# Introduction to Stateflow® Hands-On Workshop

Welcome to The MathWorks Introduction to Stateflow Workshop! Stateflow extends the capabilities of the Simulink® environment for development of

* State Machines
* Flow Charts
* Truth Tables
* Control Logic
* Fault Tolerance Logic
* Mode Transition Logic

This half-day session is intended for individuals with no Stateflow experience, though experienced users should learn a few new tricks. Users looking to maximize their Stateflow proficiency should consider formal [Training from The MathWorks](http://www.mathworks.com/services/training/index.html).

In the mean time, this document will provide detailed instructions on how to construct our Stateflow chart. Refer to it during the workshop and afterwards when you return to your office.

*Models*

[Step 1 Model](matlab:Step_1)

[Step 2 Model](matlab:Step_2)

[Step 3 Model](matlab:Step_3)

[Step 4 Model](matlab:Step_4)

[Step 5 Model](matlab:Step_5)

[Step 6 Model](matlab:Step_6)

[Step 7 Model](matlab:Step_7)

[Step 8 Model](matlab:Step_8)

*Contents*

[Introduction to Stateflow® Hands-On Workshop 1](#_Toc219093448)

[1.0 Problem Overview 2](#_Toc219093449)

[1.1 Test out the Model 3](#_Toc219093450)

[2.0 Add a Basic Stateflow Chart 4](#_Toc219093451)

[2.1 Add a Stateflow Chart to the Model 4](#_Toc219093452)

[2.2 Add Device States 5](#_Toc219093453)

[2.3 Create Stateflow Transitions 6](#_Toc219093454)

[2.4 Create Transition Conditions 7](#_Toc219093455)

[2.5 Add State Entry Actions 8](#_Toc219093456)

[2.6 Define Stateflow Variables 9](#_Toc219093457)

[2.7 Connect Stateflow to the Simulink Model 11](#_Toc219093458)

[3.0 Protect Against Overheating 13](#_Toc219093459)

[3.1 Expand on the Transition Conditions 13](#_Toc219093460)

[3.2 Create a Stateflow Subchart 15](#_Toc219093461)

[3.3 Create a Graphical Function 18](#_Toc219093462)

[3.3 Test the Model 19](#_Toc219093463)

[4.0 Add On/Off Fan Logic 20](#_Toc219093464)

[4.1 Add a Parallel State 20](#_Toc219093465)

[4.2 Define Fan States 21](#_Toc219093466)

[4.3 Specify Variable Properties in the Model Explorer 23](#_Toc219093467)

[4.4 Add a State During Action 23](#_Toc219093468)

[5.0 Add a Second Fan Speed 24](#_Toc219093469)

[5.1 Create a Third Fan State 25](#_Toc219093470)

[5.2 Create a Truth Table 26](#_Toc219093471)

[5.3 Define Truth Table Logic 27](#_Toc219093472)

[5.4 Integrate Truth Table into Chart 29](#_Toc219093473)

[5.5 Add Time-dependent Logic 29](#_Toc219093474)

[6.0 Add a Backup Fan 30](#_Toc219093475)

[6.1 Create a Second Fan Subchart 30](#_Toc219093476)

[6.2 Define Fan Transitions 32](#_Toc219093477)

[6.3 Create an Embedded MATLAB Function 33](#_Toc219093478)

[7.0 Add a Warning Event 34](#_Toc219093479)

[7.1 Create an Event 34](#_Toc219093480)

[7.2 Add a Warning Subchart 35](#_Toc219093481)

[7.3 Broadcast an Event 37](#_Toc219093482)

[7.4 Add a State Activity Output 37](#_Toc219093483)

[8.0 Quiz Time! 39](#_Toc219093484)

[9.0 Further Information 40](#_Toc219093485)

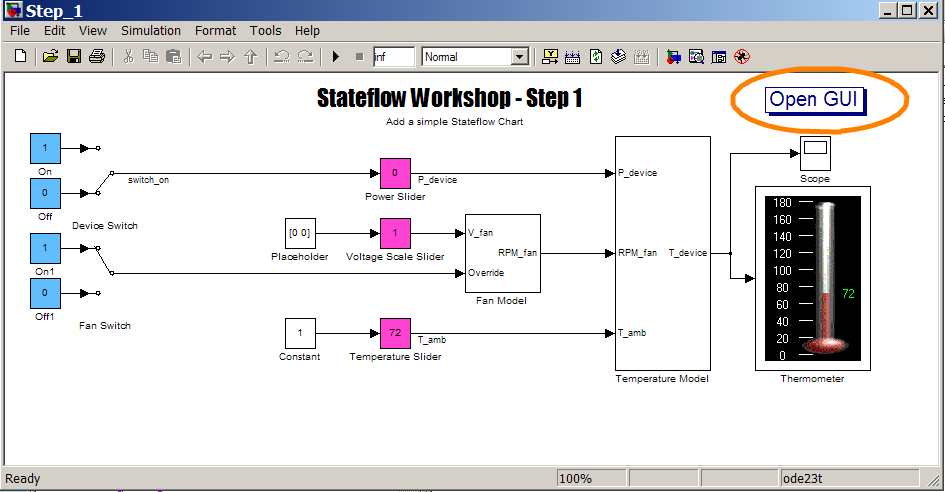
[9.1 Product Information 40](#_Toc219093486)

[9.2 Documentation 40](#_Toc219093487)

## 1.0 Problem Overview

We are building the control logic for a fan-cooled device. This device is a piece of electronics equipment that generates internal heat when activated. It could be an avionics R&D package, a CD player in automobile, a PC projector, etc. The system is being modeled in Simulink at present with no control logic.

The ultimate objective of the model is to simulate the temperature of the device as a function of its activity and local atmospheric conditions. A Thermometer block from the Gauges Blockset has been connected to the temperature signal for ease of viewing.

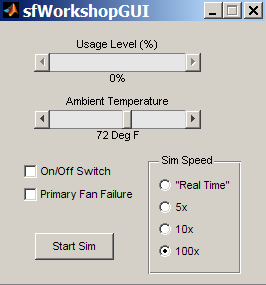


There are five tunable parameters in the model:

|  |  |  |
| --- | --- | --- |
| Parameter | Default Setting | Range |
| Device On/Off Switch | off | [off on] |
| Fan Failure Switch | off | [off on] |
| Device Power Usage | 0% off, 20% on | [0 100] |
| Ambient Temperature | 72° F | [0 125] |

A graphical user interface is provided to control the values of all these parameters. Initially, some of these parameters won't affect the model. As we refine our Stateflow chart, all elements of this GUI will come into play. To open the GUI, click the "Open GUI" button in the upper right corner of the Simulink model.

[Back to Top](#_top)



### 1.1 Test out the Model

Do the following:

1. [Open the initial Simulink model](matlab:Step_1)
2. [Open the graphical interface](matlab:sfWorkshopGUI) by clicking Open GUI
3. Start the simulation by pressing the Start Sim button in the GUI
4. Observe the effect of changes in the GUI to the Thermometer
   1. Turn the device on and off
   2. Change the device power usage
   3. Change the ambient temperature
5. Stop the simulation by pressing the Stop Sim button in the GUI

In addition to the Thermometer, there is a scope attached to the temperature signal. You may also observe the last 5000 temperature values by double-clicking on the Scope block. Note that the model is running faster than real time unless specified in the GUI.

[Back to Top](#_top)

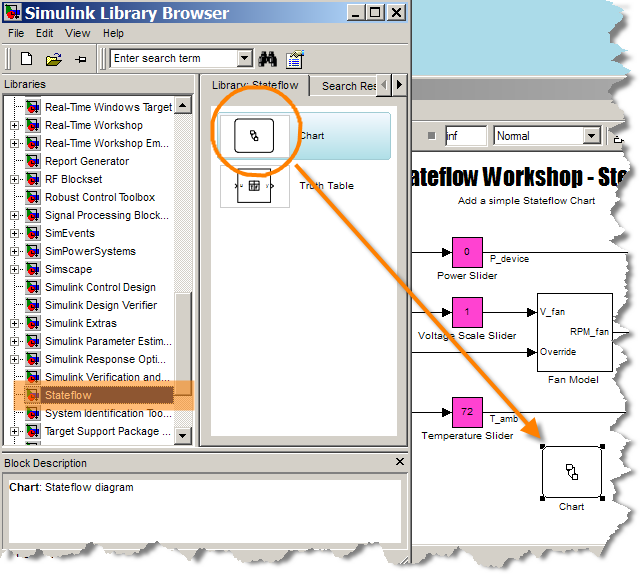
## 2.0 Add a Basic Stateflow Chart

Initially, the Simulink model lacks functionality to activate a fan or automatically deactivate the device when it gets too hot. We will use a Stateflow chart to manage these activities. Our initial step in this process will be to delineate between the switch being turned on and the device itself being on (i.e. generating heat).

### 2.1 Add a Stateflow Chart to the Model

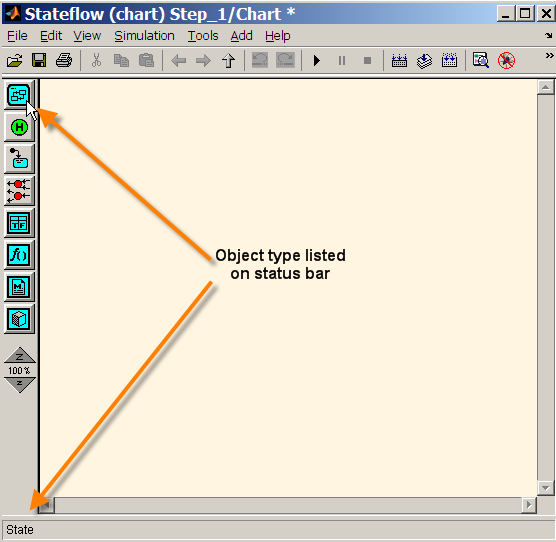
Do the Following:

1. [Open the initial Simulink model](matlab:Step_1)
2. [Open the graphical interface](matlab:sfWorkshopGUI) by clicking Open GUI
3. [Open the Simulink Library Browser](matlab:simulink)
4. Scroll down and click Stateflow in the library list
5. Click and drag the Chart block into the Step\_1 Simulink model
6. Double-click on the Chart block to open the Stateflow editor



At this point, you can close the Library Browser if you wish; it won't be used for the bulk of the workshop.

