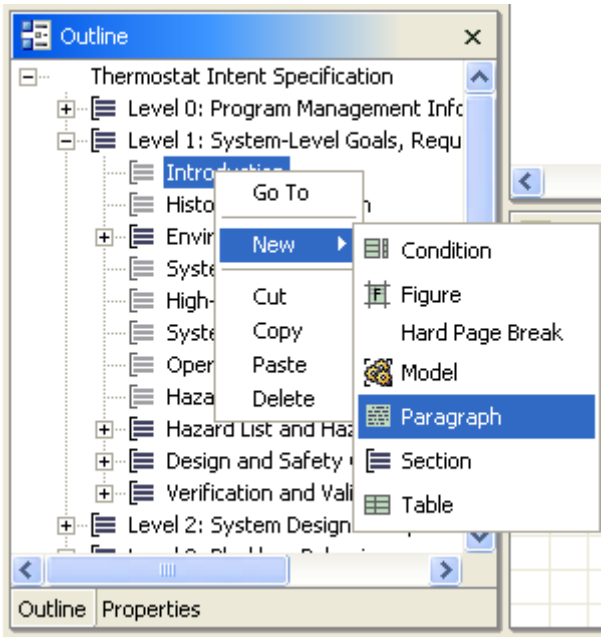

Figure 11 - Add Paragraph Command

3. Now we can add an introduction to the specification, giving a quick overview of the system.

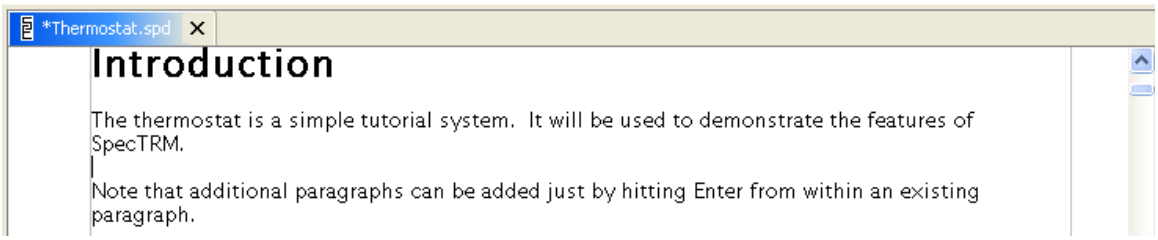

Figure 12 - Text Added to the Specification

Using the same procedure, we can add a system-level goal for the thermostat, shown in figure 13, below. The goal is then refined into a high level requirement.

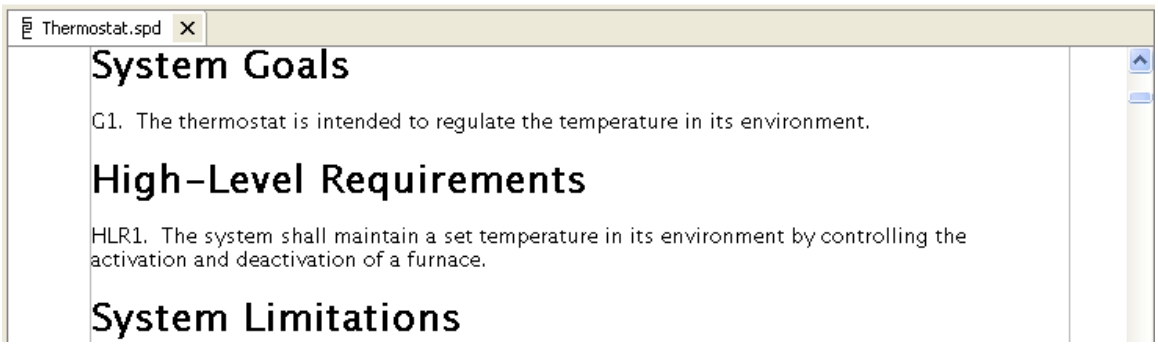

Figure 13 - System Goal and High-Level Requirement

Creating Traceability Links

It is important to create traceability links that preserve the rationale behind design decisions. In this case, the high-level requirement HLR1 is motivated by the system goal G1. This relationship can be recorded using a traceability link.

1. Add the text of the link. In intent specifications, convention is to add links in parentheses at the end of a requirement, constraint, or section. The link consists of the identifier of the thing being linked to, such as G1 for the first goal. Additionally, an arrow is used to indicate where the link points: ↑ for an earlier intent specification level, ↓ for a lower level, ← for earlier in the same level, and → for later in the same level. To add the arrow to the link text, right click in the editor window and use the **Insert Arrow** sub-menu. Figure 14, below, shows an example.

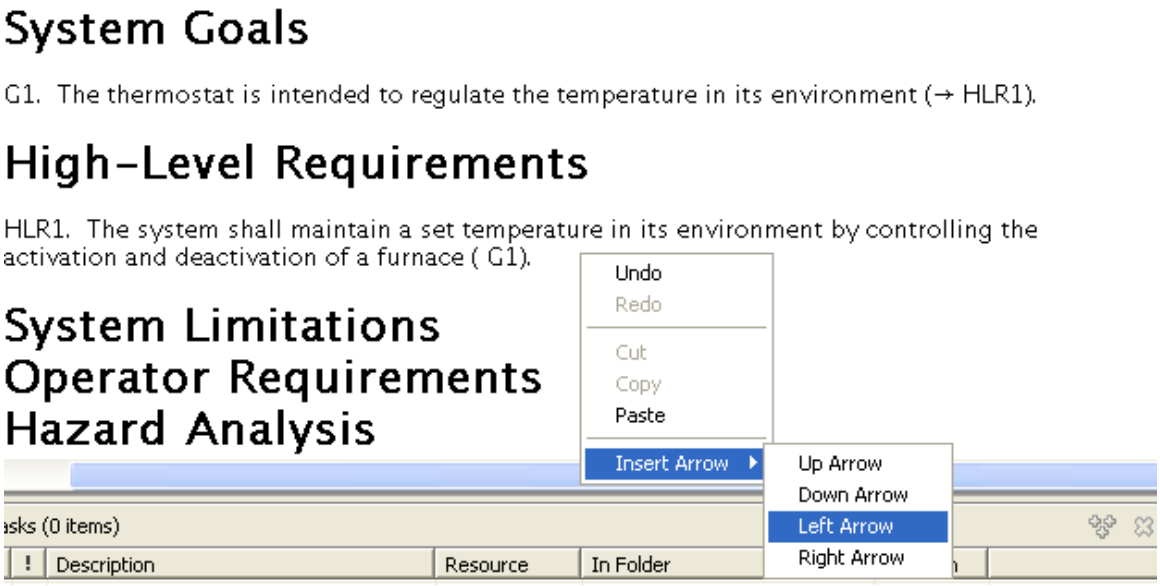

Figure 14 - Adding Link Text and Arrows

2. Highlight the text to hyperlink to another location.

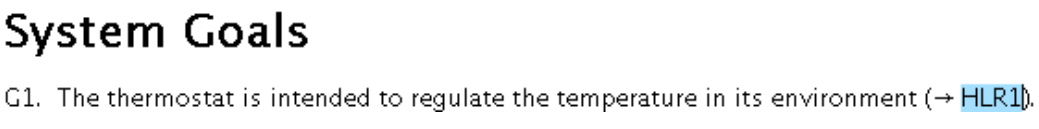

Figure 15 - Highlight Text to Hyperlink

3. Choose the **Format** > **Create Link**… command.

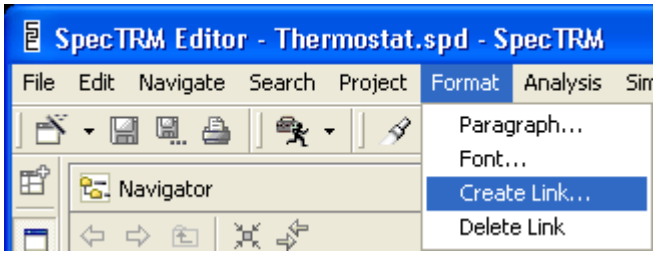

Figure 16 - Create Link Command

4. Select the file containing the destination for the link. In this case, that's the Themostat.spd file. By selecting another file, it is possible to create a link from text in one file to content in another file.

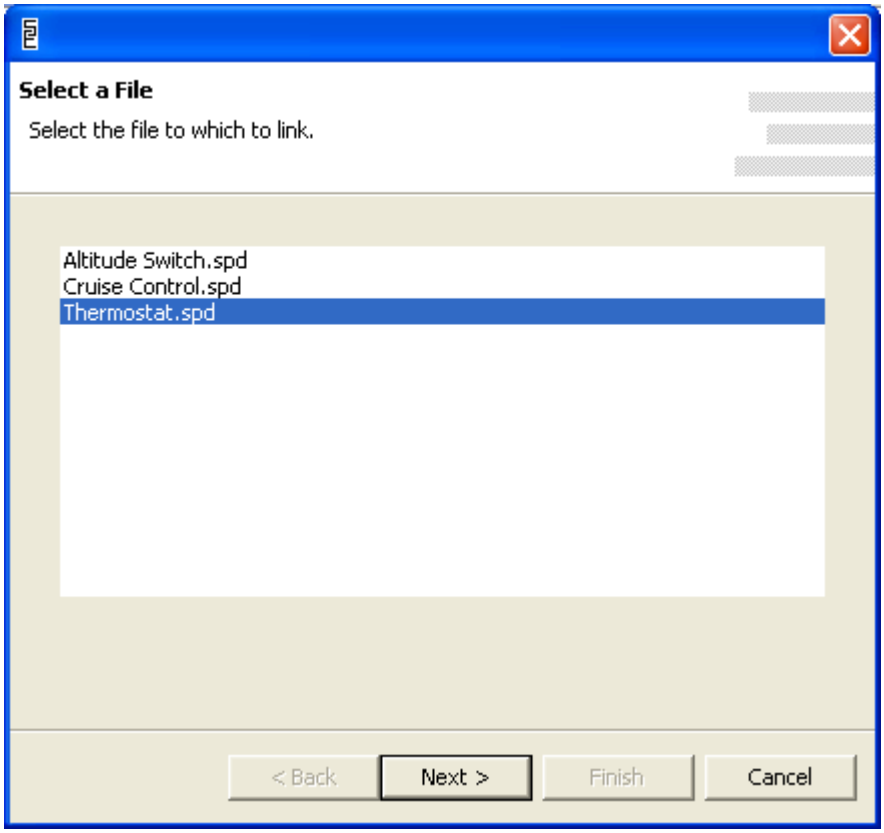

Figure 17 - Select Destination File

5. Select the portion of the document that will be the destination of the link.

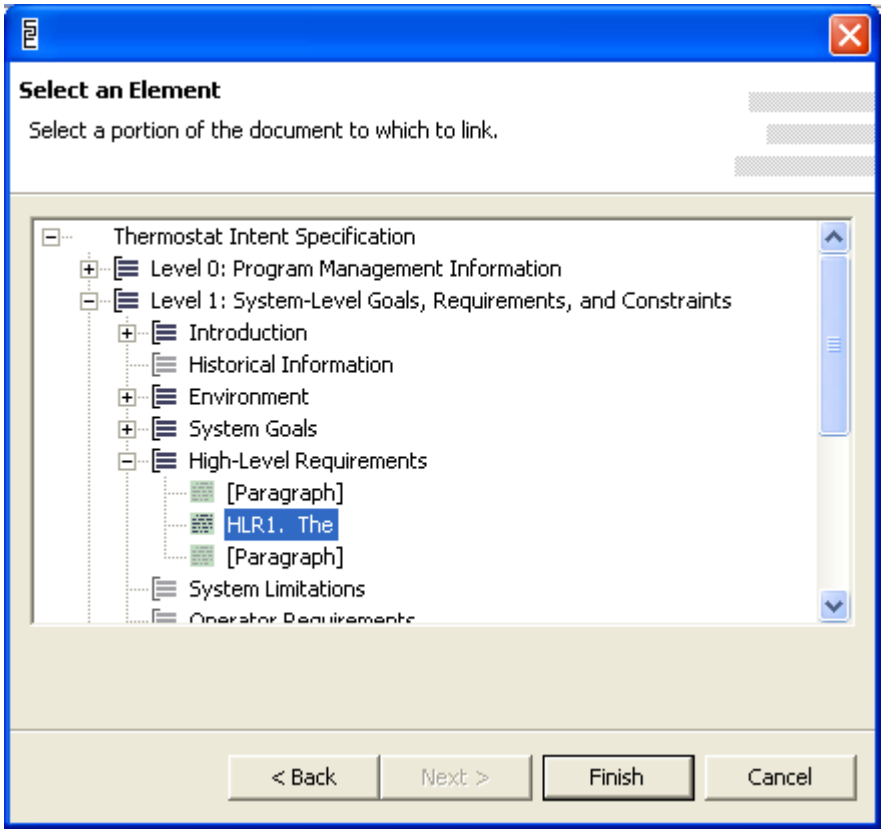

Figure 18 - Select Link Destination

6. Click **Finish**, and SpecTRM will create the link.

System Goals

G1. The thermostat is intended to regulate the temperature in its environment (→ HLR1).

Figure 19 - New Link

To follow the link, click on the blue, underlined text. Now, try creating a link from HLR1 to G1.

Design Information

Next, we add design decisions at level 2 of the intent specification in the "System Design Principles" section. Use the same steps that were used above to add the goal and high-level requirement. When finished, they'll look something like figure 20, below.

Level 2: System Design Principles
System Design Principles

2.1 When the temperature exceeds the set point, the furnace is deactivated († HLR1).

2.2 When the temperature drops below the set point, the furnace will be activated († HLR1).

Figure 20 - Design Principles

SpecTRM-RL Component Requirements Models

SpecTRM-RL (SpecTRM Requirements Language) is a modeling language used in SpecTRM to describe the behavior of system components. The language was developed to describe reactive embedded control system software, but it has been successfully applied to other kinds of components, including hardware. This section shows the development of a very simple SpecTRM-RL model describing the control software for our thermostat example. Figure 21, below, shows the outline view of the model skeleton provided in a new intent specification. The model begins with an empty example of each kind of model element: inputs, outputs, states, modes, macros, and functions.

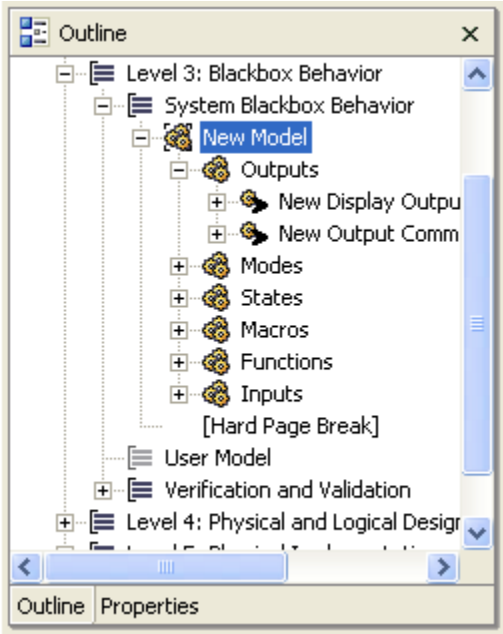

Figure 21 - Empty Model Outline

Scroll down to the model name, “New Model”, and change it to “Thermostat Controller.”

We have found that it’s helpful to begin modeling with the outputs from the component and work backwards, across the logic of the component, to the inputs. For the thermostat controller, we will need two outputs: one to turn on the furnace and one to turn off the furnace. As both of these are output commands, we do not need the display output and can delete it.