While the Stateflow Chart exists inside the Simulink diagram, the specifics of the chart are not displayed on the top-level diagram. Much like a subsystem, double-clicking enables you to examine Stateflow details. Given that Stateflow manages different kinds of problems, a specialized Stateflow design environment is employed. All objects used to build a Stateflow chart are displayed on the left side of the window. This sidebar is similar to the Simulink Library Browser except that it is integrated into the Stateflow chart rather than a separate window. Hovering your mouse over an icon will result in the description of that object displayed in the status bar at the bottom.

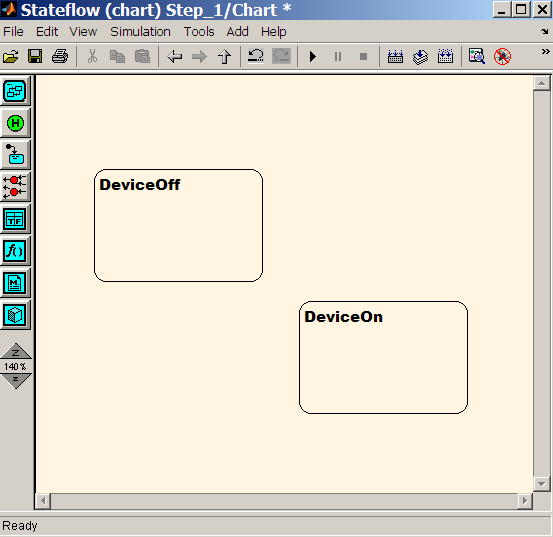


[Back to Top](#_top)

### 2.2 Add Device States

Do the following:

1. Click the State icon in the Stateflow editor and place a state in the chart
2. Give the state a name by typing DeviceOff
   1. Click the question mark if you missed the prompt
3. Create a second state and name it DeviceOn



Now we have two independent states in our chart: DeviceOff and Device On. The device will be in one of these two states, but never both. As such, DeviceOn and DeviceOff are considered exclusive states.

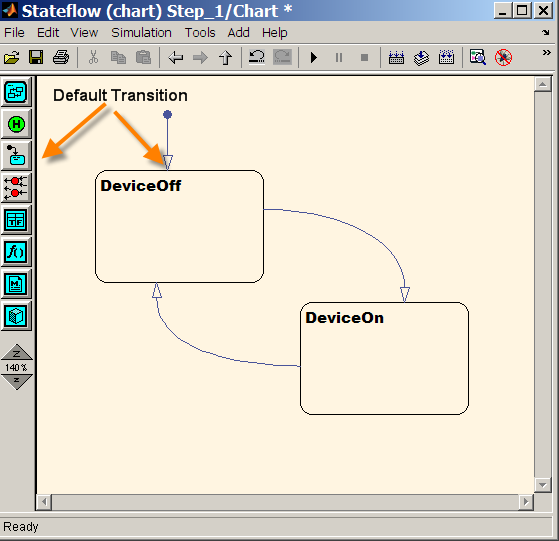
Based on certain conditions in the system, the device will need to switch between the two states DeviceOff and DeviceOn. The two states are currently unconnected, so let's specify pathways for transition.

[Back to Top](#_top)

### 2.3 Create Stateflow Transitions

Do the following:

1. Hover the mouse near the edge of DeviceOff until you see a crosshair
2. Hold down left-click and drag the mouse to the edge of DeviceOn
3. Create a second transition from DeviceOn to DeviceOff
4. Grab a Default Transition from the left sidebar and connect it to DeviceOff



By adding these transitions, we are specifying that the state of the system can change from DeviceOff to DeviceOn and vice-versa. The final action of adding a default transition is critical in that it defines the initial state of the system. Without a default transition, a warning will occur.

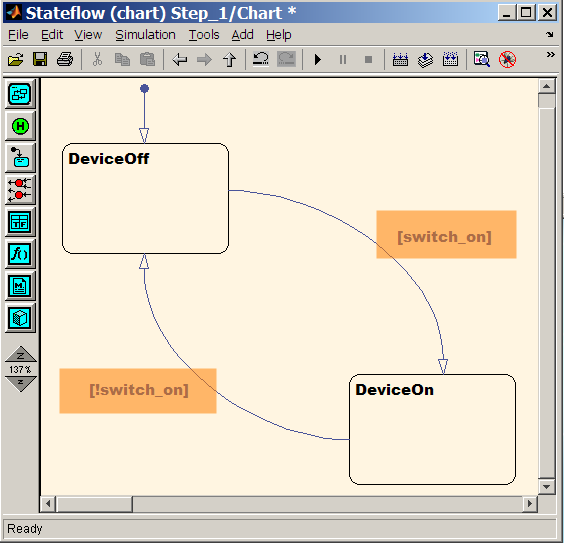
Each transition may have a label associated with it. A variety of information can be prescribed with these labels. There is a specialized Stateflow language used to define pieces of information on labels and elsewhere. We will touch on a few but not all of the items that can be prescribed on labels.

[Back to Top](#_top)

### 2.4 Create Transition Conditions

Do the following:

1. Left-click on the transition from DeviceOff to DeviceOn
2. Click the question mark to begin editing the label text
3. Type [switch\_on]
4. Repeat the process for the DeviceOn to DeviceOff transition, typing [!switch\_on] this time



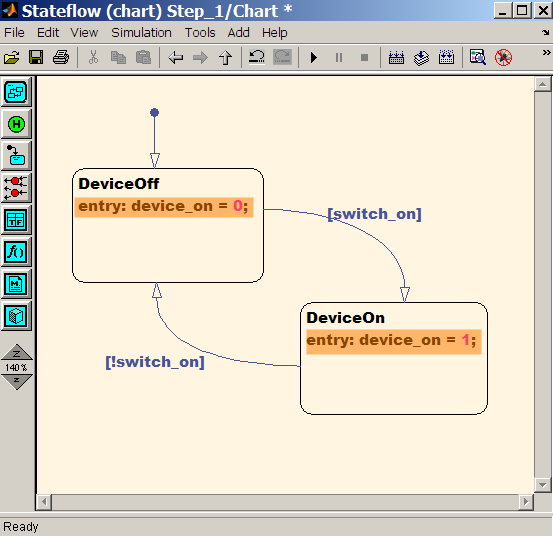
In the Stateflow language, brackets signify a transition condition. Thus by adding the preceding text to the transition labels, we have stated "switch to DeviceOn when the switch is on and back to DeviceOff when it is (!) not on."

[Back to Top](#_top)

### 2.5 Add State Entry Actions

Do the following:

1. Click the text of DeviceOff and press enter
2. Type entry: device\_on = 0;
3. Repeat the process for the DeviceOn state, typing entry: device\_on = 1; instead



In addition to names for states, we can add text that specifies additional information similar to that of transition labels. The syntax entry: will cause all subsequent text on the line to be executed once at entry to that state. In our case, we have established a variable that is set to 0 or 1 depending on which of the two states we are in. Note that without the semicolon, our statements would echo to the MATLAB workspace.

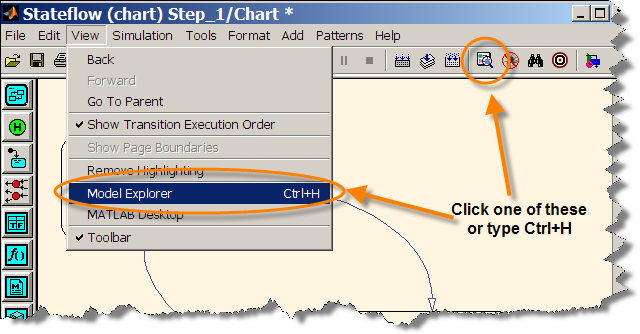
You may notice that we've established two variables in our chart: switch\_on and device\_on. These are going to correspond to the input and output respectively for our chart. If however we examine our chart in the Simulink model, we still lack input and output ports. No correlation has been made between our two variables and chart I/O.

[Back to Top](#_top)

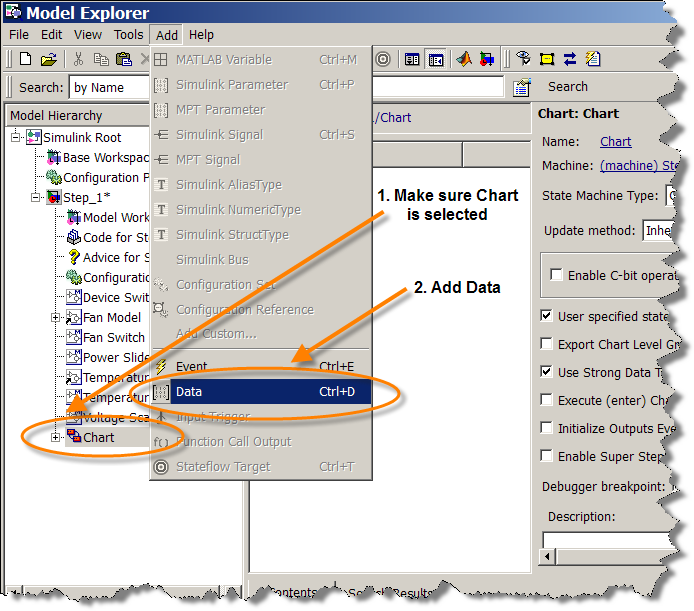
### 2.6 Define Stateflow Variables

Do the following:

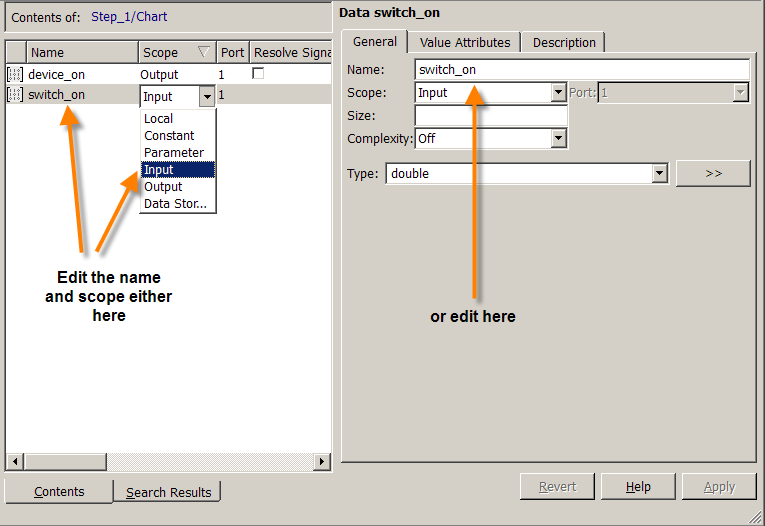
1. Open the Model Explorer through the View Menu, Model Explorer Icon, or Ctrl+H



1. With the Chart selected in the Model Hierarchy, select Data from the Add Menu or type Ctrl+D to create 2 data objects



1. Using either the middle or right pane, name one of the data objects switch\_on and set its scope to **Input**
2. Using either the middle or right pane, name one of the data objects device\_on and set its scope to **Output**



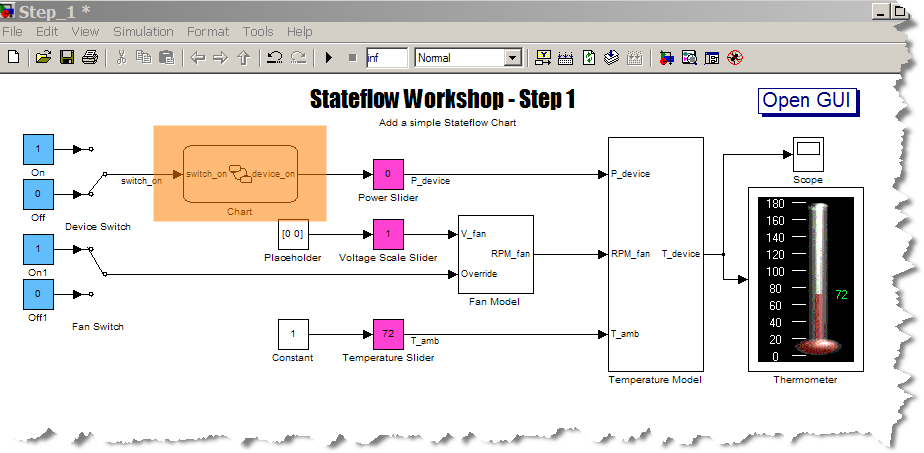
Unlike MATLAB, Stateflow requires all variables in the workspace be explicitly defined. With the Stateflow Symbol Wizard, we have defined switch\_on and device\_on as input and output. There are other Scopes for variables in Stateflow, some of which will be touched on in this Workshop. If you ever need to make changes to these data objects, you can return to the Model Explorer and make the proper adjustments.

[Back to Top](#_top)

### 2.7 Connect Stateflow to the Simulink Model

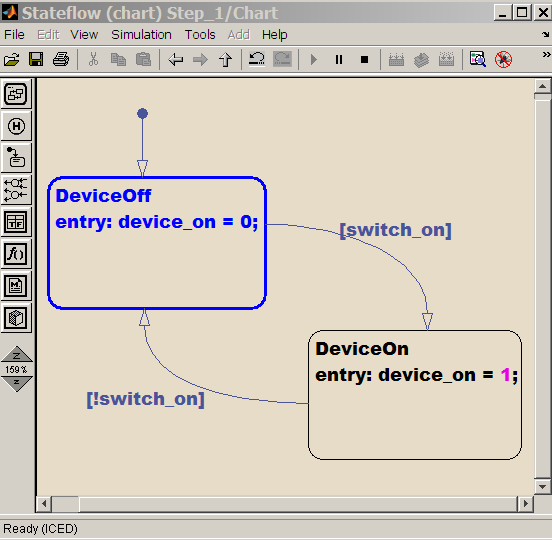
Do the following:

1. Insert your Stateflow chart in between the Device Switch and Power Slider in the Simulink Model
2. Start the simulation
3. Observe the Stateflow chart changing from DeviceOff to DeviceOn as you toggle the switch



You may have noticed a small delay when you pushed play. Stateflow charts are converted to C code and compiled upon simulation execution. If no changes are made to the chart, then subsequent executions will not incur the build time penalty since the code is already compiled.

If the Stateflow chart was open, you would have noticed that the present state of the chart was highlighted in blue as the simulation ran. The visual cues provided by Stateflow are handy for analyzing your chart performance, especially as charts become more elaborate.



[Back to Top](#_top)

## 3.0 Protect Against Overheating

At the moment, we have a functioning Stateflow chart that turns our device on and off depending on the setting of the switch. Considering that this is the same functionality that we started with before the chart, nothing really has been gained yet. We will now expand on our existing Stateflow chart and protect our device from overheating.

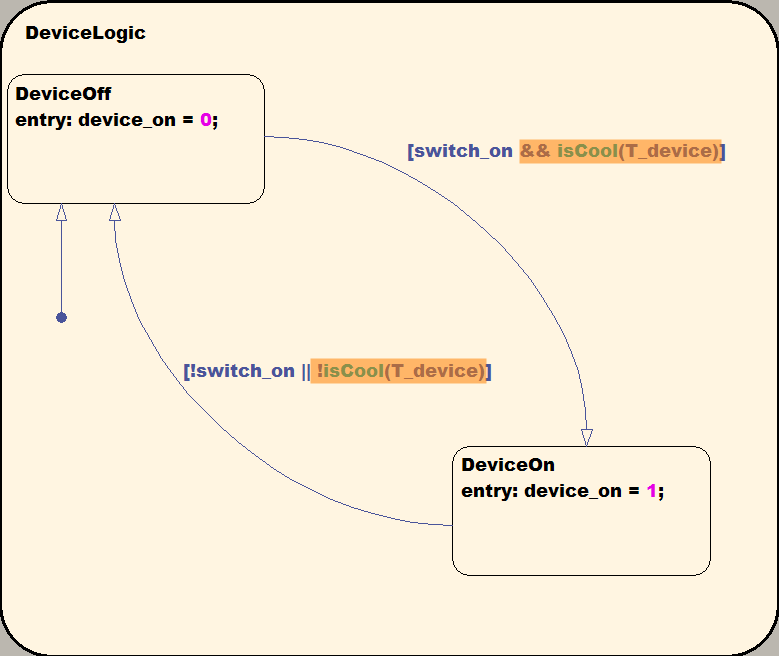
Our requirements specify that device operation is not critical for overall system performance (i.e. our CD player can be inactive yet we can still drive). It is more important that if the device begins overheating, the system is shut down before damage to the hardware occurs. In this particular problem, we don't want the device operating above 160° F.

[Back to Top](#_top)

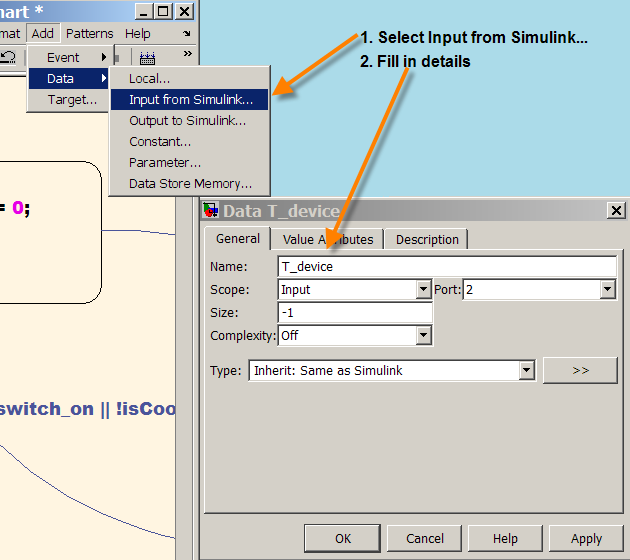
### 3.1 Expand on the Transition Conditions

Do the following:

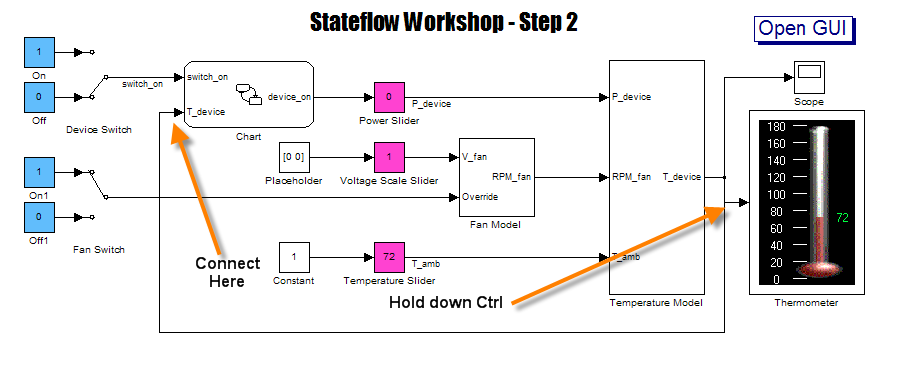
1. [Open the Step 2 model](matlab:Step_2) or continue working off your existing file
2. Expand the transition conditions between DeviceOn and DeviceOff by typing && isCool(T\_device) and || !isCool(T\_device)



1. Add a new data object, T\_Device
   1. Use the Model Explorer to add T\_Device and set its scope to be **Input**
   2. Alternatively, accomplish this from the Add Menu



1. Connect the Stateflow chart inport in Simulink
   1. Go the Simulink model
   2. Find the T\_device signal coming out of the temperature model
   3. Hold down Ctrl and left-click the signal wire
   4. Drag the mouse to the chart inport



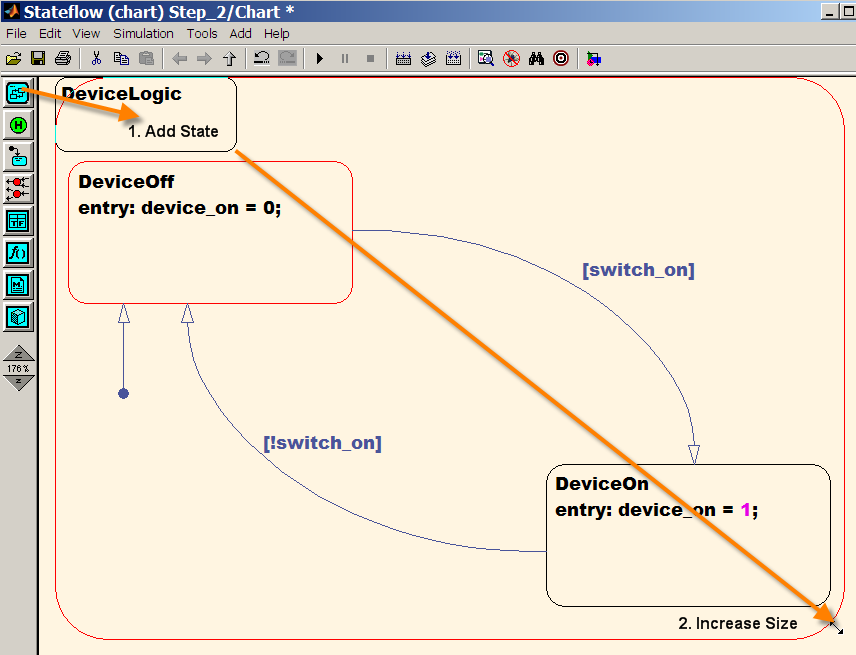
These steps establish a temperature dependence on our transition conditions. We will not transition from off to on anymore unless the device is sufficiently cool (even if the switch is turned on). Similarly, the device will automatically turn off when it gets too hot. The function that makes the distinction between cool and hot, isCool, now needs to be defined. First, we'll need to free up space in our chart to make room.