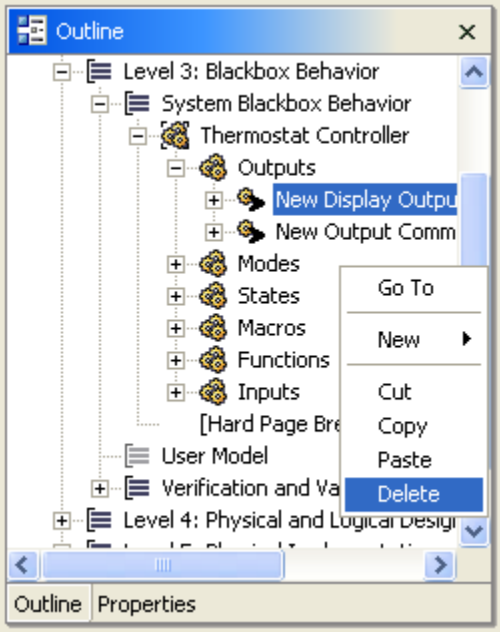

Figure 22 - Deleting Display Output

1. Go to the output command.
2. Change the output name to Activate Furnace, as shown in figure 23, below.

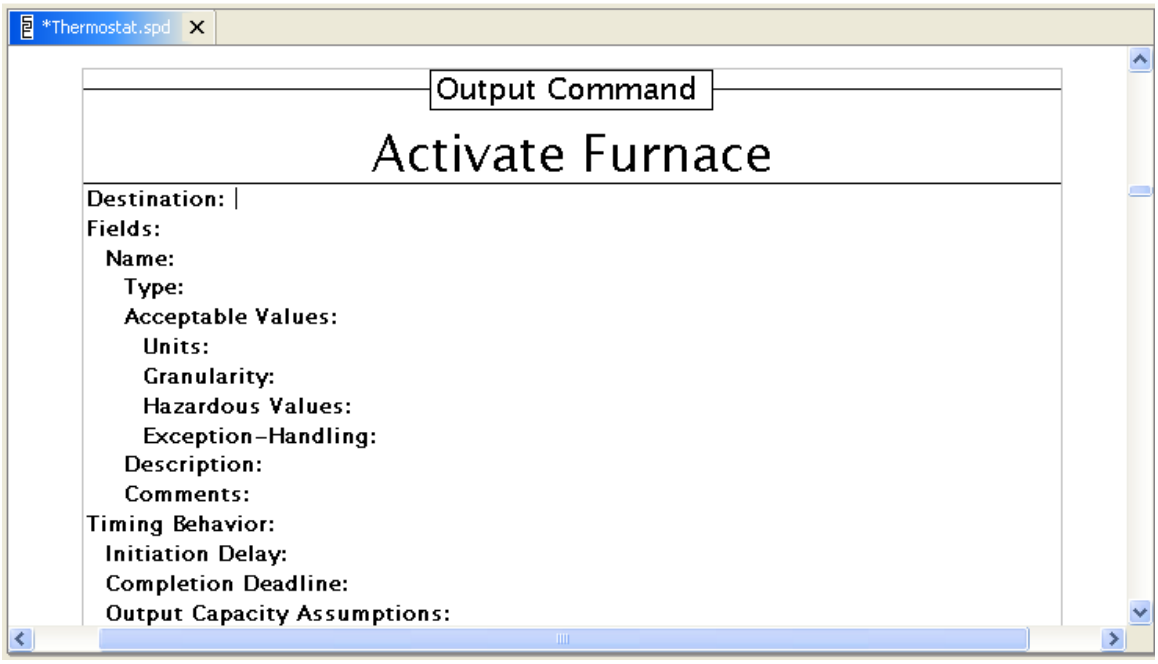

Figure 23 - Change Output Command Name

Beneath the command name are a number of attribute-value pairs. Many of these attributes address requirements completeness. Accidents are often caused by incompleteness in requirements. Consider the timing behavior cluster of attributes in the

output command attributes. During the Three Mile Island accident, reactor status information was being sent to a line printer. So much information was generated that the line printer was ran up to three hours behind the state of the reactor, leading the operators to act on out of date information. The output capacity assumptions section of the output template reminds the engineers developing the component requirements to consider these issues. If no information is available, the attribute can be left blank, and the blank attribute provides a salient reminder that information is missing.

3. The Destination attribute names the component to which the output is sent. Give the **Destination** attribute a value of **Furnace**.
4. The **Fields** attribute block lists the fields in the output message. The command to activate the furnace has only one field.
    a. Give the **Name** attribute a value of **Command**.
    b. Give the **Type** attribute a value of **{Activate, Deactivate}**. The **Type** attribute describes what values can appear in the field. Outputs may be integers, real numbers, or values drawn from a finite set of choices (called enumerated values). Enumerated values are written as a comma-separated list between two braces.
    c. Give the Acceptable Values attribute a value of **{Activate}**. Although the command can be either activate or deactivate, the activate furnace output only ever activates the furnace. A separate command is responsible for deactivating it. The acceptable values attribute is used to indicate when more values are allowed in the type than make sense for this output.
    d. The rest of the attributes describing the field may be left blank for now. Attributes like granularity and units do not make sense for an enumerated type – they are used with integers and real numbers. The description of the field can be filled in later.
5. The output, as described so far, is shown in figure 24, below.
6. The rest of the attributes on the output can be left blank for this small example. A complete industrial project would complete the template in order to ensure that the component requirements are complete. Notably, the **Description** attribute would contain a hyperlink back up to design decision 2.2, which motivates the need for an output to activate the furnace. Similarly, design decision 2.2 would contain a hyperlink down to 3.Activate Furnace.

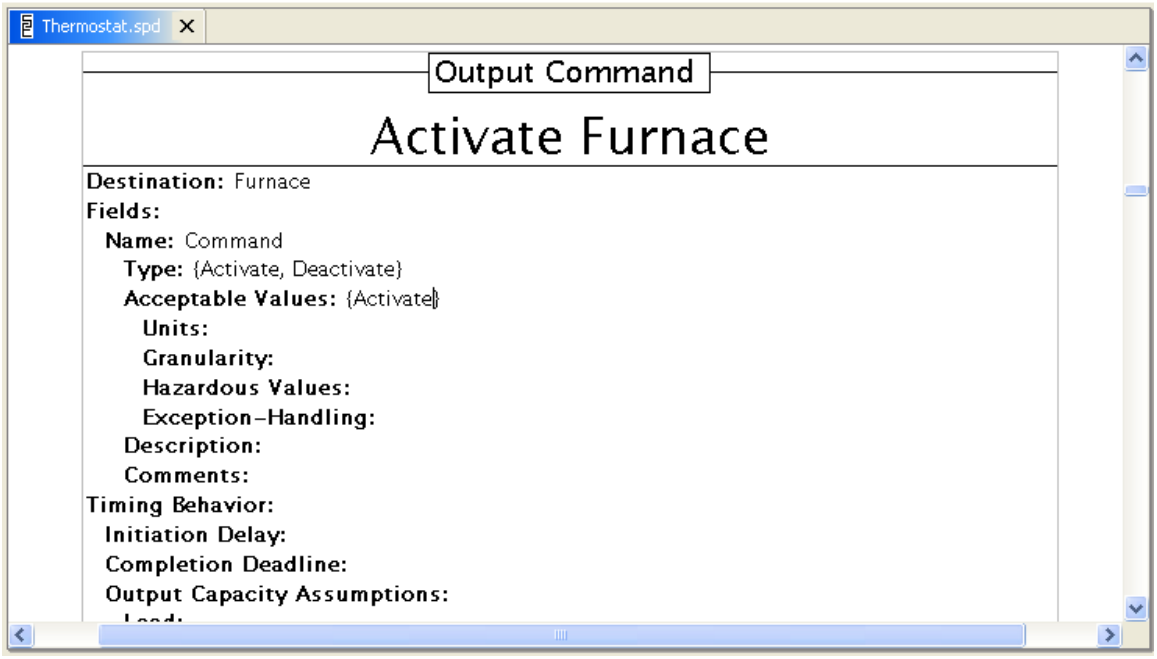

Figure 24 - Output Command Attributes

7. Scroll down to the triggering condition for the output. The triggering condition is an AND/OR table that determines when the output is sent. It starts out empty, as shown in figure 25, below.

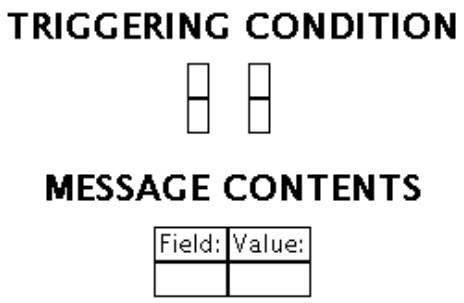

Figure 25 - Empty Triggering Condition and Message Contents Tables

8. According to the system design principles at level two, the furnace will be activated whenever the temperature falls below a set point. The controller must infer the temperature of the room in order to determine whether the room is above, below, or at the set point. We will call this inferred state the room temperature and write the table as shown below, in figure 26.

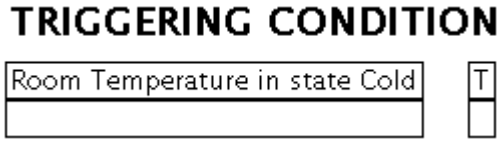

Figure 26 - Partial Triggering Condition

9. However, the requirements for this output are not quite finished. The output to activate the furnace should only be sent when the room initially becomes to cold. As written in figure 26 above, the command to activate the furnace would be sent repeatedly until the room warmed up, even if the furnace was already active. The completed triggering condition is shown in figure 27, below. (For a more complete discussion of SpecTRM-RL's syntax, consult the online user manual and the two example projects included with the software.)

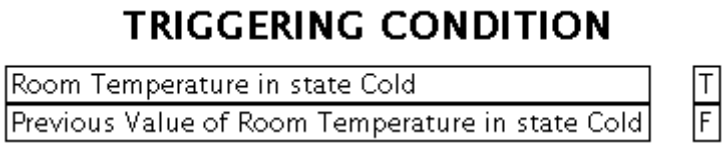

Figure 27 - Output Triggering Condition

10. The last step in writing the output is to describe the contents of the output message. The message contents for this output are simple: the command field with an activate value, as shown in figure 28, below.

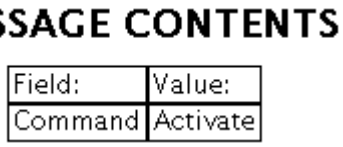

Figure 28 - Output Message Contents

The next step is to add an output command that deactivates the furnace when the room is no longer cold.

1. To add an output command, right-click on the Outputs group and choose **New** > **Output Command**.

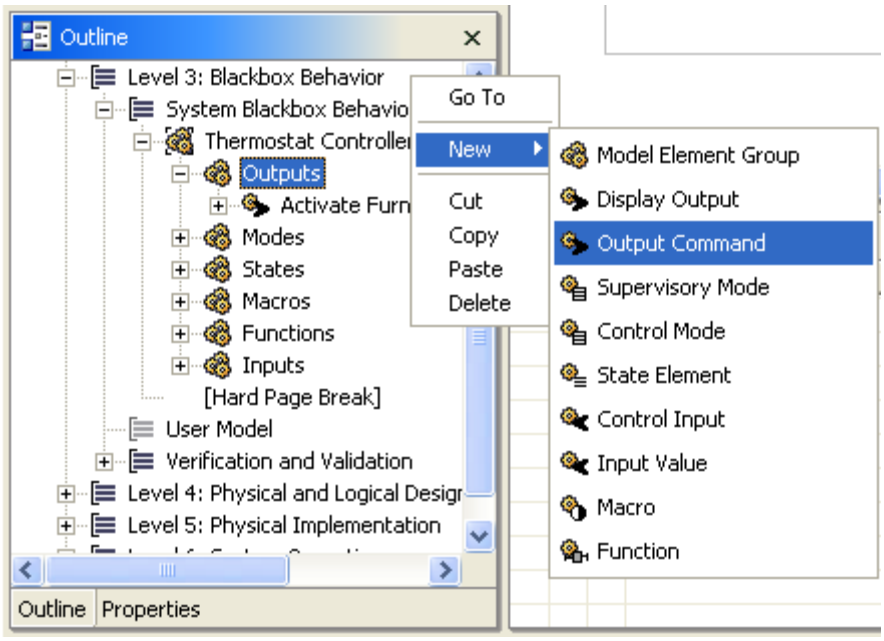

Figure 29 - Adding an Output Command

2. Now fill in the output command, using the Activate Furnace command as a guide. The results of filling in the name and attributes should look something like figure 30, below.

Output Command

Deactivate Furnace

Destination: Furnace
Fields:
Name: Command
Type: {Activate, Deactivate}
Acceptable Values: {Dectivate}
Units:
Granularity:
Hazardous Values:
Exception-Handling:
Description:
Comments:

Figure 30 - Deactivate Furnace Output Attributes

3. The triggering condition and message contents should look something like figure 31, below.

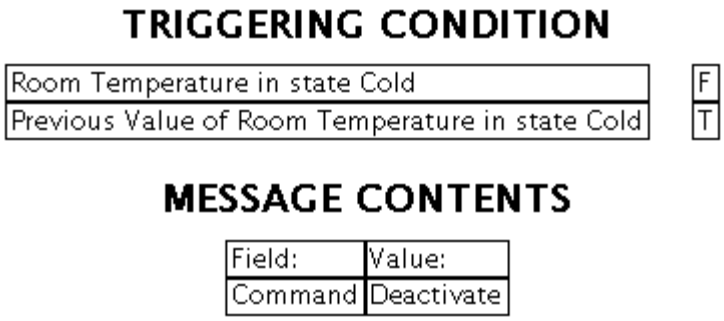

Figure 31 - Deactivate Furnace Triggering Condition and Message Contents

Next, delete the modes group, as the behavior of the thermostat is too simple to require division into control, operating, or supervisory modes.

The output refers to a state called "Room Temperature," so we'll turn the empty state element already in the model into a state that tracks the temperature of the room.

1. Edit the name of the state to read **Room Temperature**.
2. The **Room Temperature** state element can transition to several state values. All states are required to have an **Unknown** state. The component transitions to the unknown state whenever it is not possible to determine the temperature of the room. This could happen, for example, if the temperature probe does not report data to the software controller. In addition to the unknown state, we referred to the **Cold** state in an output triggering condition above. It would make sense to have **At Set Point** and **Hot** transitions as well. First, we fill out the AND/OR table for the **Unknown** state. This AND/OR table describes the conditions that must be true in order to transition to the unknown state.

   We know that determining whether the room is too hot, too cold, or at the set point is going to rely on a measurement of the room's temperature and the set point desired by the user. If either of these inputs are Obsolete, meaning that the data is too old to be relied upon or was never received in the first place, then the temperature of the room cannot be determined. To enter this data, it will be necessary to add a column and a row to the AND/OR table. To add or remove a row or column, right-click inside the cell to delete or after which to add, then select the command, as shown in figure 32, below.

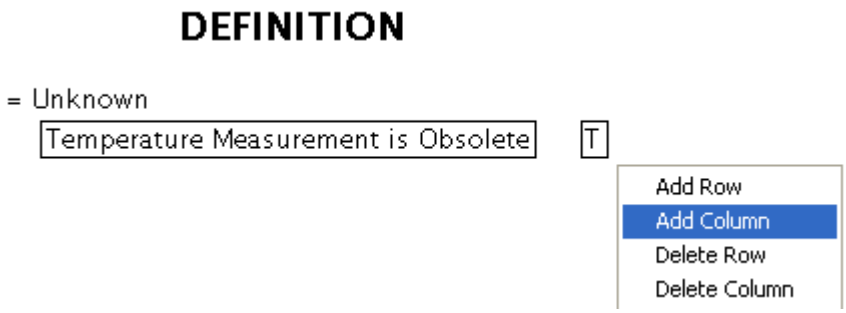

Figure 32 - Adding a Column to an AND/OR Table

3. When finished, the table should look like figure 33, below.

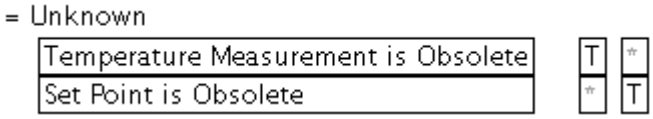

Figure 33 - AND/OR Table for the Unknown Transition

4. Now, we add three more transition conditions: **Cold**, **At Set Point**, and **Hot**. This is done by right clicking on the state element in the outline view and selecting the **New** > **Transition Condition** command.

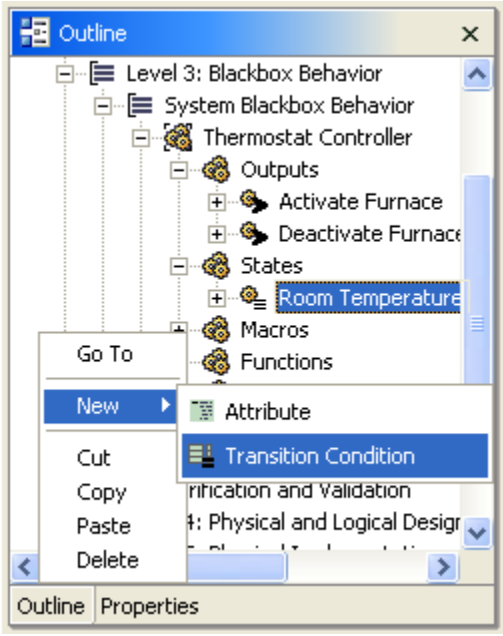

Figure 34 - Adding a Transition Condition to a State Element

5. When edited, the AND/OR tables will look something like those in figure 35, below.

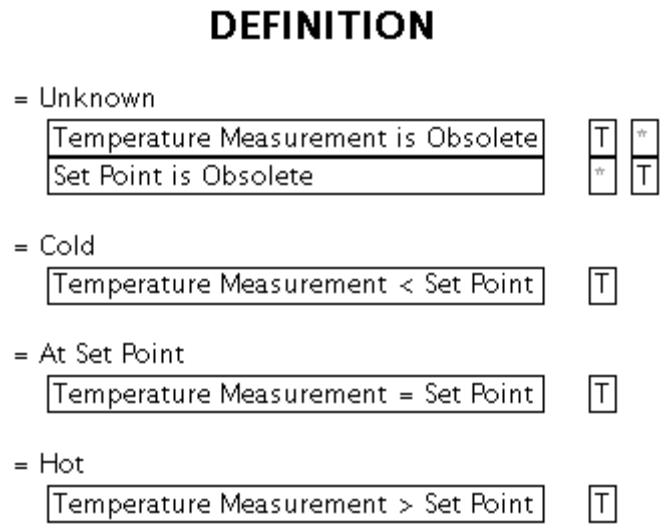

Figure 35 - AND/OR Tables for the Room Temperature State

Simply delete the macro and function model element groups. This simple thermostat component does not use macros or functions. At this point, most of the logic that governs the workings of the thermostat has been decided. All that remains is the creation of inputs to the component. In the process of working backward from the outputs, we've identified two inputs necessary to accomplish the component's function: a temperature set point and a measurement of the current temperature.