[Back to Top](#_top)

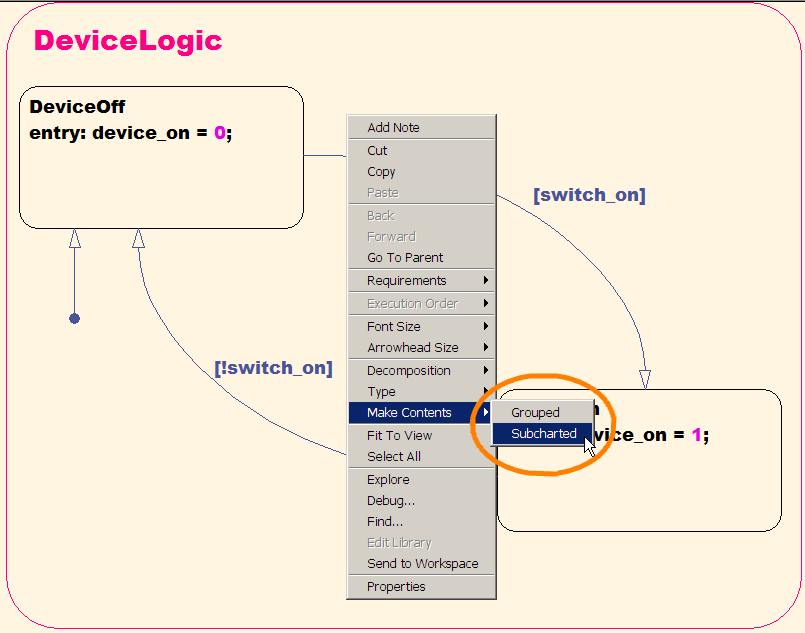
### 3.2 Create a Stateflow Subchart

DO the following:

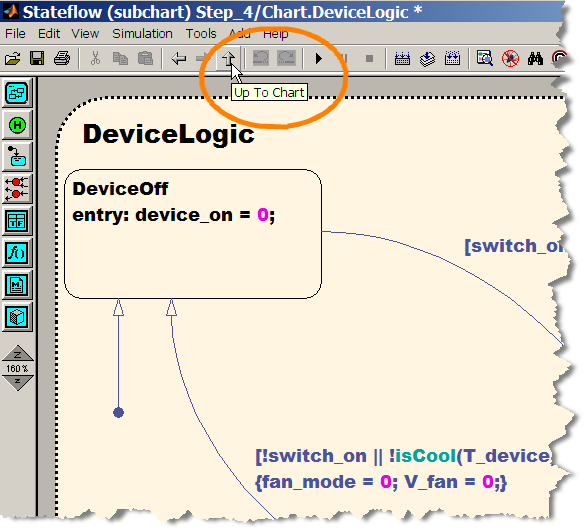
1. Add a new state to the chart and name it DeviceLogic
2. Grab the edge of the state and resize it to contain all other chart contents



1. Right-click the white space inside DeviceLogic and select Make Contents -> Subcharted



1. Double-click the state to check the contents
2. Click the "Up To Chart" button to return to the top level of the Stateflow chart



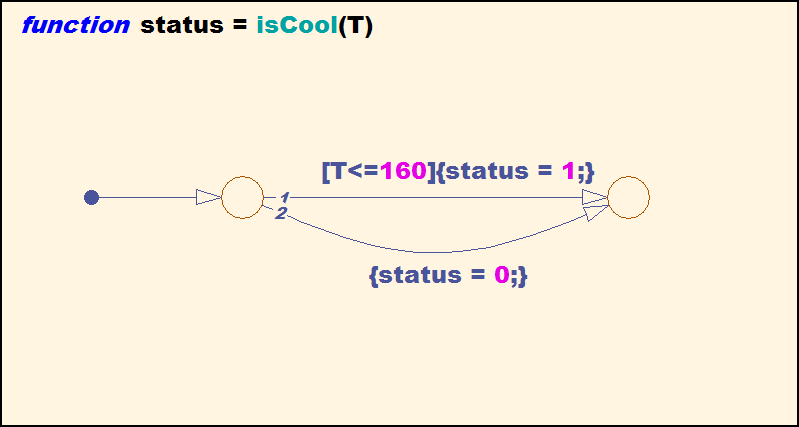
Like Simulink subsystems, creating subcharts is useful to organize our Stateflow logic. As we continue adding features to our chart, the DeviceLogic is neatly partitioned. Note that Stateflow doesn't open a new window when you explore a subchart. Instead, it recycles a single editing window. The "Up To Chart" button is used to return to a higher level of the chart. The Esc key is a shortcut for this button.

[Back to Top](#_top)

### 3.3 Create a Graphical Function

Do the following:

1. From the left sidebar, add a function to the Stateflow chart below DeviceLogic
2. Define the function as status = isCool(T)
3. Make the function Subcharted
4. Create a Stateflow chart like the figure below
   1. Add two Connective Junctions
   2. Attach a Default Transition to one of the Junctions
   3. Define two transitions from the default junction to the other
   4. Label transition 1 [T<=160]{status = 1;}
   5. Label transition 2 {status = 0;}



You have just created a Stateflow graphical function! Such functions are useful for creating and visualizing decision logic. In the process, we have touched on a number of new concepts. First is the Connective Junction. This object is useful for defining decision logic. In our case, we enter the first junction based on the default transition and have two paths to follow.

We take the path denoted by the 1 when our transition condition of T <= 160 is met. This transition is tested first (hence the 1). If the condition isn't met, Stateflow will see if transition 2 is valid. In our problem, we haven't specified a condition to follow the second path, which means that Stateflow will follow it. It is similar to the else statement in if/then/else constructs. As a rule of thumb, defining the last transition from a junction without a transition condition is a good idea. Otherwise, Stateflow might get stuck at the condition junction.

In the Stateflow language, text within curly brackets is known as a condition action. We have defined condition actions on both pathways. Our function output, status, is set to 0 or 1 depending on the pathway taken.

Note that if we parse the chart, Stateflow doesn't complain about the new variables we have introduced, T and status. If we check the Model Explorer, we will find that the variables have already been defined. They are local to the function isCool and cannot be used elsewhere in the chart.

[Back to Top](#_top)

### 3.3 Test the Model

Do the following:

1. Start the simulation
2. Turn the device on
3. Increase the Usage Level and/or Ambient Temperature until the device reaches 160 degrees
4. Observe the DeviceLogic switching to DeviceOff and the temperature oscillating around 160 degrees

The device now shuts off when the temperature reaches 160. So long as the switch stays on though, the device turns back on a few seconds later when the device cools. This design could certainly be improved upon, but we'll leave it as a problem to solve later.

[Back to Top](#_top)

## 4.0 Add On/Off Fan Logic

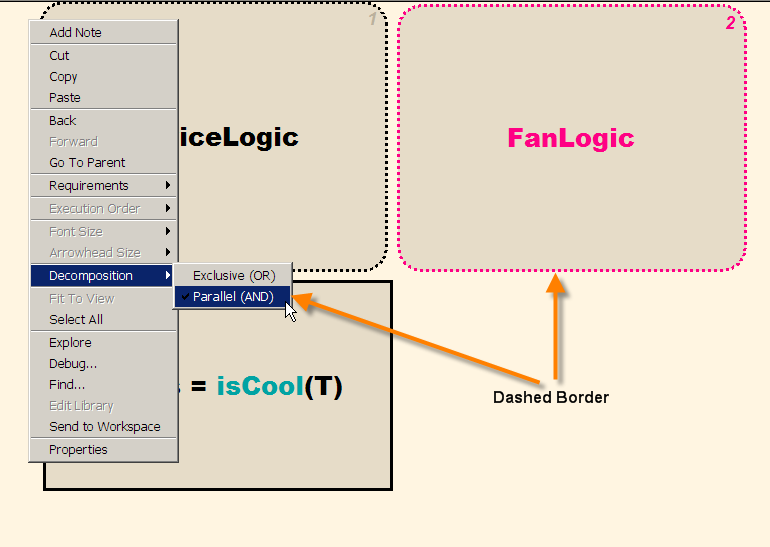
The Simulink model developers have incorporated a fan model, but without control logic, it cannot be activated. In this step, we will add to our Stateflow chart so that we can make use of a fan.

[Back to Top](#_top)

### 4.1 Add a Parallel State

Do the following:

1. [Open the Step 3 model](matlab:Step_3), or continue working off your existing file
2. Double-click the Stateflow chart
3. Add a new state next to DeviceLogic
   1. Name the state FanLogic
   2. Make the state Subcharted (right-click Make Contents)
4. Right-click on the chart background and select Decomposition -> Parallel



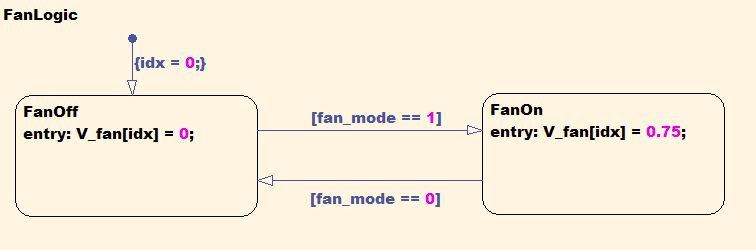
Note that when we changed the chart to have parallel decomposition, the border of DeviceLogic and FanLogic changed style to a dash. It is possible (necessary in fact) to have both of these subcharts running simultaneously. If you explore the contents of device logic, you will see that the on/off states remain exclusive. In a similar capacity, we are going to establish our initial fan control logic.