The set point input is a control input; it comes from a user to direct the operations of the thermostat. We can create the set point input by editing the empty control input skeleton at the top of the inputs group.

1. Change the name of the input to **Set Point**.
2. For the **Source** attribute, enter **Thermostat Controls**.
3. The input has a **Type** of **Integer**.
4. The **Units** are degrees **Fahrenheit**, and the **Granularity** is **1 degree**. The **Exception Handling** is **None**.
5. Put together, the attributes describing the input look like figure 36, below. Again, a real specification would involve determining values for the completeness attributes on the input, but we omit them for this simple example.

Control Input

Set Point

Source: Thermostat Controls
Type: Integer
Possible Values (Expected Range): Any
Units: Fahrenheit
Granularity: 1 degree
Exception-Handling: None

Figure 36 - Attributes for Set Point Input

Default definition tables are suggested by SpecTRM because many inputs behave similarly. These defaults assume that after some amount of time, the input's value will become obsolete. This obsolete value makes it easy to specifically identify how out of date data is handled, an important consideration in real-time systems. Thermostat set points, however, are often set by a user and left in place for some time. The edited tables look like the following:

DEFINITION

= New Data for Set Point	
Set Point was Received	T
= Previous Value of Set Point	
Set Point was Received	F
Set Point was Never Received	F
= Obsolete	
Set Point was Never Received	T

Figure 37 - Definition Tables for Set Point Input

The temperature measurement input is an input value; it comes from a measurement of the controlled process. We can create the temperature measurement input by editing the empty input value skeleton.

6. Change the name of the input value to **Temperature Measurement**.
7. For the **Source** attribute, enter **Temperature Probe**.
8. The input has a **Type** of **Integer**.
9. The **Units** are degrees **Fahrenheit**, and the **Granularity** is **1 degree**. The **Exception Handling** is **None**.
10. Put together, the attributes describing the input look like figure 38, below. Again, a real specification would involve determining values for the completeness attributes on the input, but we omit them for this simple example.

Input Value

Temperature Measurement

Source: Temperature Probe
Type: Integer
Possible Values (Expected Range): Any
Units: Degrees Fahrenheit
Granularity: 1 degree
Exception-Handling: None

Figure 38 - Attributes for Temperature Measurement Input

Default definition tables are suggested by SpecTRM because many inputs behave similarly. These defaults assume that after some amount of time, the input's value will become obsolete. This obsolete value makes it easy to specifically identify how out of date data is handled, an important consideration in real-time systems. For the temperature measurement, the finished tables look like the following.

DEFINITION

= New Data for Temperature Measurement

Temperature Measurement was Received	T

= Previous Value of Temperature Measurement

Temperature Measurement was Received	F
Time Since Temperature Measurement was Last Received <= 1 seconds	T

= Obsolete

System Start	T	*	*
Temperature Measurement was Never Received	T	T	*
Time Since Temperature Measurement was Last Received > 1 seconds	*	*	T

Figure 39 - Definition Tables for Temperature Measurement Input

Model Validation

Once the model is constructed, it can be executed and analyzed. However, to execute and analyze a model, the syntax of the model must be valid. For example, consider the case in figure 40, below. This is the triggering condition for one of our outputs with an error introduced. The **Room Temperature** name has been mistakenly written as **Temperature**. When the file is saved, the model is automatically validated. Because of the error here, tasks are added to the task view at the bottom of the window, as shown in figure 41, below.

TRIGGERING CONDITION

Temperature in state Cold	T
Previous Value of Room Temperature in state Cold	F

Figure 40 - Erroneous Triggering Condition

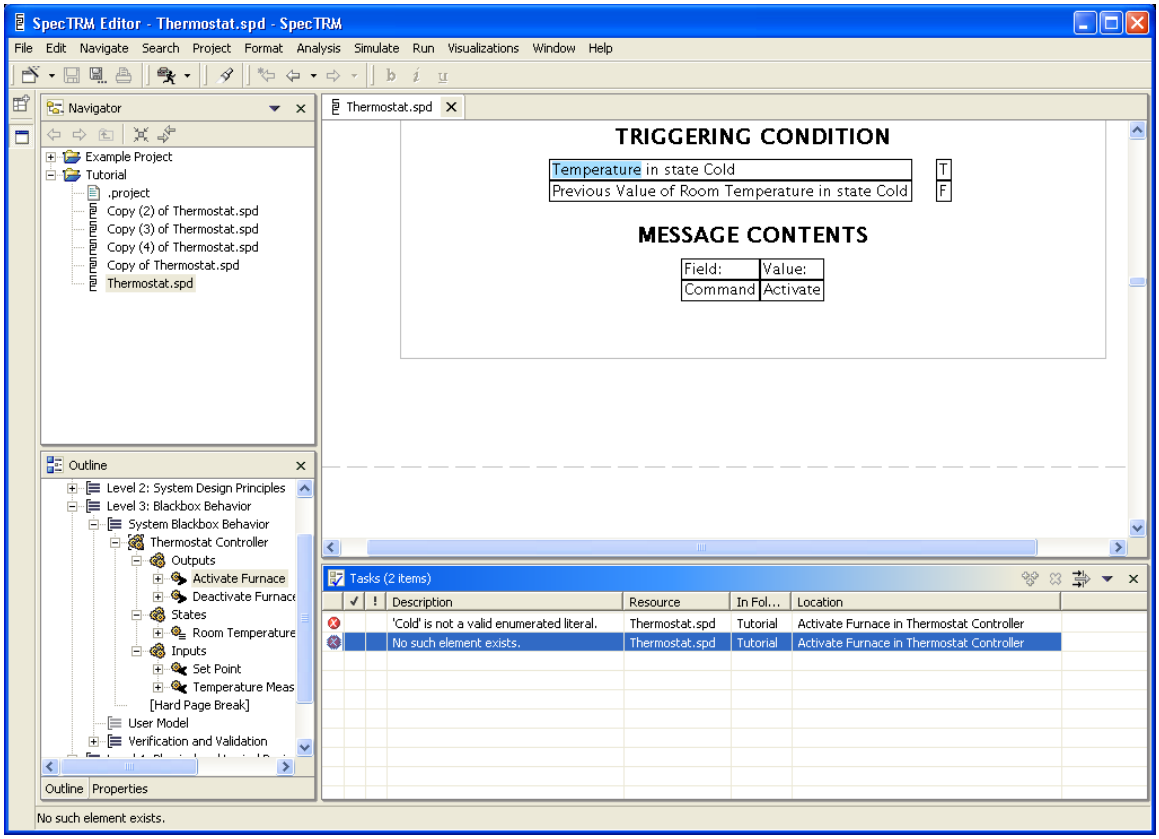

Figure 41 – Task Added for Erroneous Name

Correcting the name and saving the file removes the error notice from the task view.

Sometimes it is desirable to remove validation error tasks from the task list without having to make changes to a model. This is often the case when working on a partially complete model – it isn't intended to be finished yet and errors are expected. In this case, it is possible to clear the validation errors related to a model from the task list.

1. Right-click on the model in the outline view.
2. Choose the **Clear Validation Tasks** command.

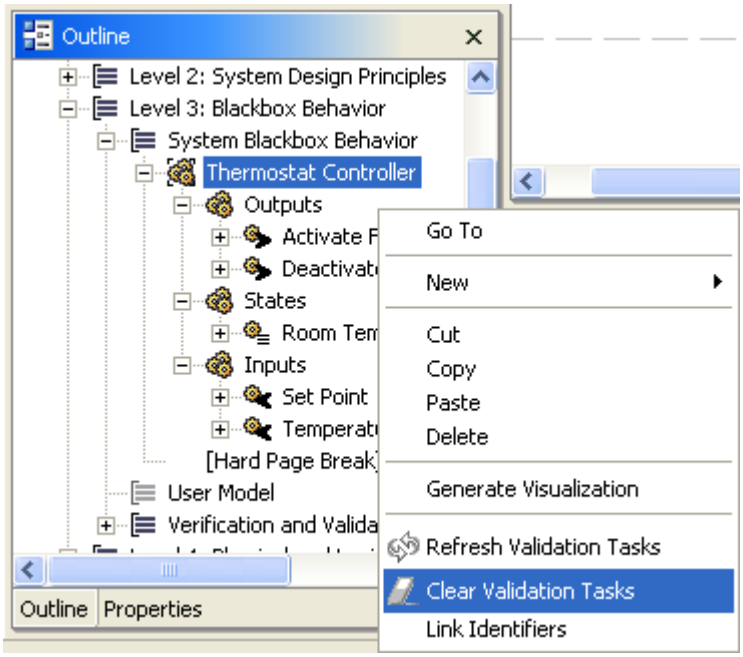

Figure 42 - Clear Validation Tasks Command

Alternatively, you can right click on a specification (.spd) file in the resource navigator, right-click, and select the **Clear Validation Tasks** command from that menu. This will clear all validation tasks related to that file.

Simulation

Once the model is complete, simulation makes it possible to see how the component will behave before the costly steps of design and implementation. SpecTRM simulates the behavior of the component by executing the requirements specification. This section describes how to set up and run a simulation of a SpecTRM-RL model. First, we need a valid model, which we've developed in the sections above. Next, we need a visualization to observe during the simulation.

1. Highlight the model to be simulated in the outline view of the specification file, right-click, and choose the **Generate Visualization** command.

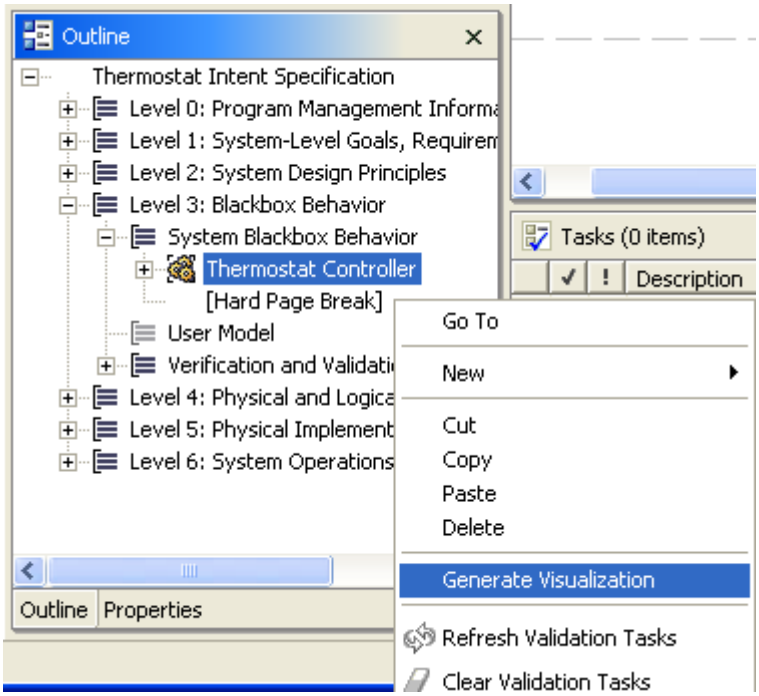

Figure 43 - Generating Visualization

2. Provide a location and file name for the visualization file.

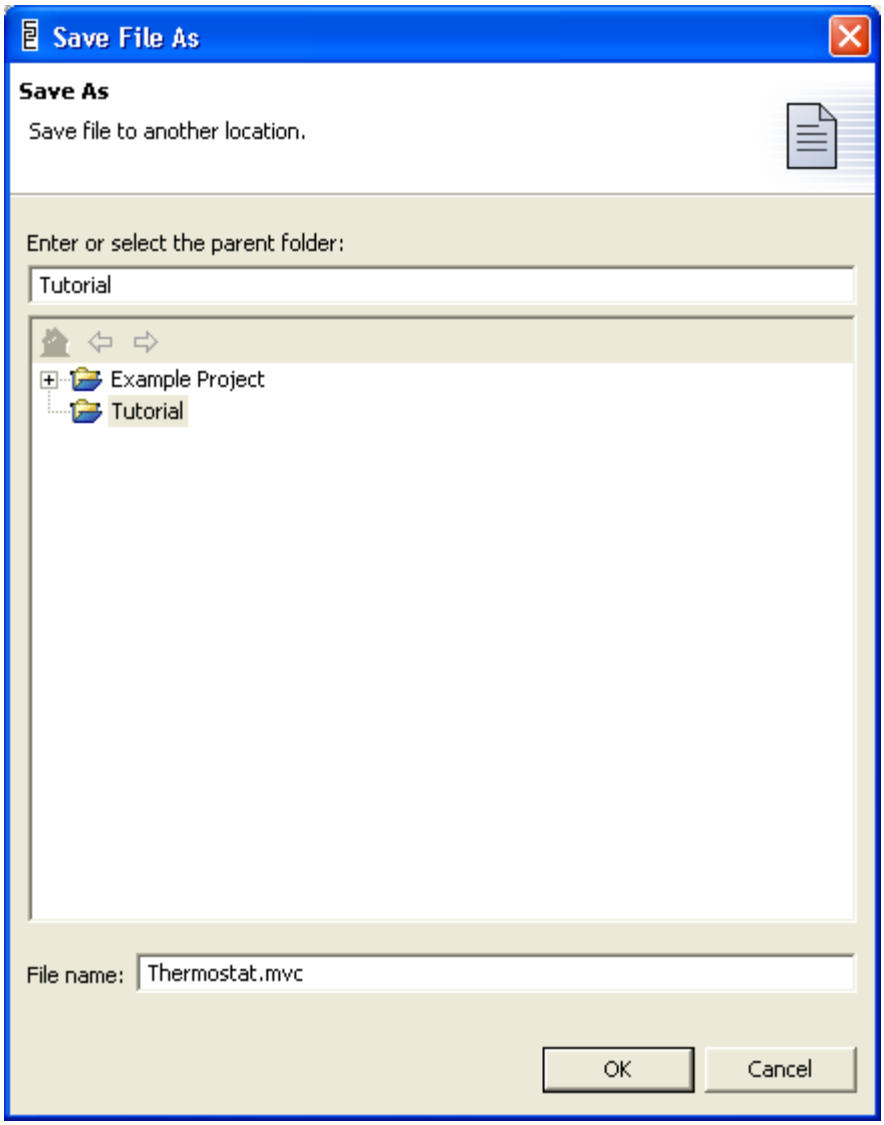

Figure 44 - Supply Name for Visualization File

3. Double-clicking on the .mvc file in the resource navigator will open an editor for the visualization, as shown in figure 45, below. The editor can be used to refine the layout of the visualization. For this example, however, no changes are necessary.

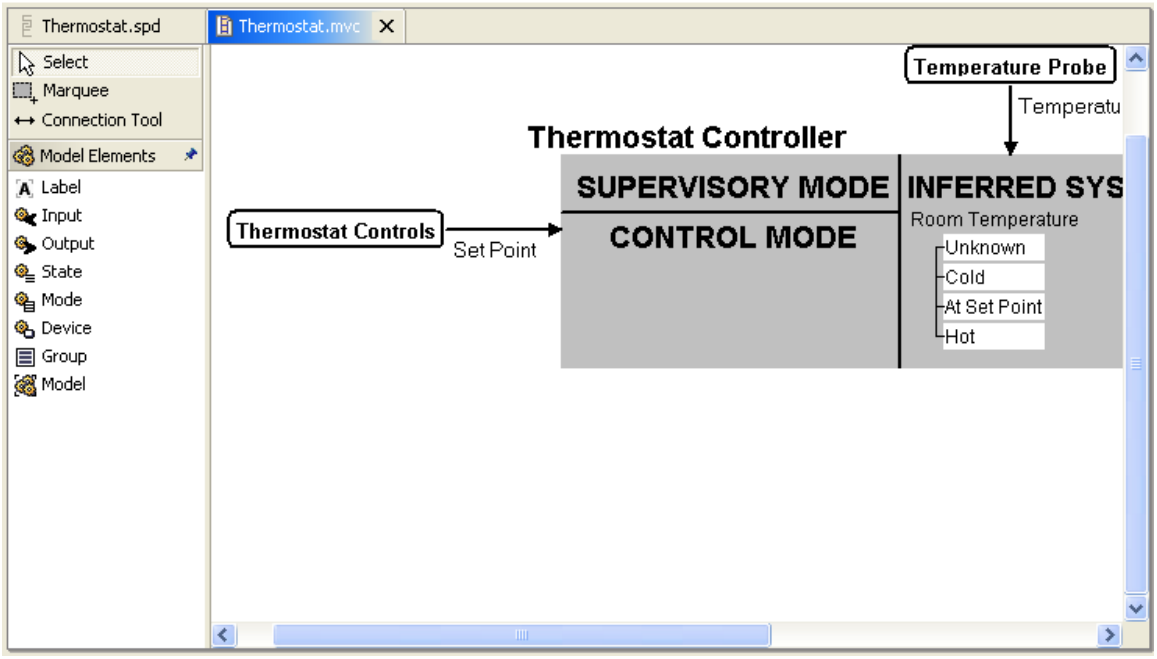

Figure 45 - Visualization Editor

The thermostat simulation will be fed data from text files, which must be created next.

1. First, create a new folder for the simulation data by right clicking on the Tutorial project and choosing **New** > **Folder**.

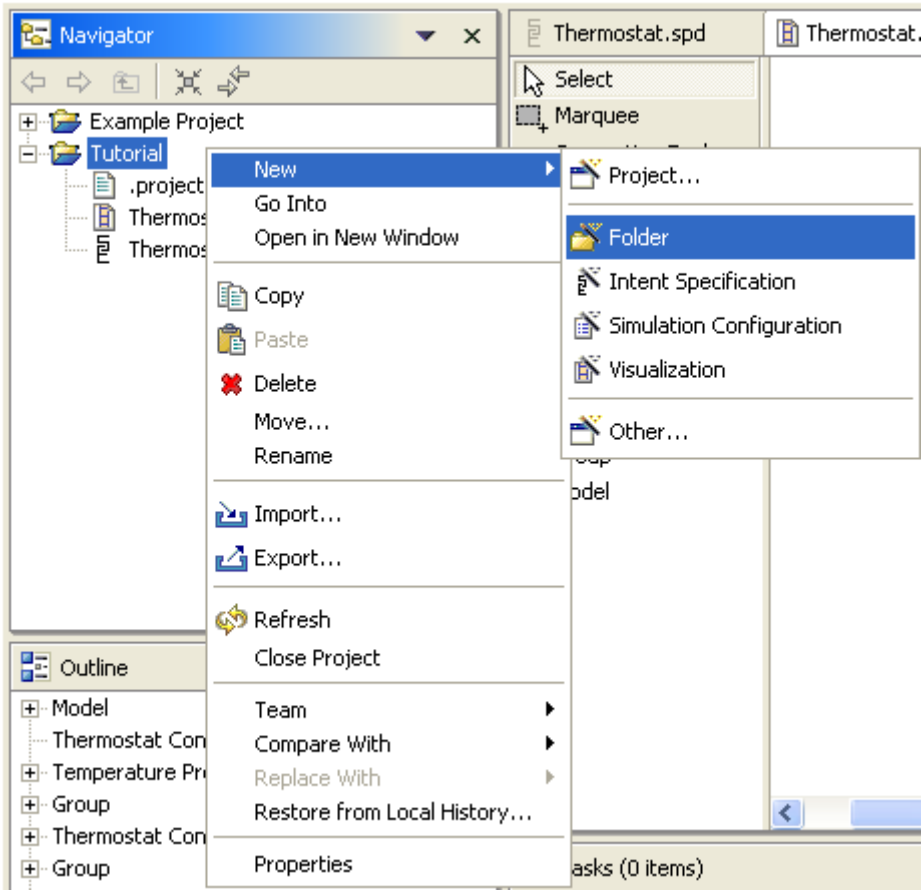

Figure 46 - New Folder Command

2. Provide a name for the folder of simulation data files.

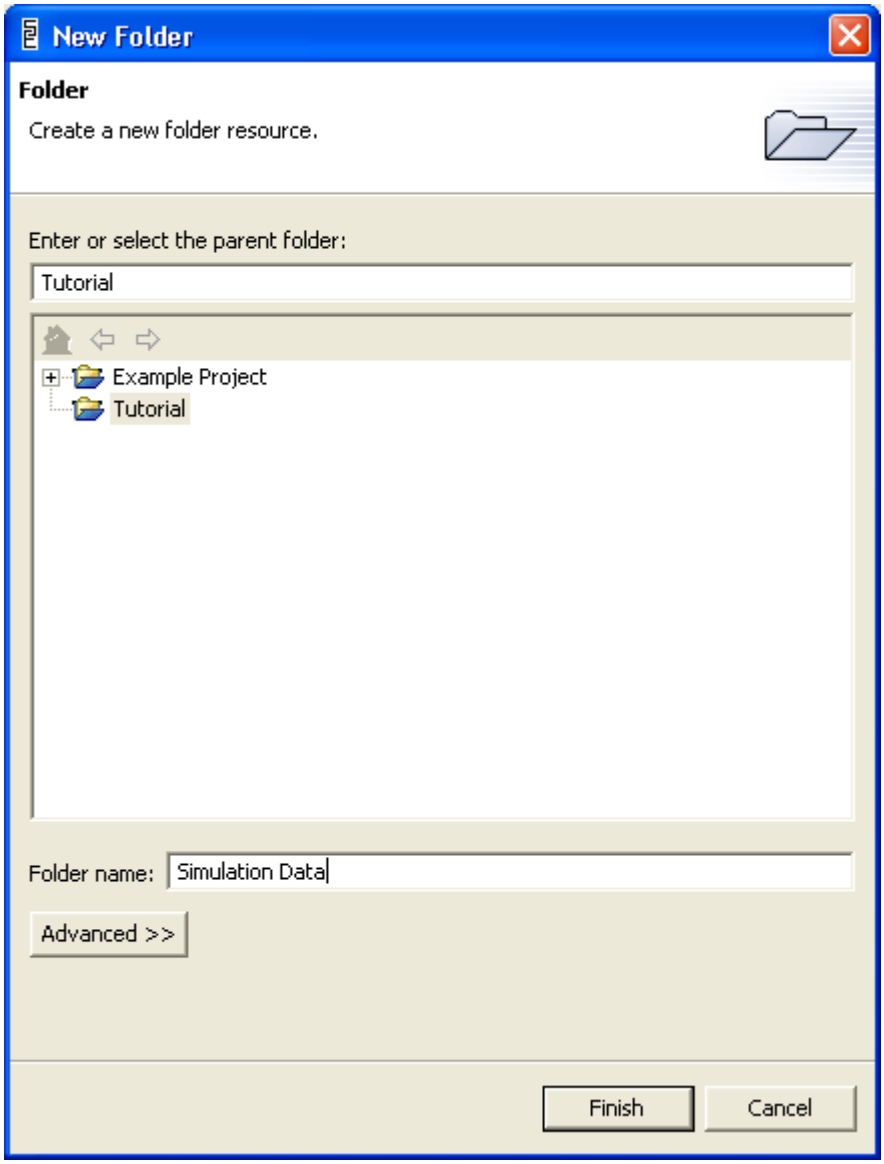

Figure 47 - Simulation Data Directory Creation Dialog

3. SpecTRM creates the folder underneath the Tutorial project.

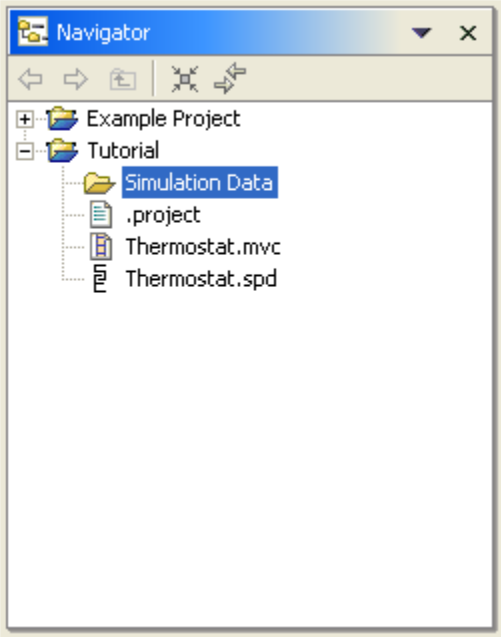

Figure 48 - Simulation Data Folder

4. Now create a text data file for the set point input. This is done with by right clicking on the folder and choosing the **New** > **Other…** command.

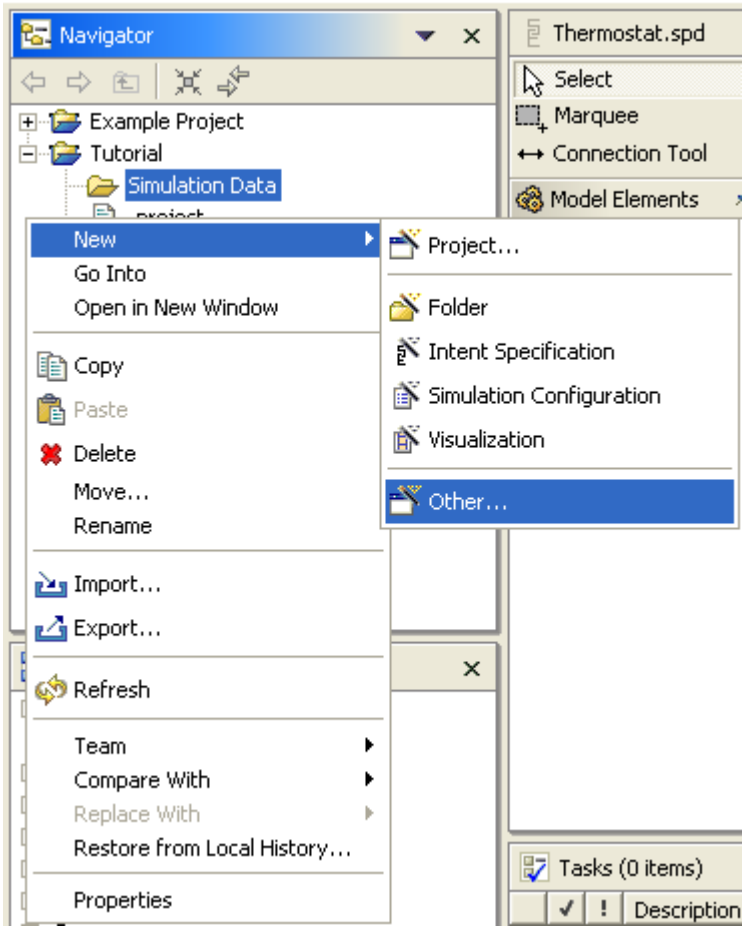

Figure 49 - New File Command

5. The file is created as a simple file, shown in figure 50, below.

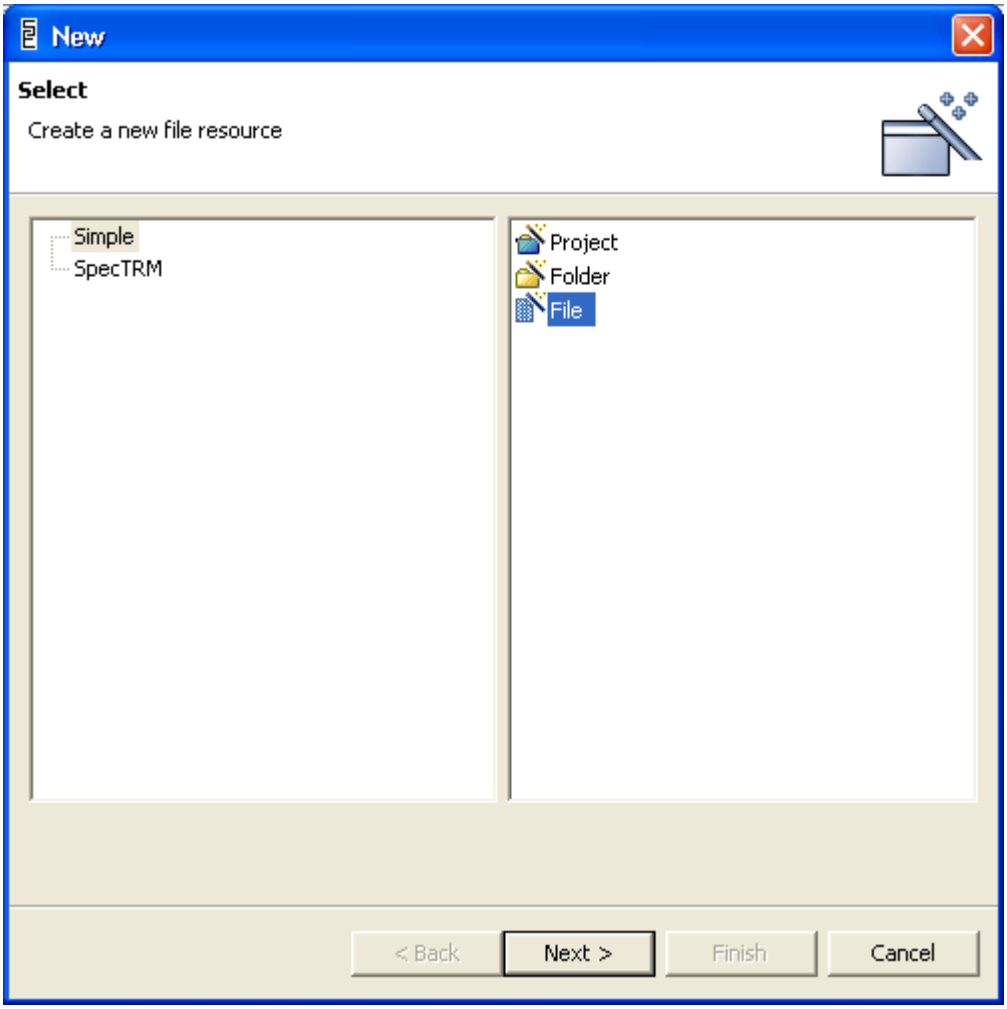

Figure 50 - Simple File

6. Give the file a name of Set Point Data.csv. When you click finish, the file will automatically be opened. If your computer has Microsoft Office installed, the file will be opened in Microsoft Excel. Put times in the first column and data in subsequent columns, as shown in figure 51, below. The data file below says that at 1 second into the simulation, the user inputs a set point of 70 degrees Fahrenheit.

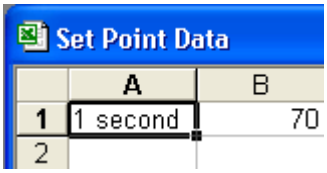

Figure 51 - Simulation Data File

7. When saving the file, you must save the file as a comma separated value (CSV) file, as shown in figure 52, below. Excel will warn that information may be lost by saving to CSV. Confirm that you wish to save the file as a CSV file.

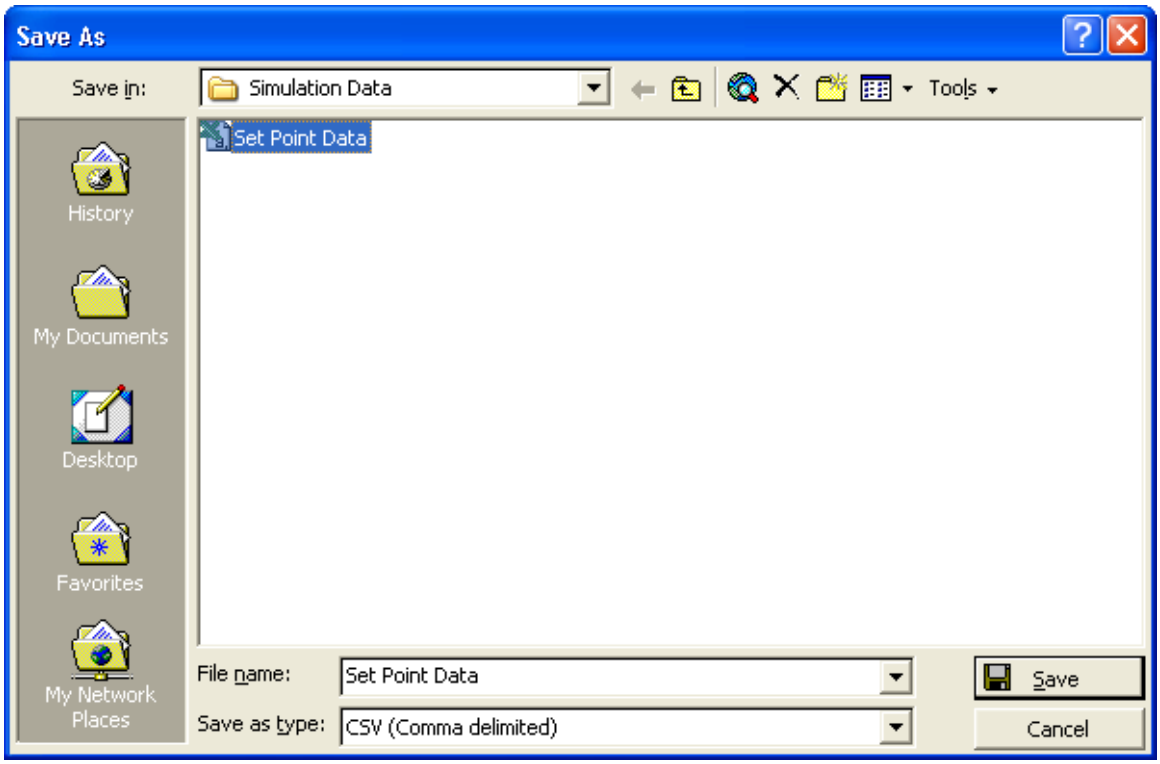

Figure 52 - Save as a CSV file

Now use the same procedure to create an input file for the temperature reading called **Temperature Measurement.csv**. The only difference is that the file of temperature readings will have more data in it. Create a file that looks like the following:

Temperature Reading

	A	B	C
1	500 milliseconds	72	
2	1000 milliseconds	71	
3	1500 milliseconds	70	
4	2000 milliseconds	69	
5	2500 milliseconds	68	
6	3000 milliseconds	69	
7	3500 milliseconds	70	
8	4000 milliseconds	71	
9	4500 milliseconds	72	
10	5000 milliseconds	71	
11	5500 milliseconds	70	
12	6000 milliseconds	69	
13	6500 milliseconds	68	
14	7000 milliseconds	69	
15	7500 milliseconds	70	
16	8000 milliseconds	71	
17	8500 milliseconds	72	
18	9000 milliseconds	71	
19	9500 milliseconds	70	
20	10000 milliseconds	69	
21			

Figure 53 - Temperature Measurement Data

8. Lastly, right-click on the Simulation Data folder and select the **Refresh** command. This forces SpecTRM to synchronize with the file system, which has been changed by Excel.