[Back to Top](#_top)

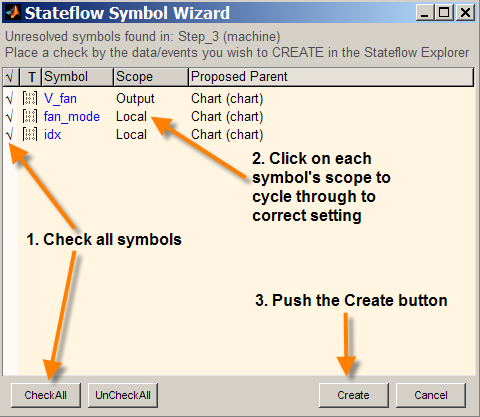
### 4.2 Define Fan States

Do the following:

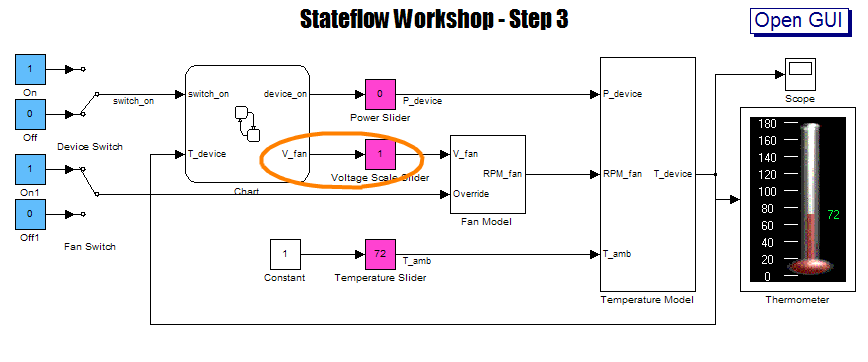
1. Double-click on FanLogic
2. Create a chart like the figure below
   1. Add two states: FanOff and FanOn
   2. Create transitions from off to on and on to off
   3. Define transition conditions [fan\_mode == 1] and [fan\_mode == 0] for the transitions
   4. Define entry actions V\_fan[idx] = 0.0; and V\_fan[idx] = 0.75; for the states
   5. Connect a Default Transition to FanOff and give it a transition condition {idx = 0;}



1. Define Data Objects V\_fan with scope **Output**, fan\_mode with scope **Local**, and idx with scope **Local**
   1. Try this with the Model Explorer
   2. Alternatively, try this from the Add Menu
   3. Alternatively, try to start your simulation, observe an error, and use the Stateflow Symbol Wizard to define the data



1. In the Simulink model, delete the Placeholder block, and connect the Chart outport to the Voltage Scale Slider



Our fan model in Simulink relies on voltage as an input to determine the fan speed in rotations per minute. When the fan is turned on, the Stateflow chart will now supply 0.75 volts to that model. The variable fan\_mode isn't necessary as output of the system, hence we leave it as internal to the chart.

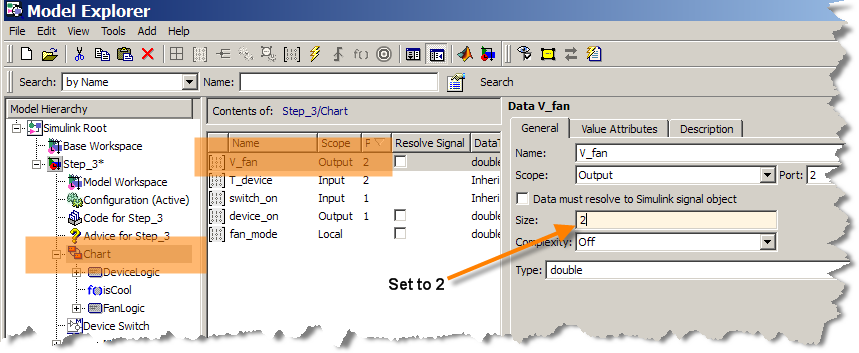
Do you notice the [idx] following our V\_fan settings? We are anticipating V\_fan to be a multi-element array, and are assigning voltage to the first index. Unlike MATLAB and Simulink, Stateflow uses square brackets to define array elements. Indexing begins at 0 instead of 1 by default, hence we set our variable idx to zero. This can be changed through the Model Explorer.

[Back to Top](#_top)

### 4.3 Specify Variable Properties in the Model Explorer

Do the Following:

1. Open the Model Explorer with Ctrl+H
2. Click Chart in the Model Hierarchy
3. Click V\_fan in Contents
4. In the General Tab, set the Size of V\_fan to 2



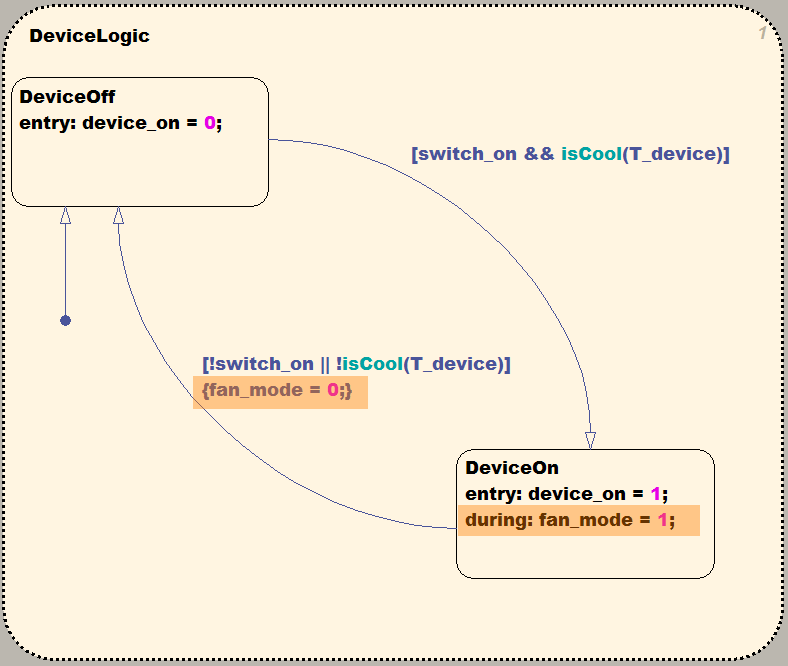
The fan model is expecting an input of size 2, and so we need to specify it here. Now to define the value of fan\_mode.

[Back to Top](#_top)

### 4.4 Add a State During Action

Do the Following:

1. Go to the DeviceLogic subchart
2. Add a new line to DeviceOn that says during: fan\_mode = 1;
3. Add a condition action statement to the on-to-off transition: {fan\_mode = 0;}



The during statement will set the fan\_mode variable to 1 every time the Stateflow chart is called and the DeviceLogic chart is in the DeviceOn state. Strictly speaking, we could maintain this as an entry command. The reason for doing it this way will become apparent in Step 4.

The condition action statements ensure that the fan is turned off when the device is off. The fan\_mode is set to 0 (causing the FanLogic state to change) as is the fan voltage. Setting fan\_mode to 0 should be sufficient given that the FanOff entry condition zeros the voltage, but we set it in DeviceLogic for extra assurance.

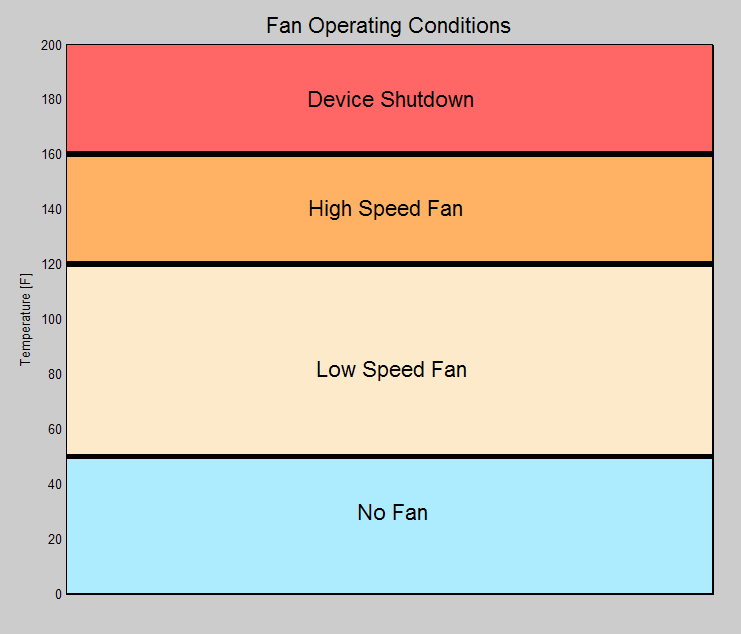
Run the simulation to observe that the device is cooler than before when activated. The device is at 107 degrees instead of 152 when the Usage Level is at 20%. If we increase the Usage Level to 50% though, the device reached the 160 degree limit.

[Back to Top](#_top)

## 5.0 Add a Second Fan Speed

The temperature of our device is improving as we add more sophistication to our Stateflow chart. In this next step, we'll add a second fan setting. By increasing the voltage, the RPM will increase and our device will be able to operate at higher usage levels.

The following design specifications have been supplied to us.



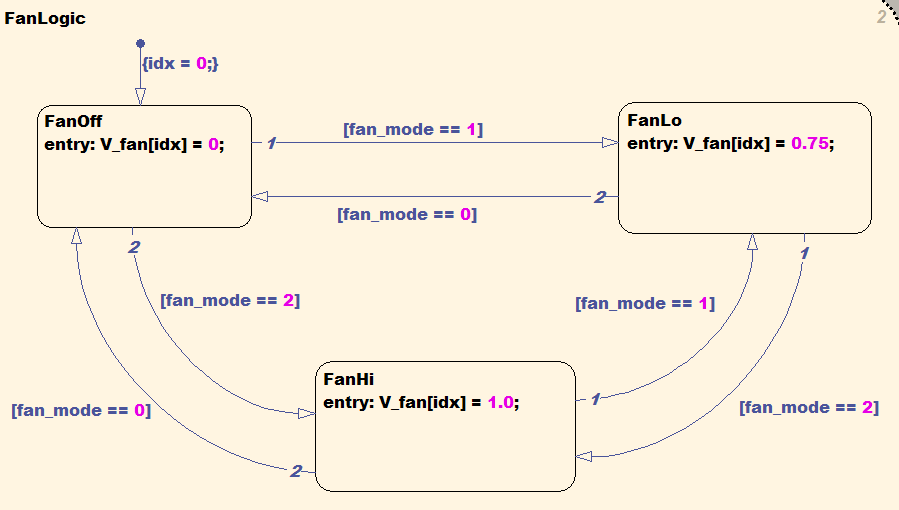
When the equipment is between 50 and 120 degrees, we are to use the low speed fan setting of 0.75 volts that we have already developed. When the temperature exceeds 120 degrees, the fan voltage needs to be increased to 1.00 volts. In situations where the ambient temperature is exceptionally cold, there is no need for the fan to be on. If the device is below 50 degrees, requirements specify that the fan is to be turned off. In order to accommodate this design, we will need to add a new state to our FanLogic.