Next, we must create a simulation configuration file that ties together the model, visualization, and data sources.

1. Create a simulation configuration file by right clicking on the tutorial project and selecting **New** > **Simulation Configuration**.

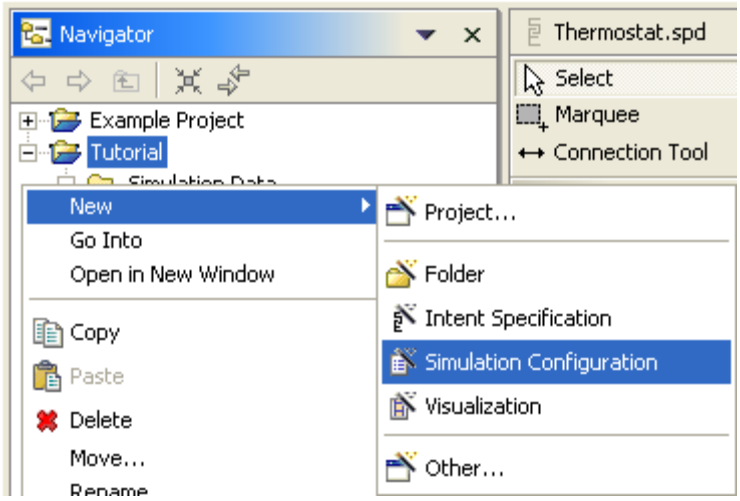

Figure 54 - Command to Create a New Simulation Configuration

2. Provide a name for the simulation configuration file, as shown in figure 55, below.

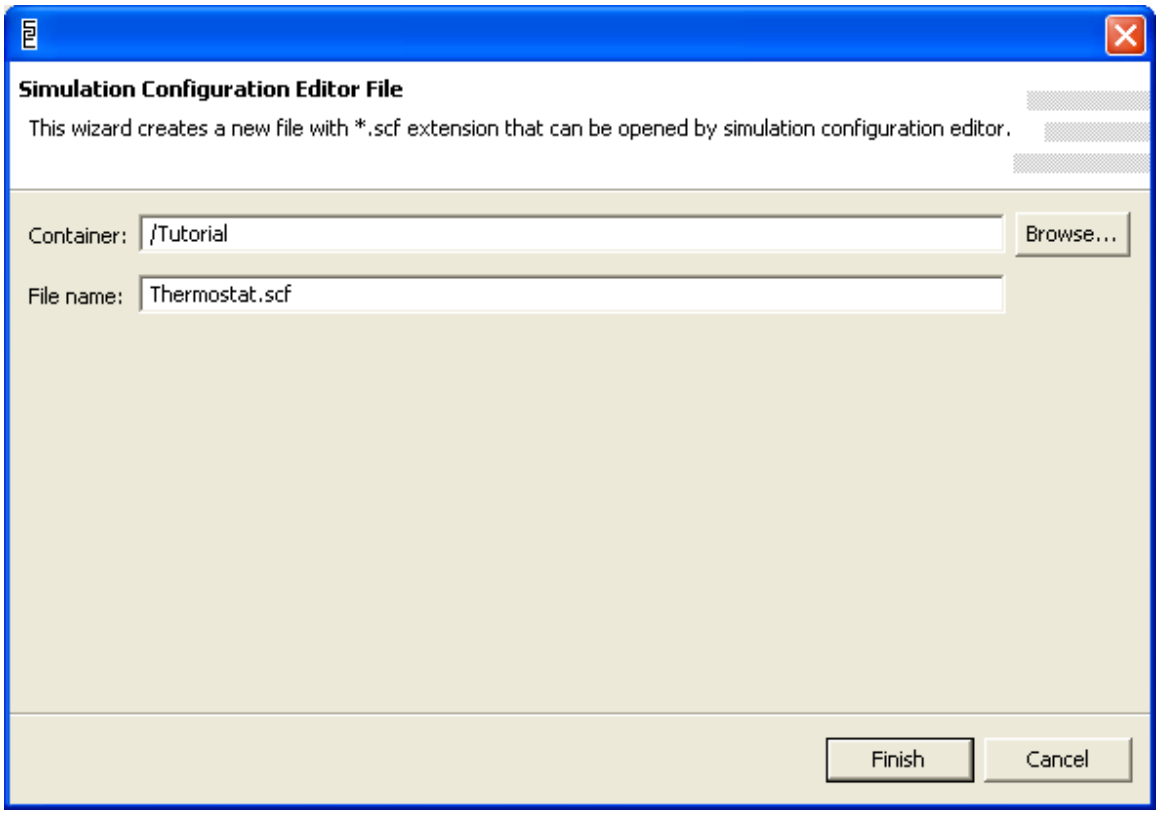

Figure 55 - Choose Name for Simulation Configuration File

3. SpecTRM will open an editor for the new simulation configuration file.

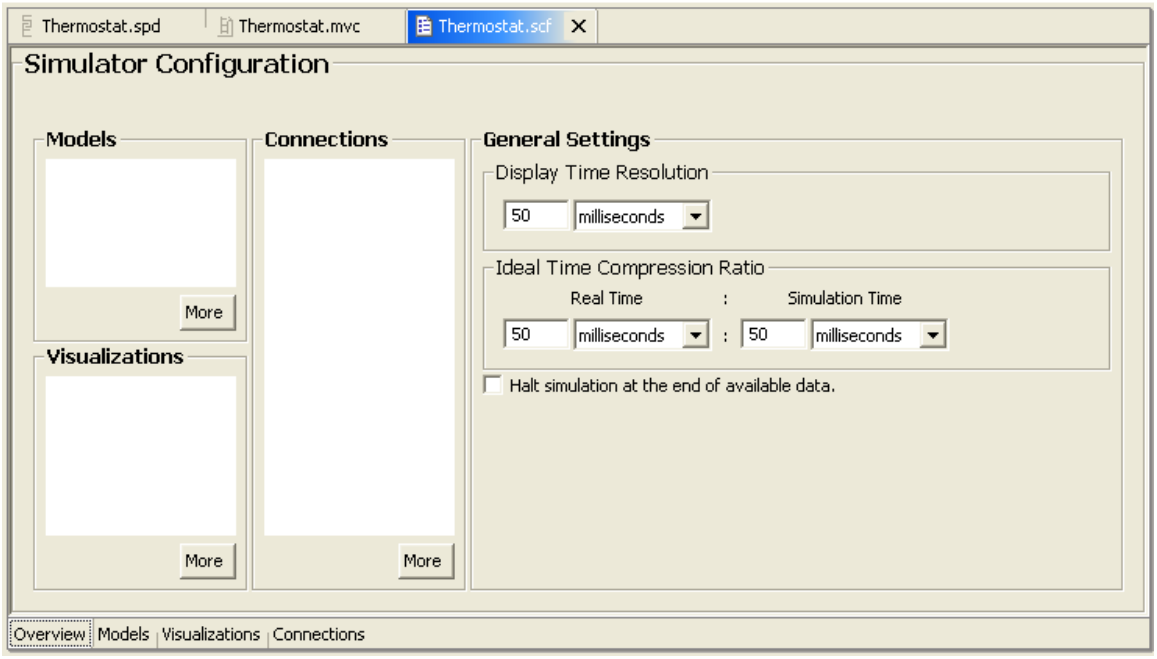

Figure 56 - Simulation Configuration Editor

4. Change the display time resolution, real time, and simulation time to 250 milliseconds. This will cause the simulation to run in real time, updating once every 250 milliseconds.

Next, add the model to the simulation:

1. Click the **Models** tab at the bottom of the editor.
2. Click the **Add Model…** button in the editor page displayed.
3. Choose the thermostat specification file in the list displayed.

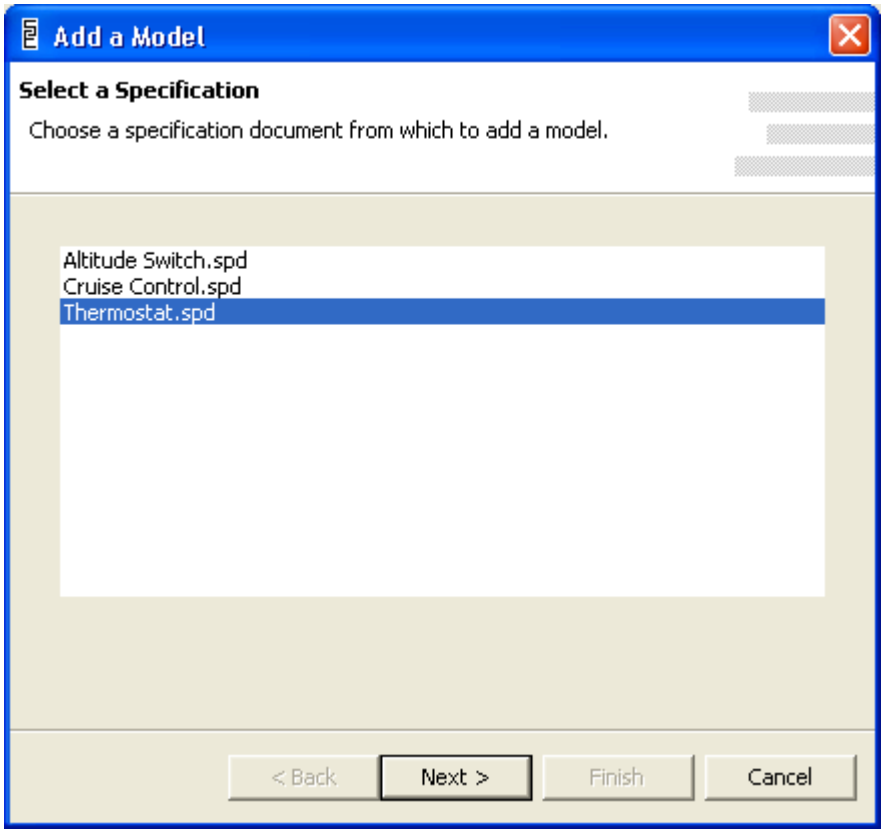

Figure 57 - List of Specification Files

4. Select the Thermostat model from the file.

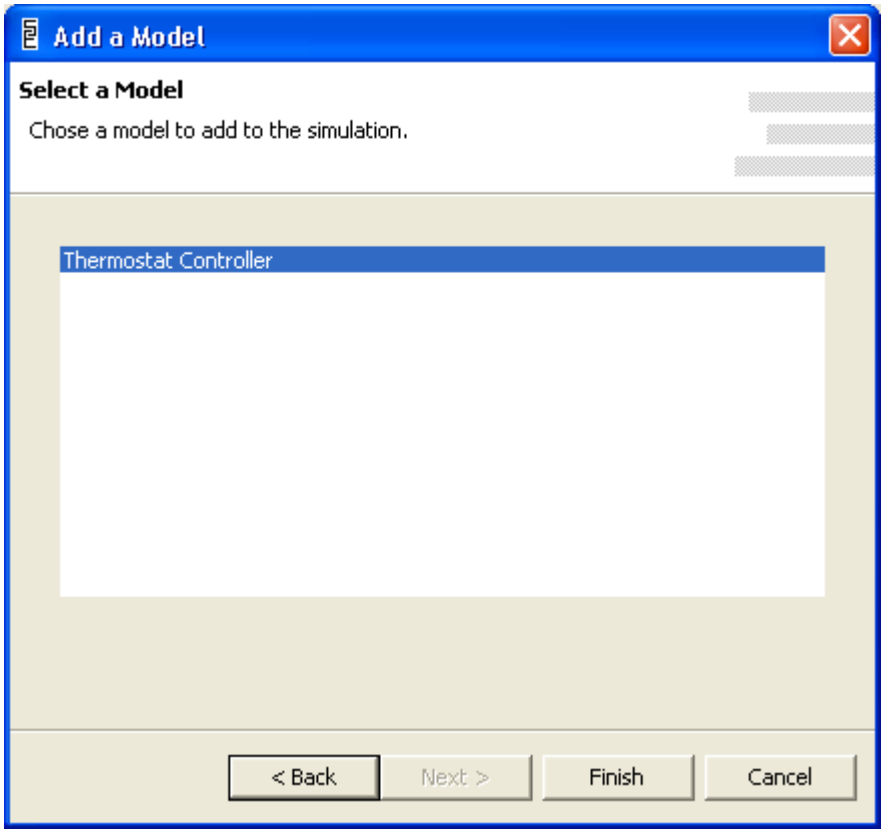

Figure 58 - List of Models in the File

5. Click **Finish**, and SpecTRM will add the model to those in the simulation.

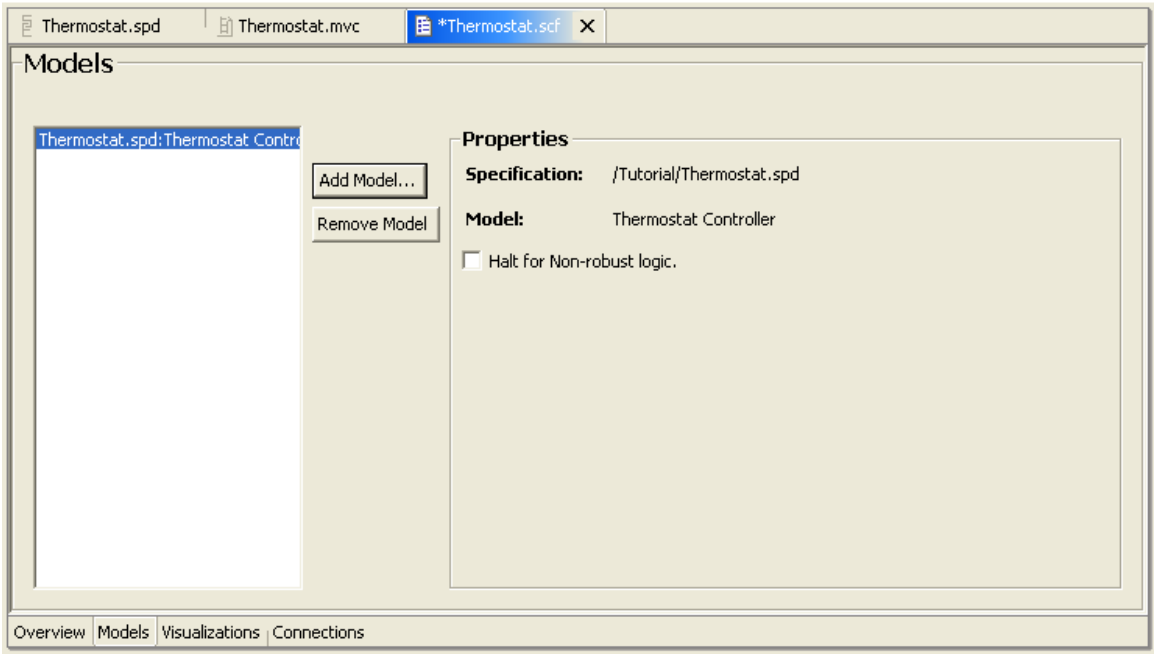

Figure 59 - Simulation Configuration with Model Added

Next, we add the visualization to the configuration. This process is very similar to adding a model. Click on the tab for visualizations, then on the add button. When prompted for the name of the visualization to add, choose the Thermostat.mvc file.

Lastly, the input data files must be connected to the inputs in the model. To do this, follow the steps below:

1. Click on the **Connections** tab.

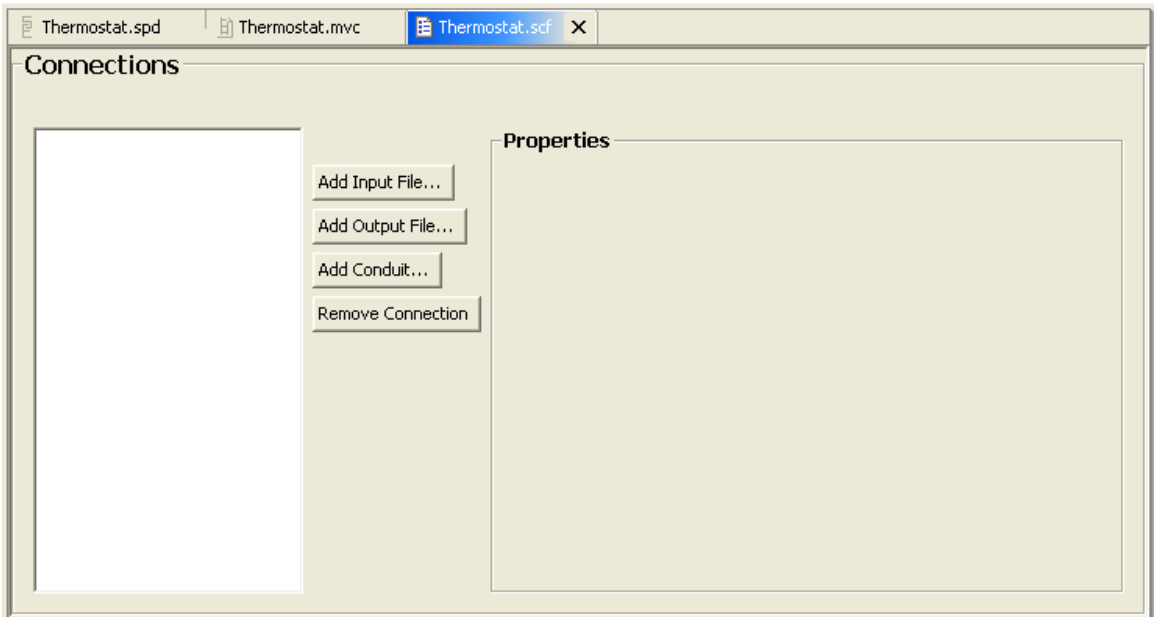

Figure 60 - Empty Connections Page

2. Press the **Add Input File…** button.
3. Select the Set Point Data.csv file to use as an input source and click **Next**.

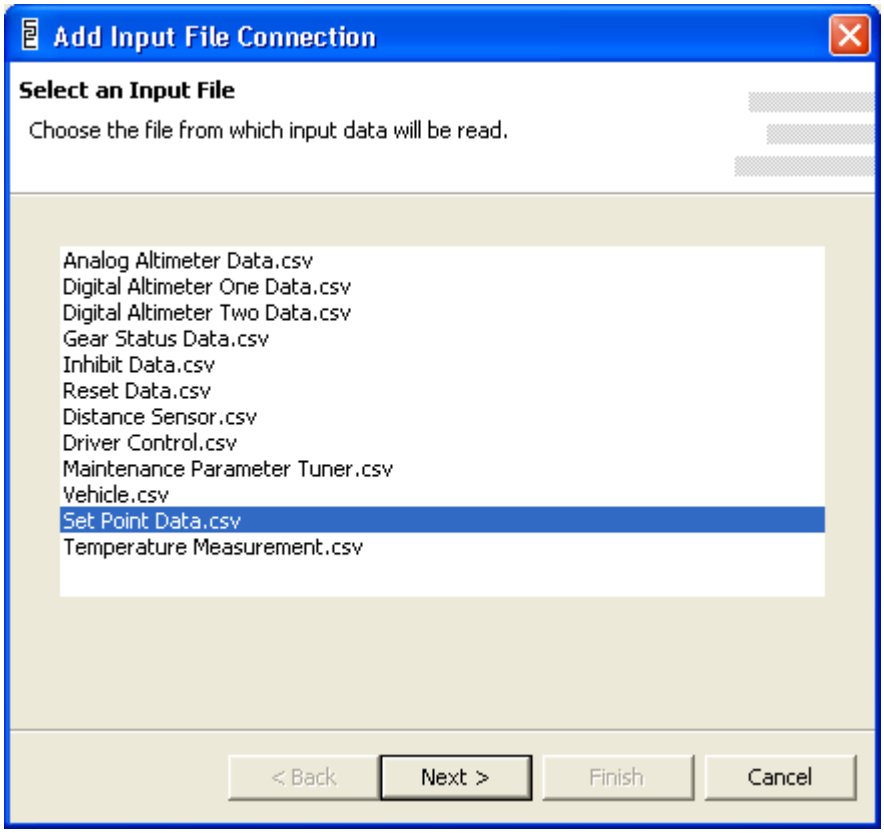

Figure 61 - Select Input File

4. Select the Set Point input as the destination for the input data and click **Finish**.

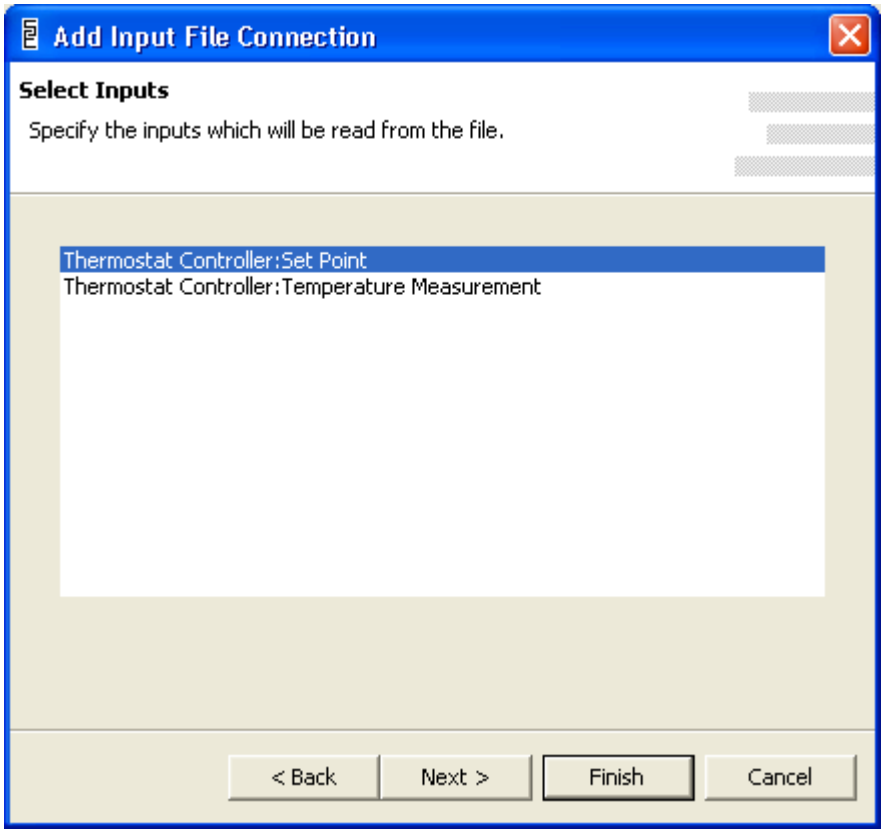

Figure 62 - Select Set Point Input

5. Now repeat the process to hook up the Temperature Reading.csv data file to the Temperature Measurement input. When finished, the connections screen should look like figure 63, below.

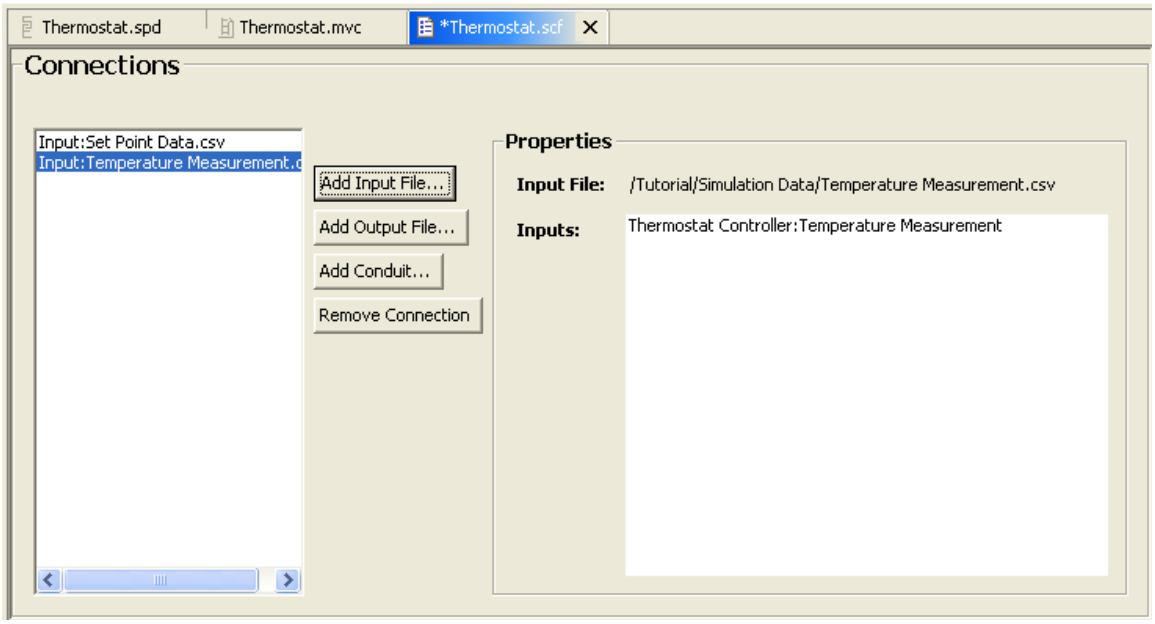

Figure 63 - Input Files Added

We now add an output files to record the furnace outputs issued by the executing model.

1. Click on the **Add Output File…** button.
2. Select the output command to activate the furnace.

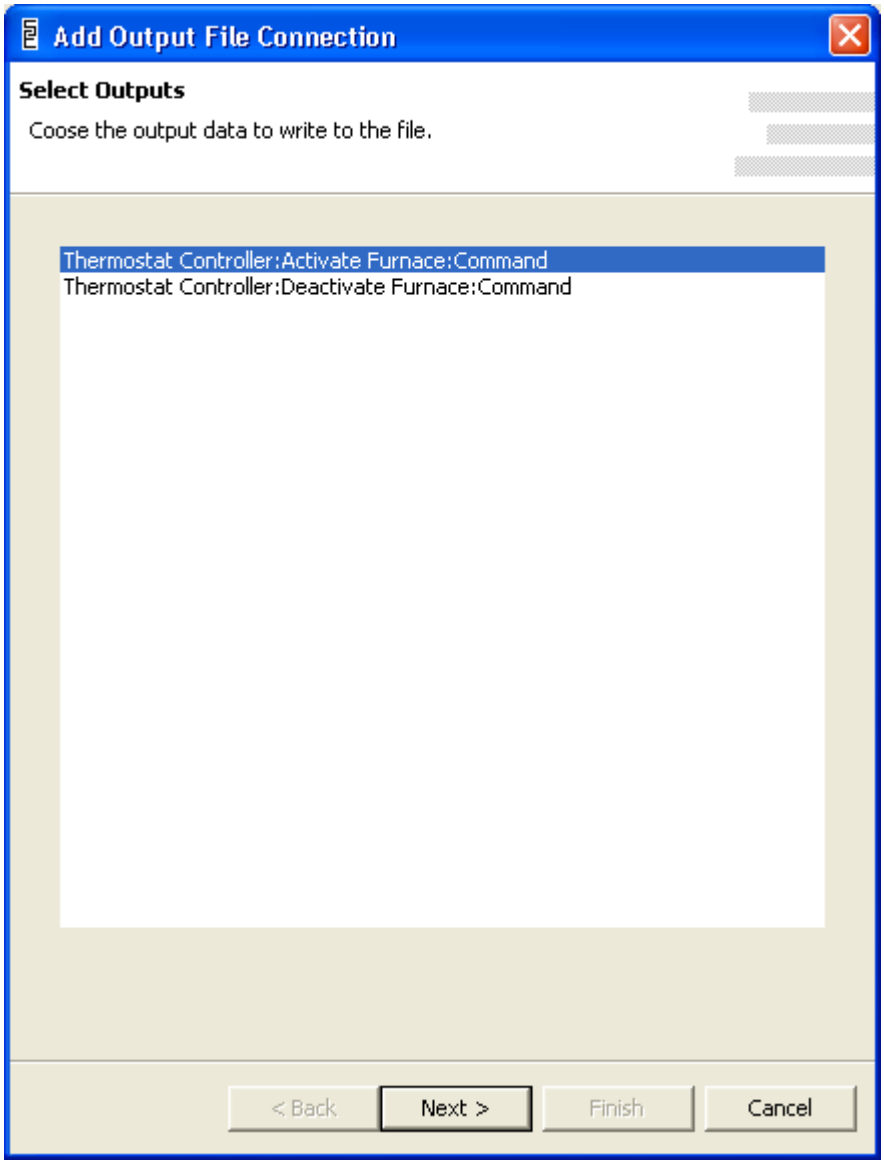

Figure 64 - Select Output for File

3. Supply a folder and file name for the output file, such as the one in figure 65, below and click **Finish**.

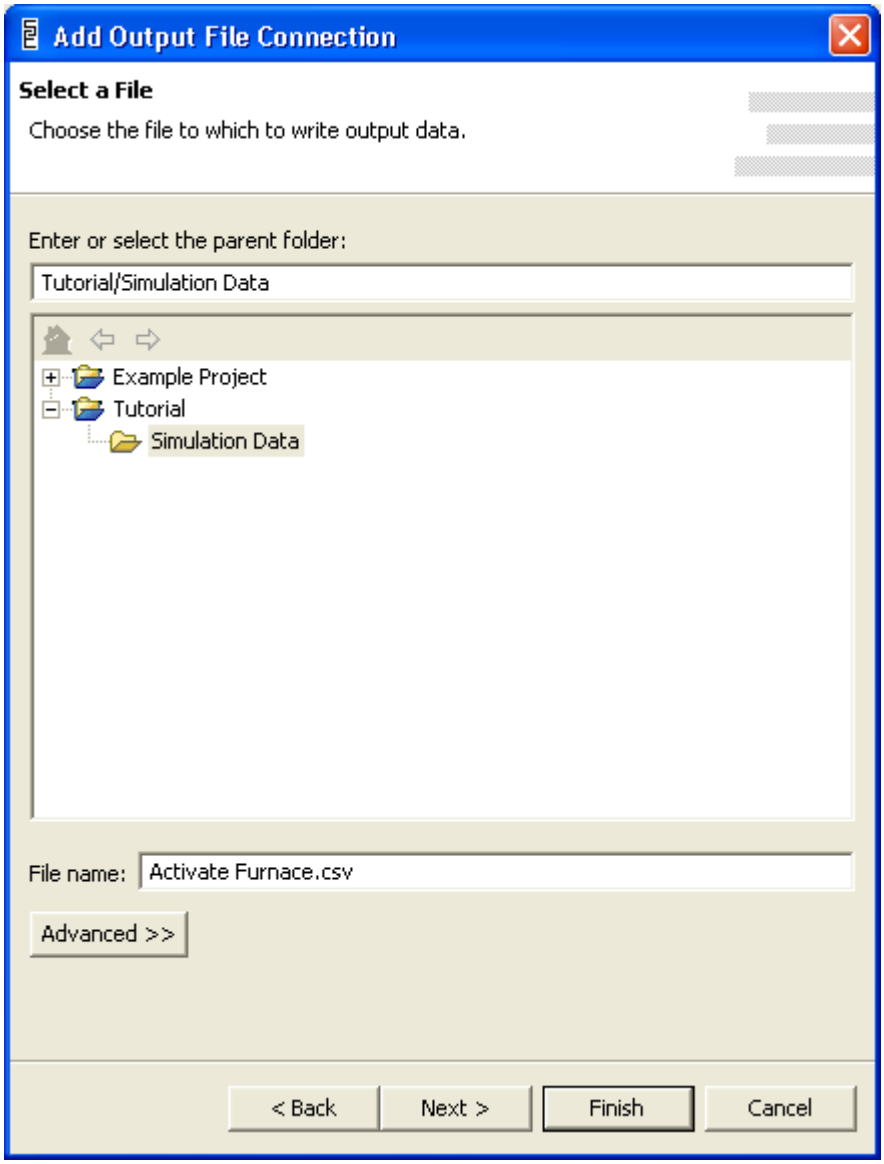

Figure 65 - File for Output

4. SpecTRM will add the output file to the list of connections, as shown in figure 66, below.

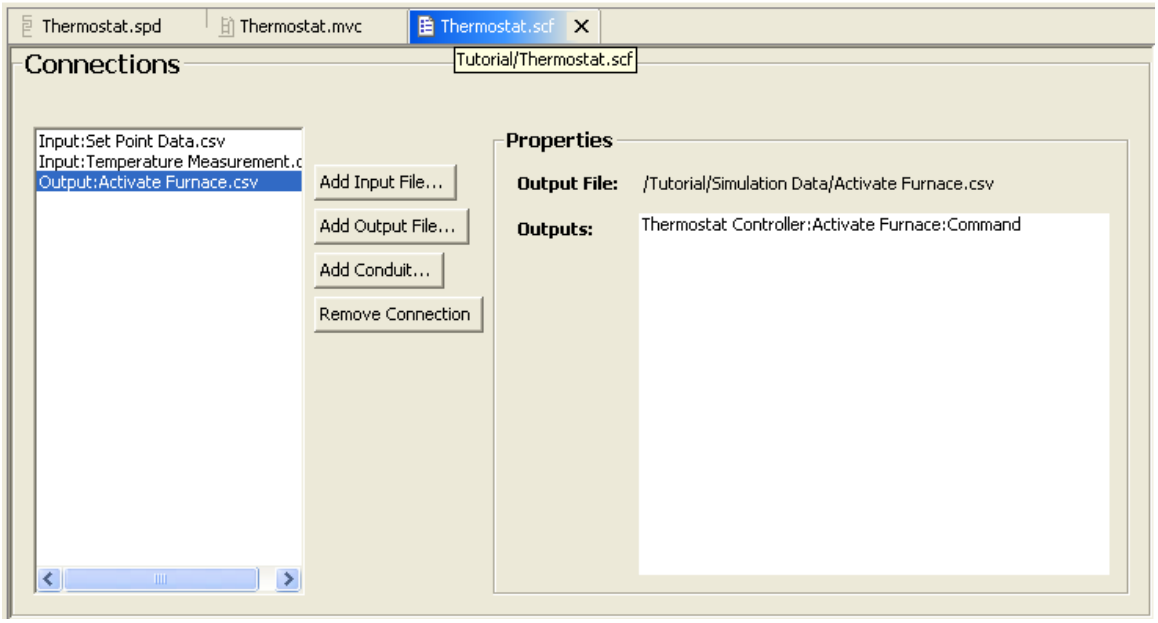

Figure 66 - Connections with Output File Added

Follow the procedure above to add an output file for the furnace deactivation command. Save the simulation configuration file. The simulation is now ready to run.