[Back to Top](#_top)

### 5.1 Create a Third Fan State

Do the following:

1. [Open the Step 4 model](matlab:Step_4), or continue working off your existing file
2. Open the FanLogic subchart
3. Rename FanOn to FanLo
4. Add a third state
   1. Name it FanHi
   2. Specify the entry action to V\_fan[idx] = 1.0;
5. Add transitions to and from FanLo to FanHi
6. Add transitions to and from FanOff to FanHi
7. Define conditions for the new transitions as specified in the figure



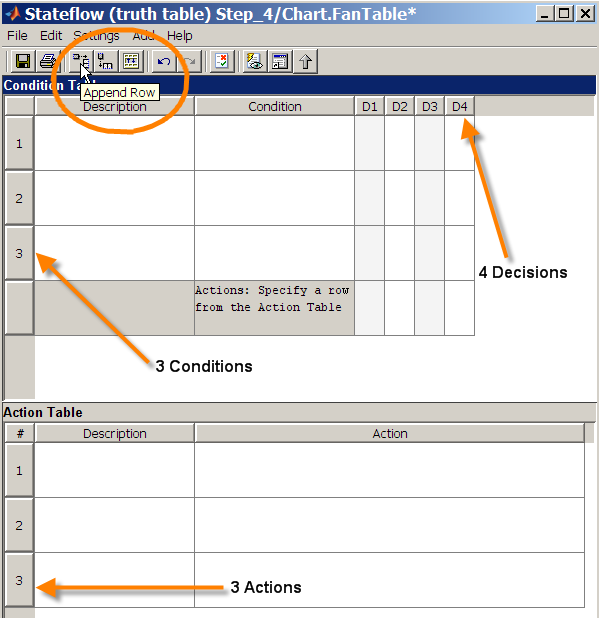
Along with our third fan state, we have defined a new possibility to the value of fan\_mode set to 2. Our during statement in DeviceLogic simply sets the fan\_mode to 1; this will need to modified to incorporate the more sophisticated, temperature-dependent logic. We could solve this problem by calling a second graphical function similar to isCool. Instead, let's try something new and make use of a truth table.

[Back to Top](#_top)

### 5.2 Create a Truth Table

Do the following:

1. Go to the top level of the Stateflow chart
2. Find Truth Table on the left sidebar and add it to the chart below FanLogic
3. Name the table FanTable(T)
4. Double-click on the table to open a the truth table editor
5. Use the Append Row and Append Column buttons to create a 3x4 Condition Table and 3x1 Action Table (not counting Description and Condition columns)



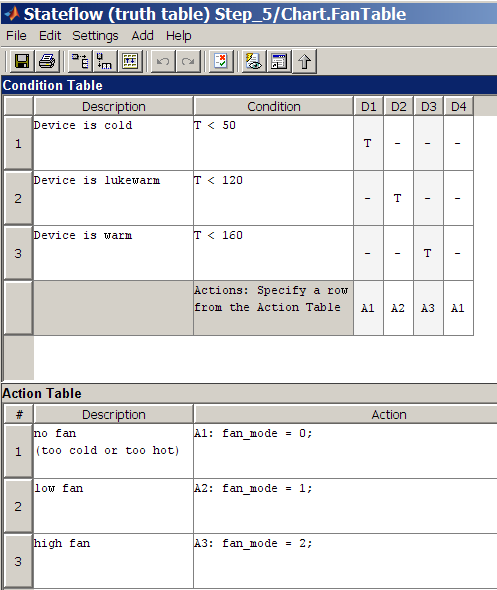
Truth tables are another method for specifying decision logic. Conditions on the system are specified in the second column of the Condition Table. In the decision columns to the left we can specify the condition one of three ways: true, false, or doesn't matter. The syntax for truth table condition states are T, F, and -.

[Back to Top](#_top)

### 5.3 Define Truth Table Logic

Do the following:

1. Define the three conditions as shown in the figure: T < 50, T < 120, and T < 160
2. Define the decisions
   1. Specify T < 50 True (T) in D1; set the action to A1
   2. Specify T < 120 True (T) in D2; set the action to A2
   3. Specify T < 160 True (T) in D3; set the action to A3
   4. Specify all remaining decision entries to Unspecified (-)
   5. Set the action to A1
3. Define the first row, second column of Action Table as A1: fan\_mode = 0;
4. Define the first row, second column of Action Table as A2: fan\_mode = 1;
5. Define the first row, second column of Action Table as A3: fan\_mode = 2;
6. Add condition and action descriptions as desired



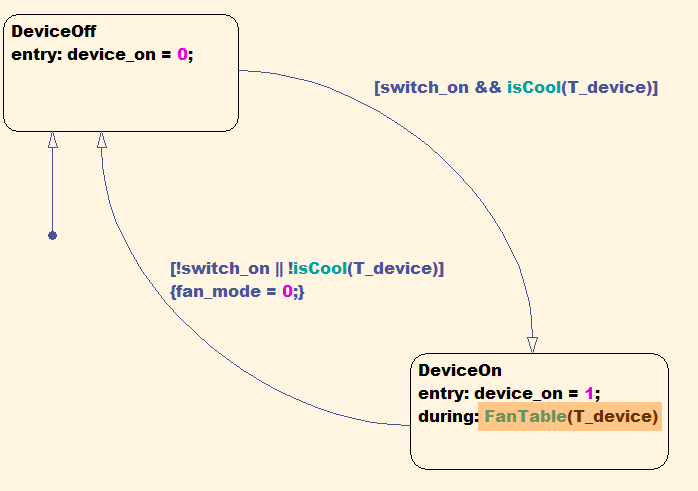
Truth Tables evaluate decisions sequentially. In our case, our first check is whether or not the temperature is less than 50 degrees. If it is, we don't care about our other temperature conditions. If we check Decision 2, then we already know the device is not less than 50 degrees and don't care about that condition. When assembled together, FanTable will now assign the proper value to fan\_mode based on the device temperature. In order for all this to come together, we'll need to do a couple more things.

[Back to Top](#_top)

### 5.4 Integrate Truth Table into Chart

Do the Following:

1. In DeviceLogic subchart, change the during statement to FanTable(T\_device)



By amending the during statement, we have declared that FanTable will be executed during every call to Stateflow when the device is on.

Go ahead and run the simulation, alter the parameters through the GUI, and observe the effect. You should find that all the fan state changes occur at the proper temperature transitions, but you may notice an unexpected problem. When the device exceeds 120 degrees, the high speed activates and can drive the temperature back below 120. At this point, our controller switches to the low speed setting, the device heats up again, and we return to the high speed state. Under certain operating conditions, we can rapidly toggle between high and low (and similarly low and off) indefinitely.

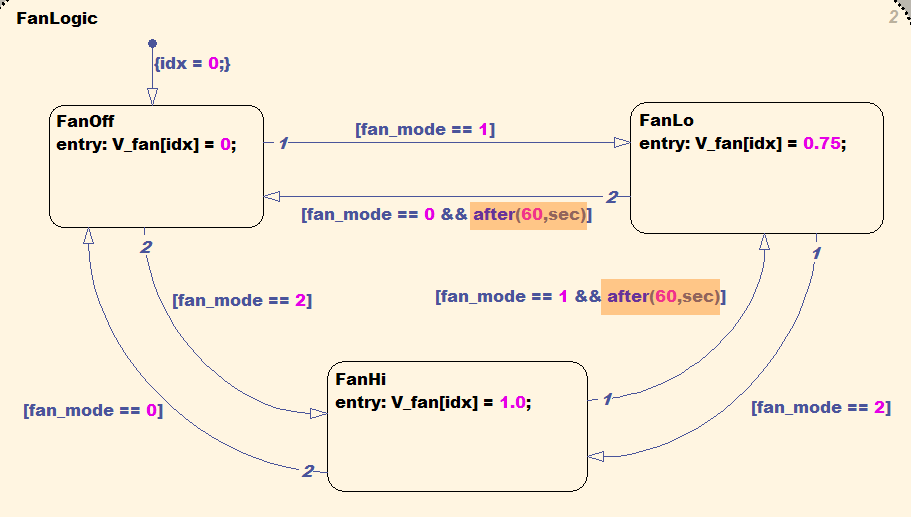
This problem is similar to the device on/off switching occurring when the device reaches 160 degrees, though not quite as detrimental. The switching between fan modes may be acceptable, but we need to limit the frequency in which it occurs.

[Back to Top](#_top)

### 5.5 Add Time-dependent Logic

Do the Following:

1. Open the FanLogic subchart
2. Add a delay term to the High-to-Low and Low-to-Off transition conditions
3. Insert && after(60,sec)



After is another feature of the Stateflow language. It enables you make declarations relative to occurrences in the chart. In our problem, we are telling Stateflow to "change from the higher to lower fan speed one minute after the temperature threshold is reached." This ensures that the cycling takes as least one minute to complete. Try running the model at slower settings in the GUI to observe the new behavior.

In older versions of Stateflow, the "sec" phrase is not recognized. In our case, we have set up a local variable ts to record when the previous transition took place. Another approach might be to use the "after(n,tick)" construct. Tick is a special phrase in Stateflow that relates to the number of times the chart has been accessed. If Stateflow was sampled at 5 Hz for instance, then we could use "after(300,tick)" in our chart.

[Back to Top](#_top)

## 6.0 Add a Backup Fan

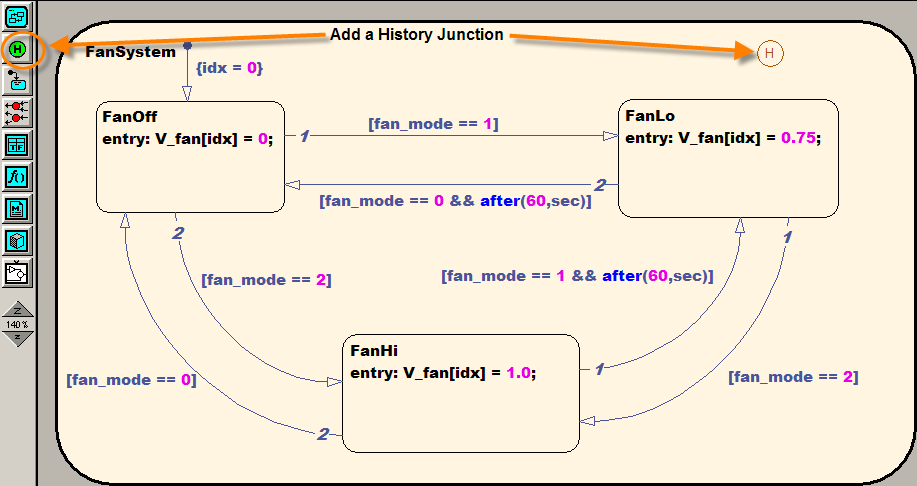
Now that we have our fan and device logic more or less in place, it's time to add some system redundancy. In our case, the device is installed with a backup cooling fan. It lies dormant under most circumstances. When the primary fan breaks however, the backup fan needs to activate and take over the job of cooling the device.

[Back to Top](#_top)

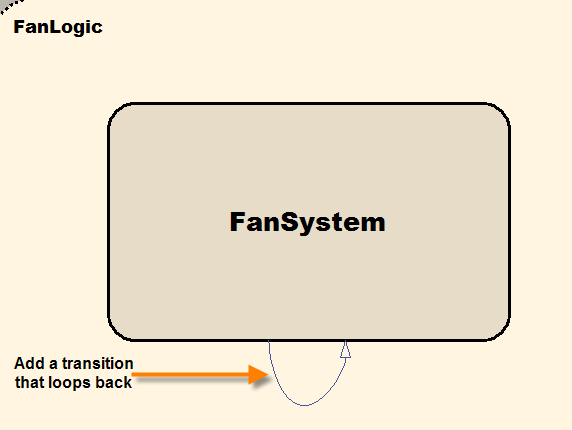
### 6.1 Create a Second Fan Subchart

Do the following:

1. [Open the Step 5 model](matlab:Step_5) or continue working off your existing file
2. Go to the FanLogic subchart
3. Place all existing contents of FanLogic in a new subchart titled FanSystem
   1. Add a new state and name it FanSystem
   2. Resize the state to fit all other chart contents within FanSystem
   3. Right-click and make FanSystem contents subcharted
   4. Shrink the size of FanSystem
4. Add a History Junction to FanSystem



1. Go back up to the FanLogic level and create a transition that goes from FanSystem back to itself



We will make use of our idx variable to activate the backup fan when a failure occurs. The History Junction enables Stateflow to remember which of the three states FanSystem was in (FanOff, FanLo, or FanHi) prior to the transition. This way, the Backup Fan will start in the same state that the Primary Fan was in before the failure.

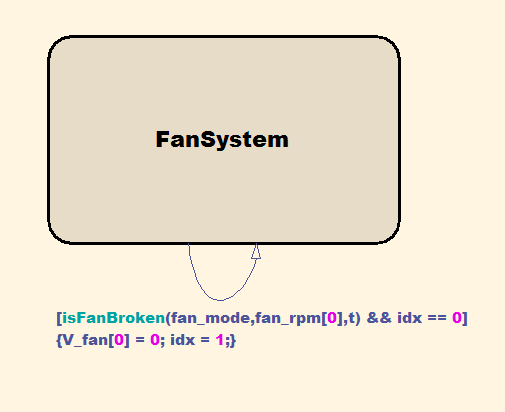
Note that we could have copied our FanSystem into a separate chart, resulting in a Primary and Backup system inside FanLogic. Instead, we choose to take advantage of the ability for transitions to loop back on their departing state. This will make management of the design easier (only place to have to make changes) and result in more efficient C code (no duplicate instruction sets).

[Back to Top](#_top)

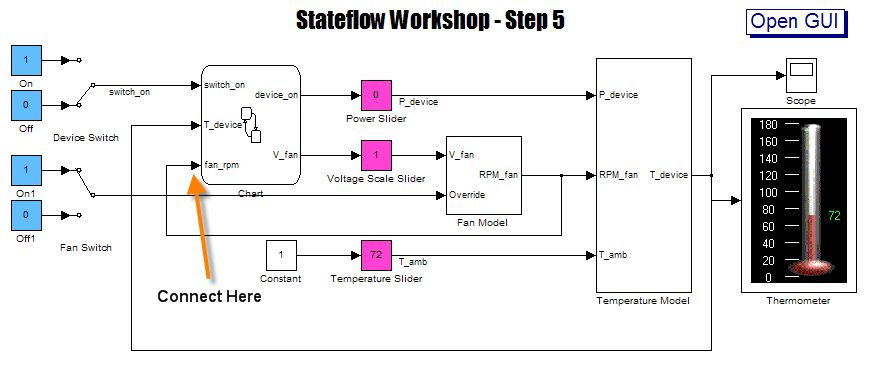
### 6.2 Define Fan Transitions

Do the Following:

1. Define the transition condition (square brackets) as isFanBroken(fan\_mode,fan\_rpm[0],t) && idx == 0
2. Define the condition action (curly brackets) as V\_fan[0] = 0.0; idx = 1;



1. Define the new variable fan\_rpm with scope **Input** via the Model Explorer, Add Menu, or Symbol Wizard
2. Go to the Simulink model and connect the RPM\_fan to the chart inport



By default, we use the primary fan. If the fan breaks, then we will use the Backup Fan, setting the voltage of that fan to the current primary fan value. Note that we have not added any logic to handle failure of the Backup Fan. We could expand this chart to include additional logic to check if the Primary Fan has been fixed. Feel free to try this on your own after the class.

We have defined a new function isFanBroken in our transition condition. This could be a graphical function or truth table like the ones we constructed in Steps 2 and 3. Instead, we are going to rely on Embedded MATLAB.

[Back to Top](#_top)

### 6.3 Create an Embedded MATLAB Function

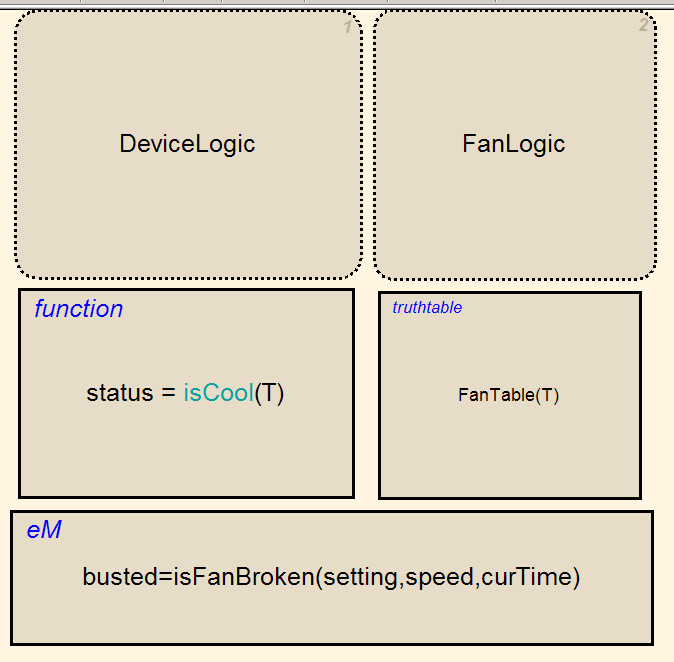
Do the Following:

1. Go to the top level of the Stateflow chart
2. Add an Embedded MATLAB block from the left sidebar (don't worry with naming it)
3. Double-click on the Embedded MATLAB block
4. Make your Embedded MATLAB code look like the following:

function busted = isFanBroken(setting,speed,curTime)

busted = checkFailure(setting,speed,curTime);

When you complete entering the function details and close the Embedded MATLAB editor, your chart should look something like this:



Embedded MATLAB provides a different approach for defining logic inside Stateflow (and also Simulink). In certain cases, you may not prefer graphical representations of the logic. eML is a method of describing your logic with traditional MATLAB coding approaches. This is especially useful in situations where existing MATLAB code is available. In our project, a generic hardware failure algorithm has been adapted for the fan failure logic.

If you open up your eML function, right click on checkFailure, and open the selection, you notice the %#eml pragma on the function line. This specifies checkFailure as an Embedded MATLAB function. In many cases, this will be the only change you will need to make to existing MATLAB code, though be careful. The Embedded MATLAB language is a subset of MATLAB; not all functions or coding constructs are available. For a list of available functions and best coding practices, refer to the [Embedded MATLAB Documentation](http://www.mathworks.com/access/helpdesk/help/toolbox/eml/index.html?/access/helpdesk/help/toolbox/eml/ug/bq37agh.html).

If you desire full MATLAB language functionality, you can make [extrinsic MATLAB function declarations](http://www.mathworks.com/access/helpdesk/help/toolbox/eml/ug/eml.extrinsic.html). eML is ordinarily compiled into C code during Simulink execution. Extrinsic functions are not compiled and instead call MATLAB directly. While handy for preliminary Stateflow design, extrinsic functions cannot be used when you desire deployment of your charts on embedded hardware (since you will be unable to call MATLAB).

Run the simulation now and switch on the Primary Fan Failure radio button in the GUI. Inside Stateflow, you can observe the FanSystem transition highlight blue when you activate a fan failure for the first time.

[Back to Top](#_top)

## 7.0 Add a Warning Event

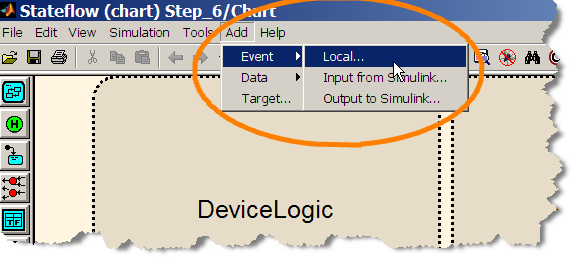
Now we have redundancy built into our system, but shouldn't we alert the operator that something has gone wrong? In this step, we introduce the concept of Stateflow events. Events signify when things happen. They can be external input from Simulink or generated internally. Events can in turn be output to the Simulink model as needed. In our case, we will create an internal event.