1. Select the **Simulate** > **Setup Simulator…** command.

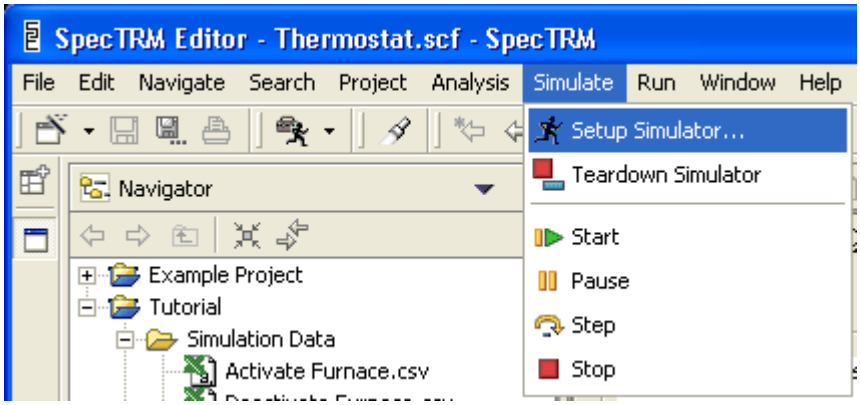

Figure 67 - Setup Simulator Command

2. Select the Thermostat simulation configuration and click **OK**.

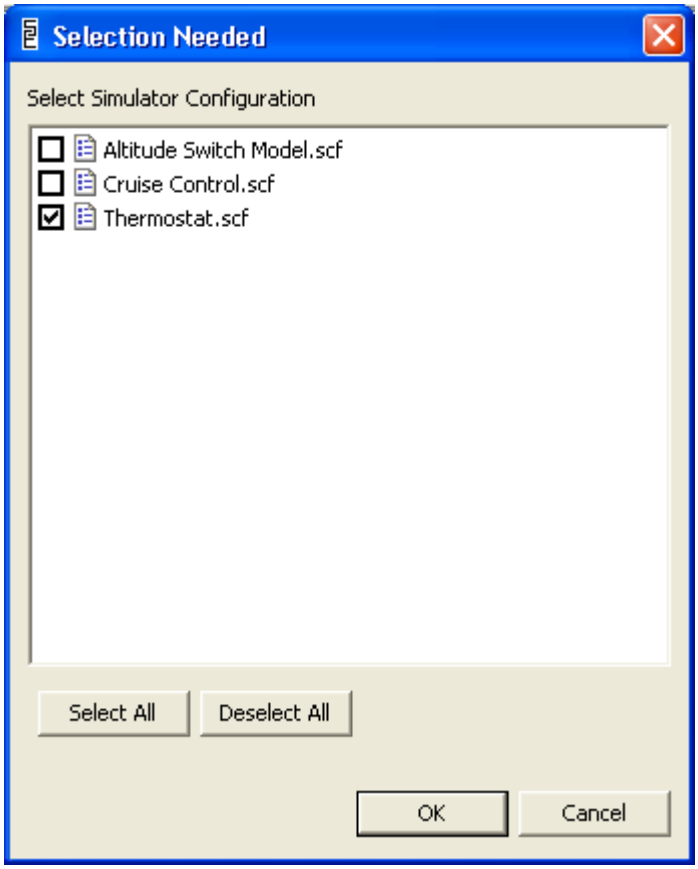

Figure 68 - Choice of Simulation Configurations

3. A little time will pass as SpecTRM reads in the model, parses it, and validates it. SpecTRM will automatically open a new perspective, the simulation perspective.

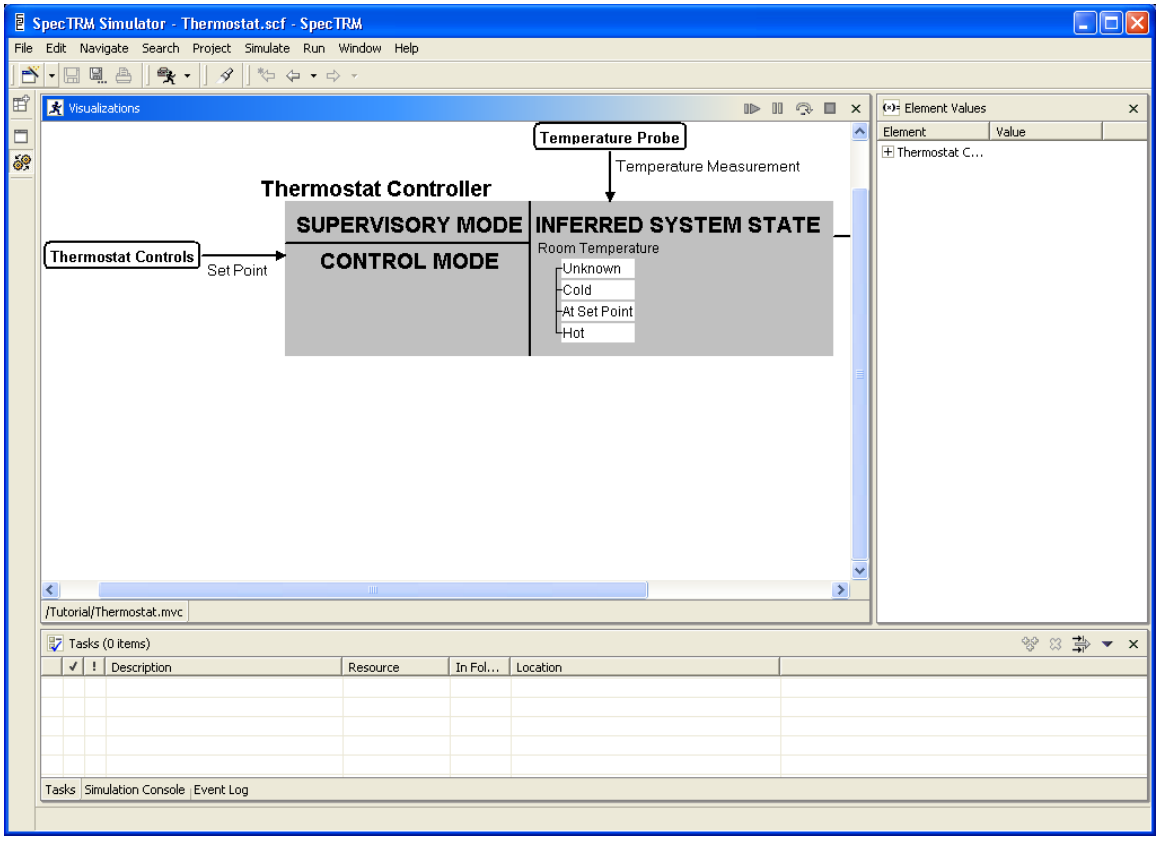

Figure 69 - Simulation Perspective

The simulation perspective contains the visualization in the top left position. The top right contains a list of element values. Expanding the list will show all the inputs, outputs, states, and modes used in the simulation along with their current values. The bottom of the perspective contains a few views, including the task view and a simulation console and log. The simulator is controlled with the buttons at the top right of the visualization view. These buttons start, pause, step, and stop the simulation.

4. Press the start button followed shortly thereafter by the pause button.

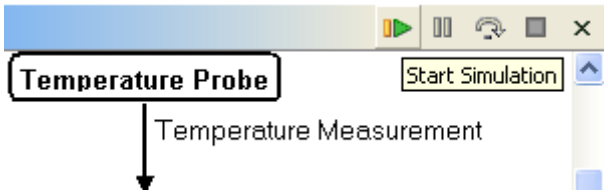

Figure 70 - Simulation Tool Bar

5. SpecTRM will start the simulation and pause it when the pause button is pushed. You should see something similar to figure 71, below.

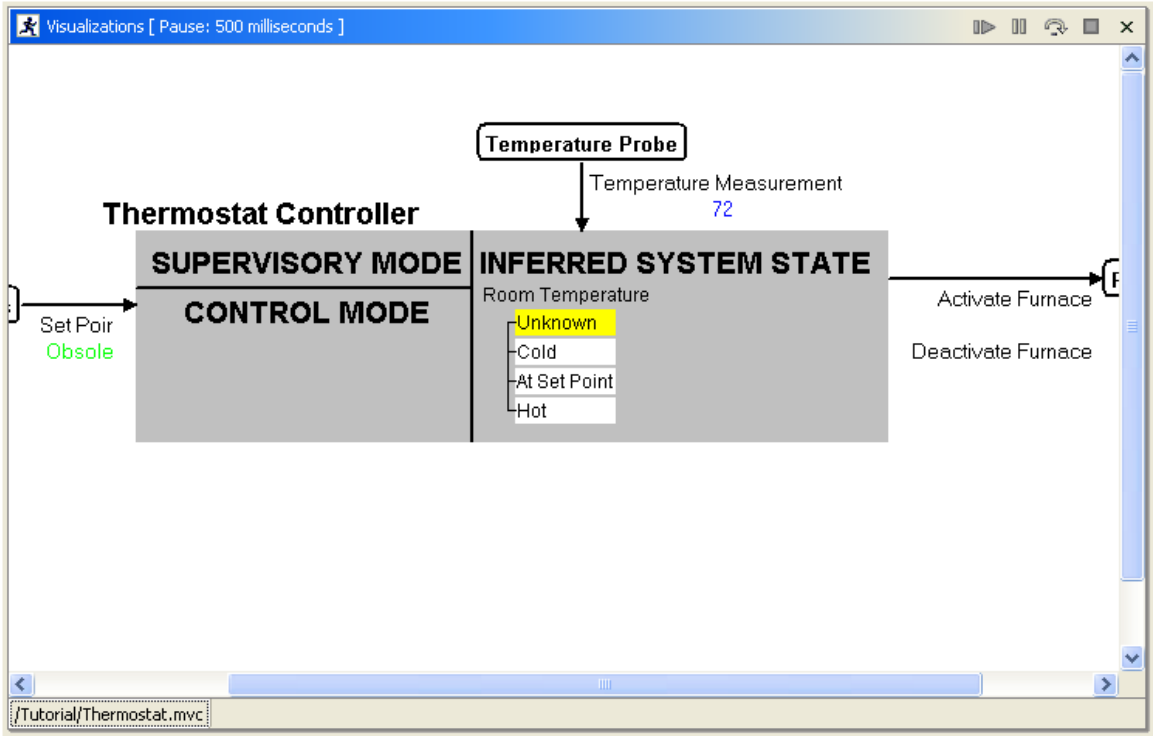

Figure 71 - Beginning of Simulation

The visualization is animated to show what the requirements model is doing during execution. Values in blue are inputs currently arriving. Inputs shown in a light green are the most recent values to arrive sometime previously. Whether any of the inputs are in blue on your screen will depend on when in the data sequence you paused the simulator. The Room Temperature state element has a yellow highlight indicating the current state. If you stopped the simulation before 1 second has passed, then the room temperature state is unknown – although a temperature measurement has come in, the user has not yet set the set point. Recall that we set up the data files for the set point to come in at 1 second. Before that, the set point is obsolete, indicating that the data is either too old or, as in this case, was never received. Use the step button to step forward to 1 second.

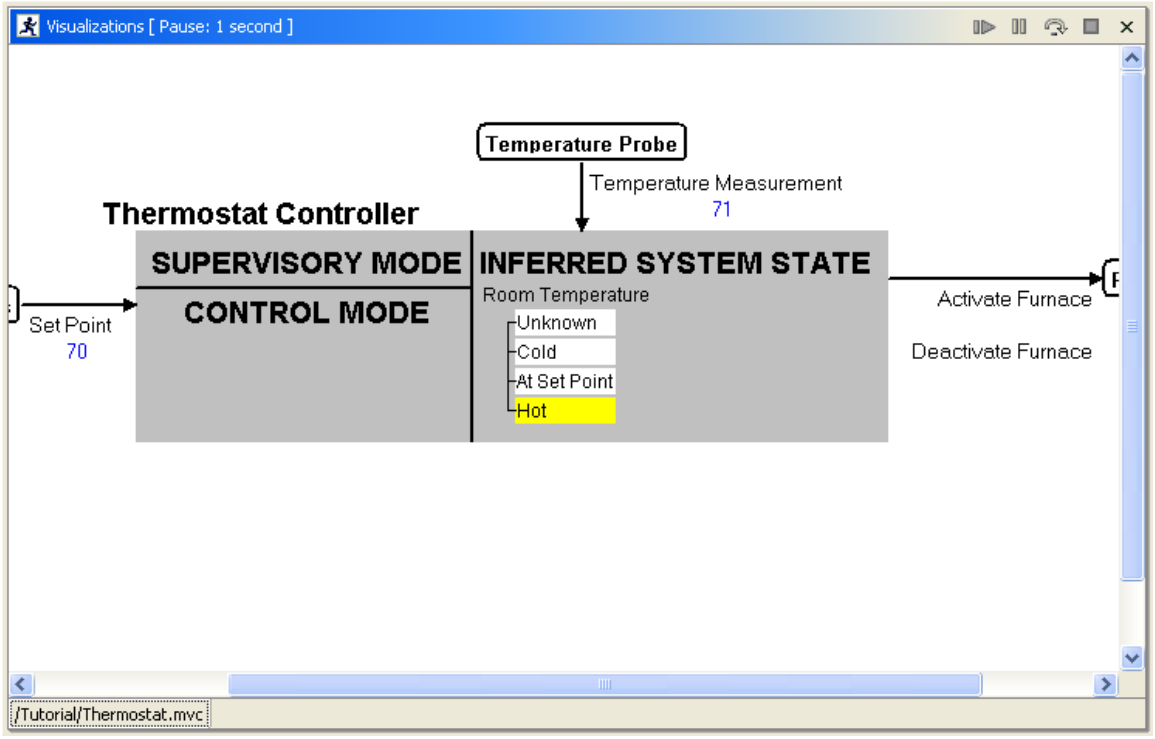

Figure 72 - Simulation at 1 Second

At one second, the user inputs the set point that the system is to maintain. Notice the room temperature transitions to hot, indicating that the temperature measurement is above the set point. Step forward to 1.5 seconds and watch the simulation transition to At Set Point. Step forward again to 2 seconds, and notice that at the room temperature falls below 70, the room temperature state transitions to Cold, causing the Activate Furnace command to be sent, as shown in figure 73, below.

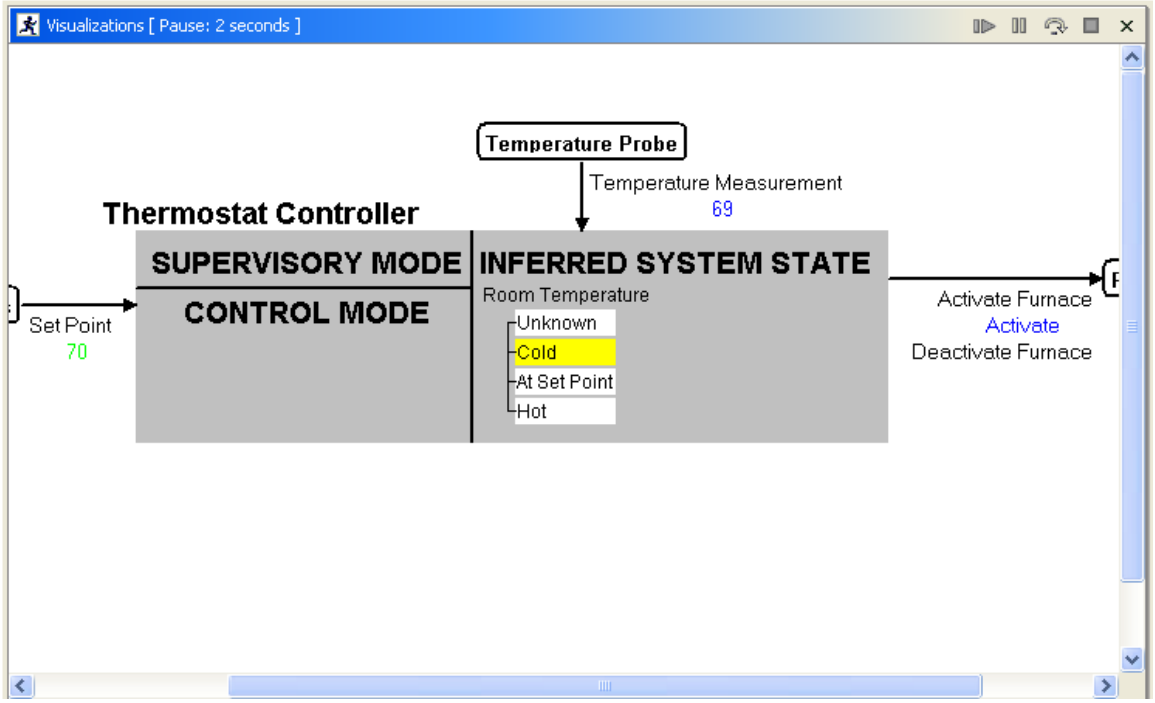

Figure 73 - Transition to Cold Sends a Furnace Activation Command

Continuing the simulation forward will run through the data provided in the text files we created earlier. When the simulation runs out of data, the measurement input will eventually go to obsolete. Press stop to halt the simulator. After the simulation is complete, tear down the simulation with the **Simulate** > **Teardown Simulator** command.