[Back to Top](#_top)

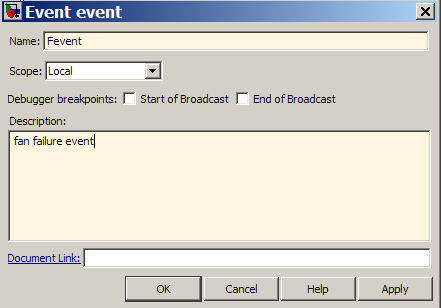
### 7.1 Create an Event

Do the Following:

1. [Open the Step 6 model](matlab:Step_6) or continue working off your existing file
2. Go to the Add -> Event -> Local in the pulldown menu



1. Name the event "Fevent"



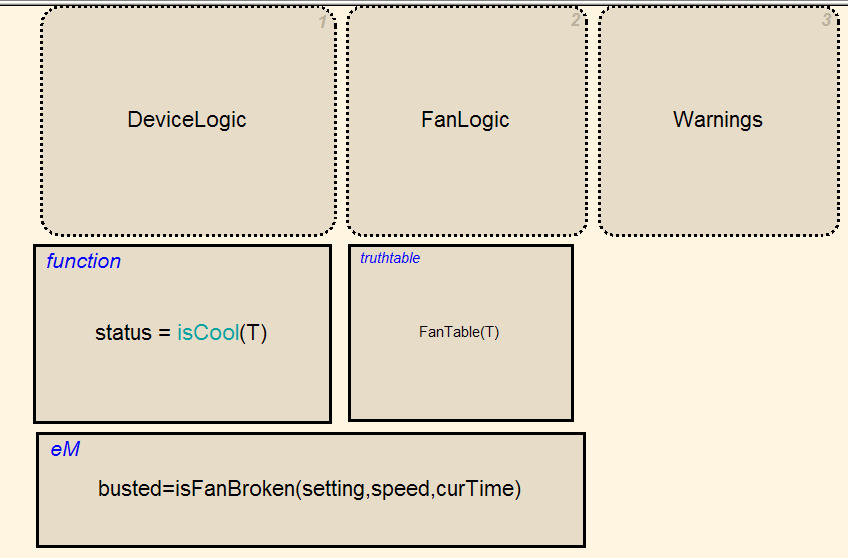
We could have set up our Fevent logic in Stateflow first and defined it later similar to the way we've done things in previous steps. In this approach, we define it in advance. It enables us to define additional settings on events and data (such as the Description) that we wouldn't have access to in the Symbol Wizard.

[Back to Top](#_top)

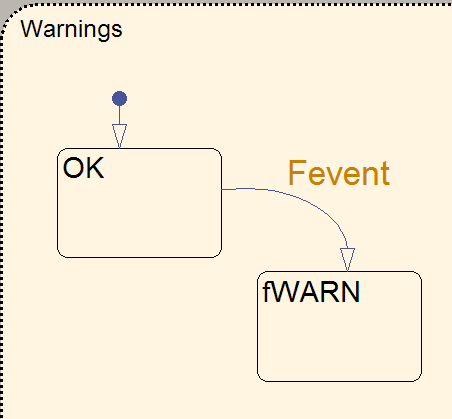
### 7.2 Add a Warning Subchart

Do the Following:

1. Add a new state and name it Warnings
2. Right-click and make the contents subcharted



1. Go into Warnings and create a chart as shown in the figure
   1. Create 2 states: OK and fWARN
   2. Add a default transition to OK
   3. Add a transition from OK to fWARN
   4. Label the transition Fevent



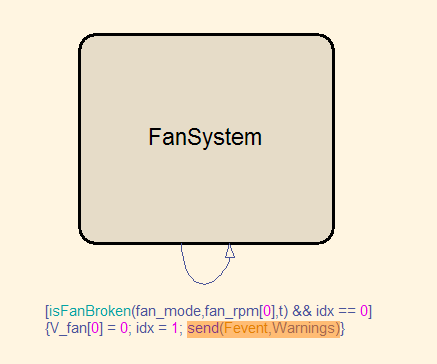
Note that Fevent is yellow on the transition and that we did not put Fevent in brackets. This is because events are different than conditions. A complete description of the differences cannot be covered in the workshop, but we will make use of one particular event capability in our model.

[Back to Top](#_top)

### 7.3 Broadcast an Event

Do the Following:

1. Go to the FanLogic subchart
2. Add send(Fevent,Warnings) to FanSystem’s transition condition action



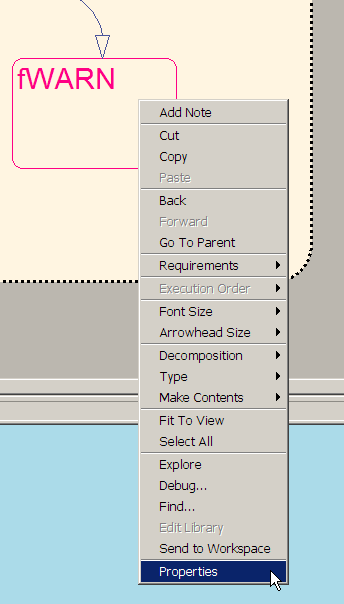
Fevent now occurs when the transition from to the Backup Fan takes place. The send statement broadcasts Fevent to the Warnings subchart, which in turn will cause the transition from OK to fWARN. Our internal event design is complete, but how do we information back to Simulink?

[Back to Top](#_top)

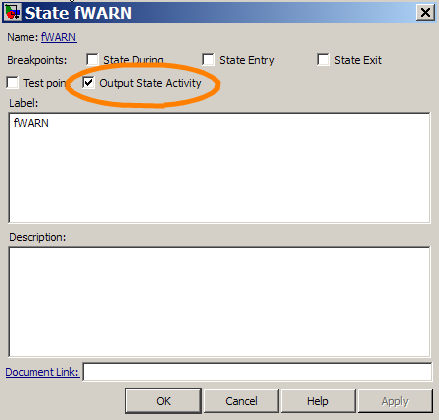
### 7.4 Add a State Activity Output

Do the Following:

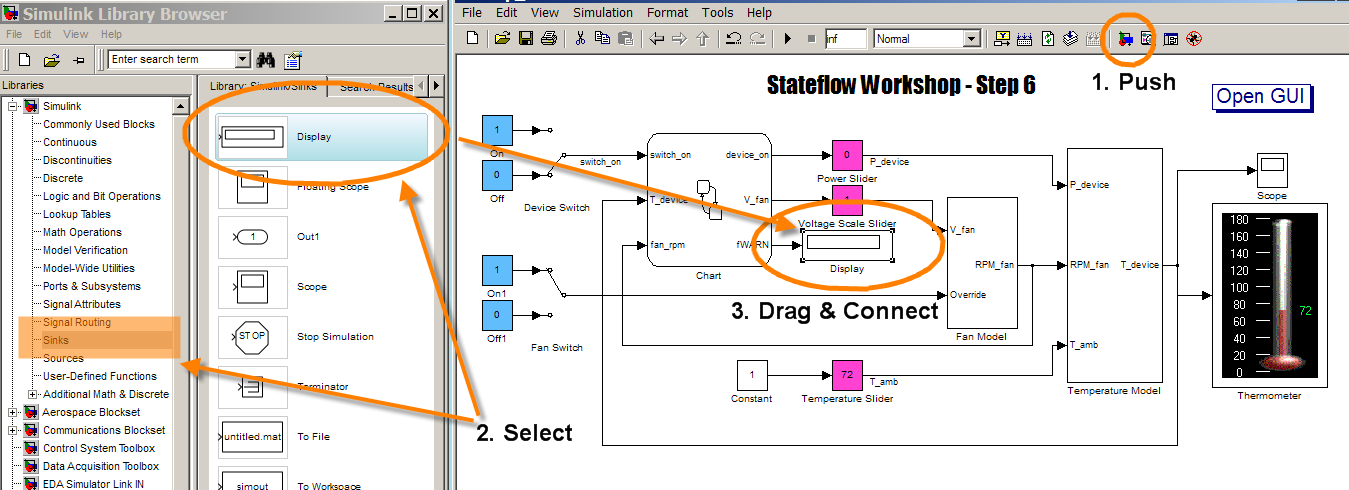
1. Right-click on fWARN and select Properties



1. Check the Output State Activity box



1. Go to Simulink and open the Library Browser
2. Under Sinks, drag a Display block into the model
3. Connect the Chart outport fWARN to the Display



Stateflow has the ability to output the status of states directly through the Properties dialogue. Note that we have created an output to the chart without defining a new Stateflow variable. When the state of the chart is in fWARN, the output will be a boolean true (and a false otherwise). Run your model and verify that causing a primary fan failure causes the display block to show a 1. The [Step 7 model](matlab:step_7) has replaced the Display block with an LED from the Gauges Blockset. Feel free to try this out.

[Back to Top](#_top)

## 8.0 Quiz Time!

It's time to take of the training wheels and apply all that you've learned about Stateflow up to this point. Remember in [Section 3](#_3.3_Test_the) when we noted that the device cycles on and off indefinitely when it reaches 160 degrees? If this was a DVD player, we'd get to watch a few seconds of our movie followed by a few seconds of blank screen. The problem can't be ignored anymore; we need to improve our control logic.

There is no single right answer to this problem. Ask yourself how you would want the device to perform if you were the user, and see if you can develop the logic in Stateflow to handle it.

Do the Following:

1. [Open the Step 7 model](matlab:Step_7) or continue working off your existing file
2. Amend the chart to prevent on/off cycling when reaching 160 degrees

One possible solution to the problem is available in the [Step 8](matlab:Step_8) model (or the [alternate Step 8](matlab:Step_8_alt) for earlier MATLAB releases). DeviceLogic has been altered somewhat from Step 7. Check it out to compare to your design or use it as a last resort if you're stuck. No cheating!

[Back to Top](#_top)

## Further Information

If you would like to learn more about Stateflow, here are some helpful links. Thank you for your time and best of luck as you begin to develop your own event-driven systems.

### 9.1 Product Information

[Stateflow Product Page](http://www.mathworks.com/products/stateflow/)

[Introduction to Stateflow Webinar](http://www.mathworks.com/products/demos/stateflow/introduction/index.html)

[Stateflow Demonstrations](http://www.mathworks.com/products/stateflow/demos.html)

[Stateflow Training](http://www.mathworks.com/services/training/courses/index.html?sec=stateflow&by=product&format_type=1)

[Stateflow User Stories](http://www.mathworks.com/products/stateflow/userstories.html)

[Stateflow Pattern Wizard Add On](http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=16304&objectType=File)

[Back to Top](#_top)

### 9.2 Documentation

[Stateflow Transition Action Types](http://www.mathworks.com/access/helpdesk/help/toolbox/stateflow/ug/f0-76034.html#f0-123134)

[Stateflow Temporal Logic](http://www.mathworks.com/access/helpdesk/help/toolbox/stateflow/index.html?/access/helpdesk/help/toolbox/stateflow/ug/f0-34084.html#brh91yy-9_2)

[Stateflow Events](http://www.mathworks.com/access/helpdesk/help/toolbox/stateflow/index.html?/access/helpdesk/help/toolbox/stateflow/ug/f7-7373.html#brdsb6f)

[Building Mealy and Moore Charts](http://www.mathworks.com/access/helpdesk/help/toolbox/stateflow/index.html?/access/helpdesk/help/toolbox/stateflow/ug/bqtktf3.html)

[Continuous Time Stateflow Charts](http://www.mathworks.com/access/helpdesk/help/toolbox/stateflow/index.html?/access/helpdesk/help/toolbox/stateflow/ug/brcw032.html)

[Back to Top](#_top)