Analysis

Execution of requirements is an excellent way to gain insight into the runtime behavior of a system. However, a simulation is dependent on the data used to drive the execution. If the data used shows a requirements error, the defect can be corrected. However, if the available data doesn't demonstrate a problem, then it will not be detected using simulation alone. Static analyses of a model help find problems without relying on particular data chosen to exercise the system.

One static analysis provided by SpecTRM detects nondeterminism. Nondeterminism occurs when more than one AND/OR table in a state or mode element is true at the same time. For example, an error in the requirements for the thermostat controller might make it possible for the tables for Cold and Hot to be true at the same time. Obviously, the room cannot be both below and above the set point temperature, so this would indicate an error in the requirements. Another error would be if the triggering conditions to activate and deactivate the furnace were true at the same time – only one of those outputs should be sent at a time. We will test this latter case using the nondeterminism analysis.

1. Select **Analysis** > **Determinism Analysis** from the main menu.

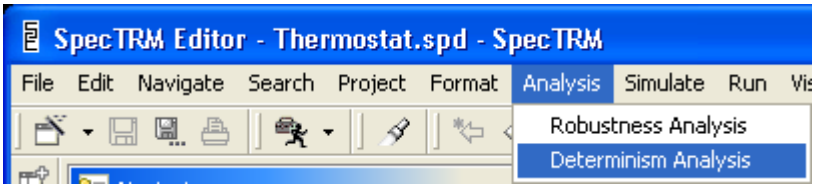

Figure 74 - Determinism Analysis Menu Command

2. Select the Thermostat.spd file.

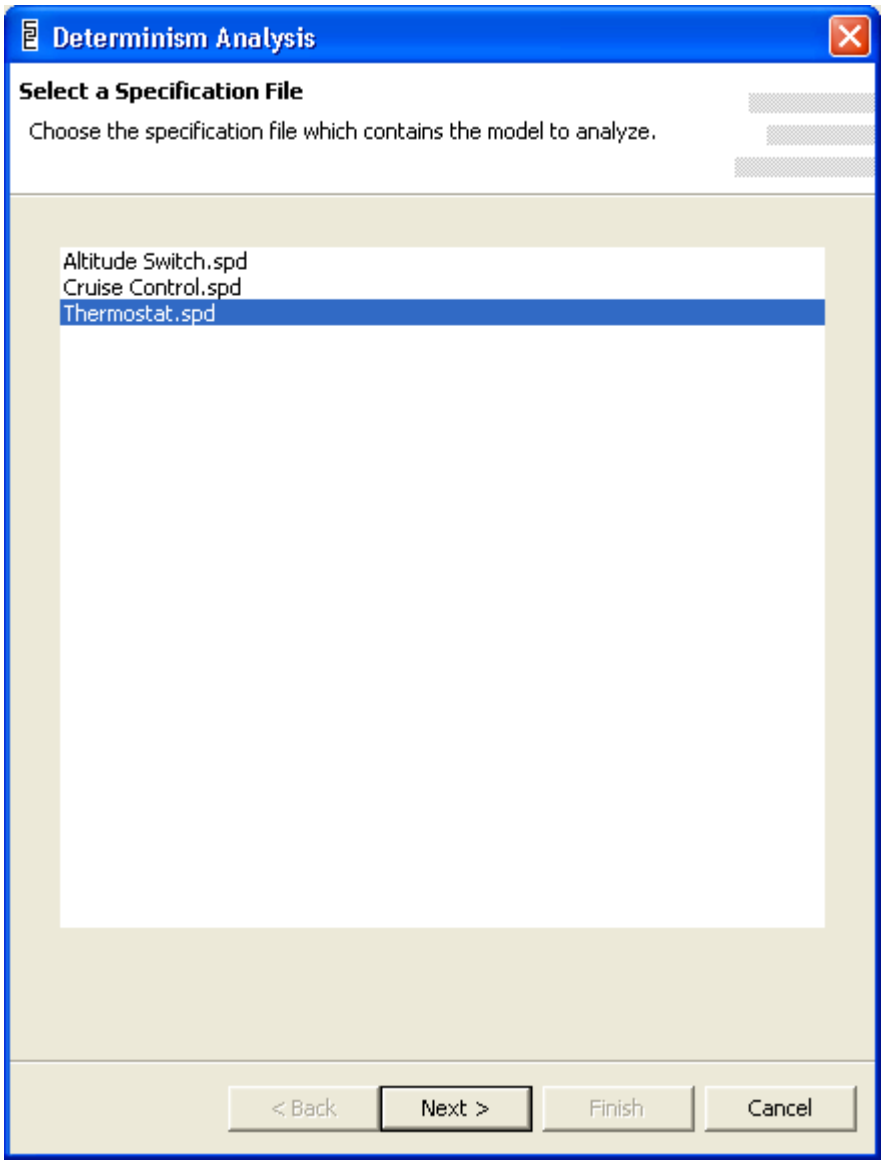

Figure 75 - Select a File for Analysis

3. Select the model to analyze.

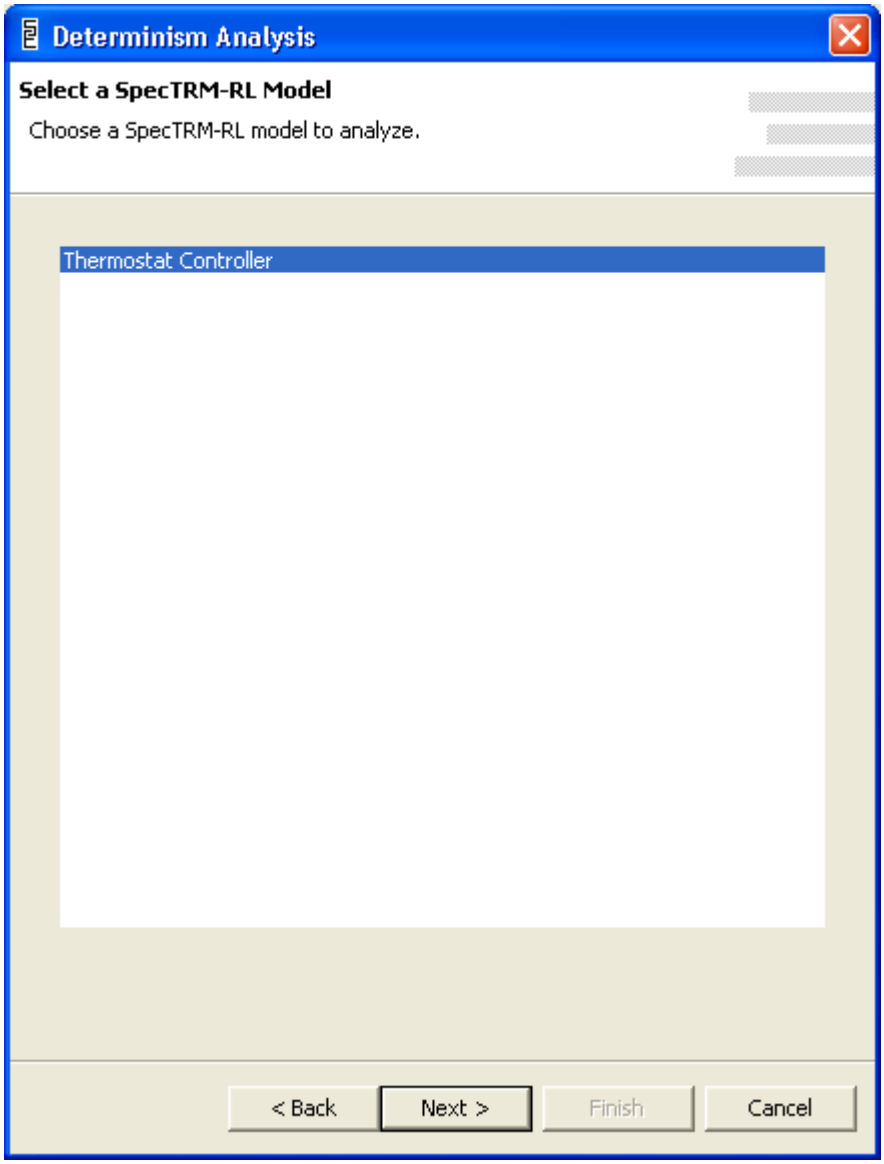

Figure 76 - Select Model to Analyze

4. Expand the output and select the two triggering conditions. To select the first, left click on it. To select the second, ctrl + left click.

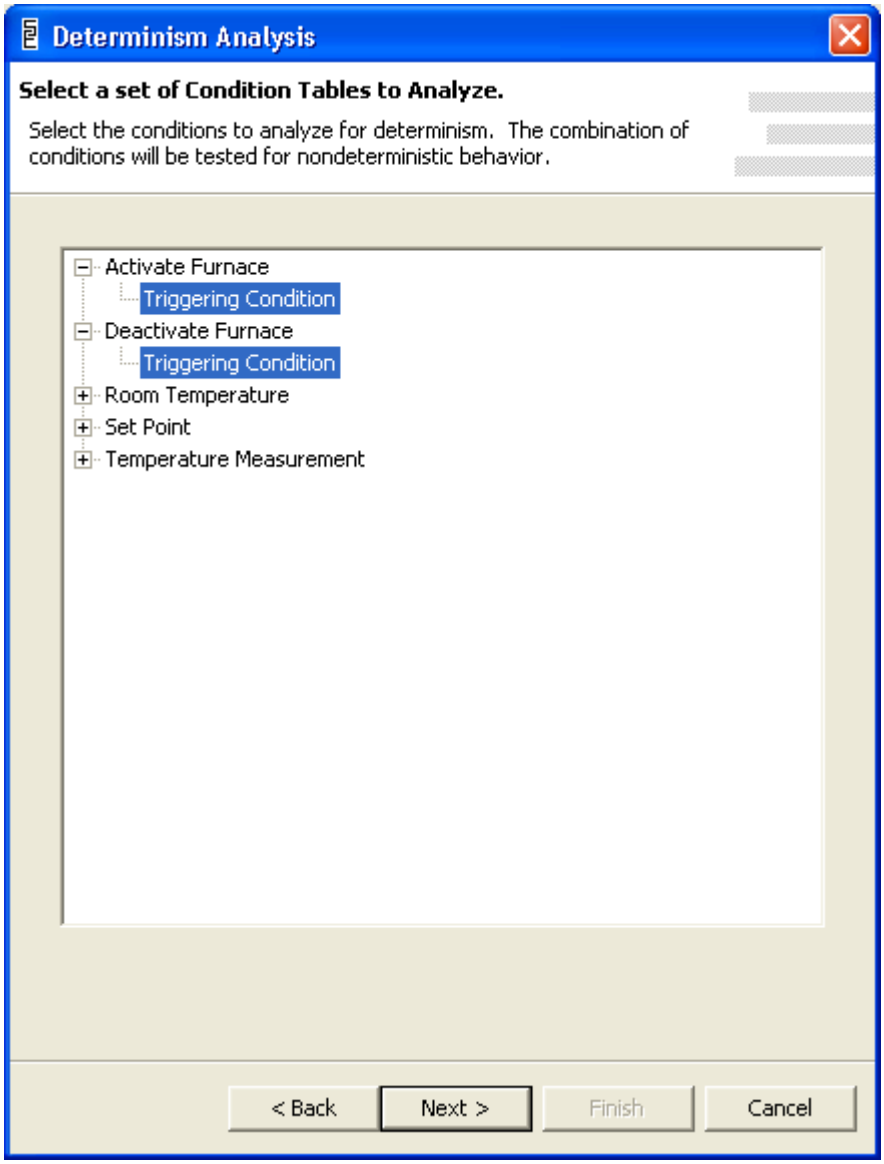

Figure 77 - Select Tables to Analyze for Nondeterminism

5. Give SpecTRM a file in which to write the results.

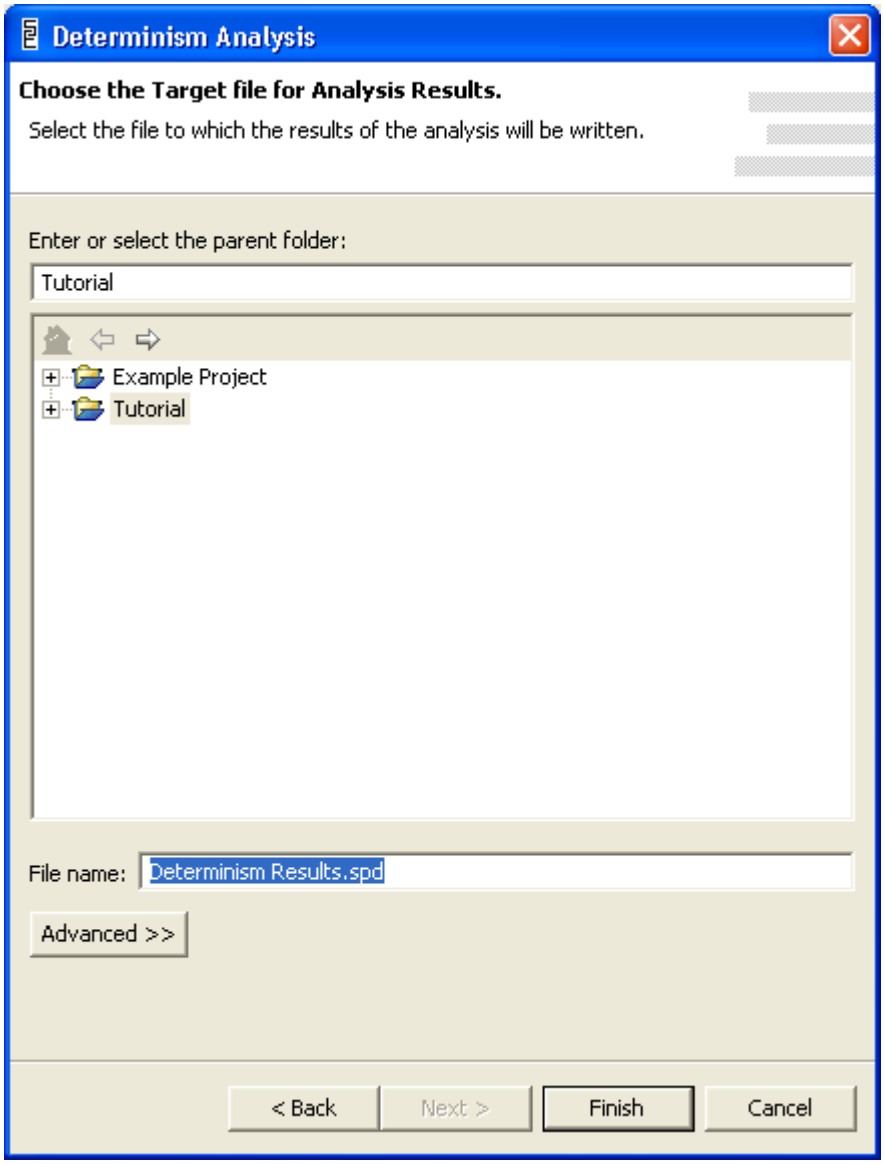

Figure 78 - File Selection Dialog

6. The analysis results will be generated and the results file opened. The results should look like figure 79, below.

Analysis for Nondeterministic Scenarios

Original Conditions

Room Temperature in state Cold	T
Previous Value of Room Temperature in state Cold	F

Room Temperature in state Cold	F
Previous Value of Room Temperature in state Cold	T

Results

No scenarios were found.

Figure 79 - Analysis Results

Note that the results have found no examples of nondeterminism here. The thermostat model is so simple that we would have to contrive examples in which nondeterminism appeared in the model. Were there examples of nondeterminism found by the analysis, the results would contain a list of expression values that would expose the nondeterminism, making it a simple task to add conditions to the tables to fix the problem. To see examples of nondeterminism, explore the altitude switch and cruise control examples.

The other static analysis SpecTRM can perform is a robustness analysis. A non-robust set of tables has scenarios under which no table is true. This is something of the inverse to nondeterminism; nondeterminism involves multiple tables being true, where non-robustness involves no tables being true. Exactly one table should be true for any combination system states and inputs.

For more information about SpecTRM's capabilities and the SpecTRM-RL modeling language, consult the online user reference manual included with the software. For more information about using SpecTRM's features to support system safety, refer to the papers at http://www.safeware-eng.com/index.php/publications and http://sunnyday.mit.edu/papers.html

If you have any questions, please feel free to contact us at support@safeware-eng.com.