

**Stateflow<sup>®</sup>**

Reference

**R2012a**

**MATLAB<sup>®</sup>  
& SIMULINK<sup>®</sup>**

## How to Contact MathWorks



[www.mathworks.com](http://www.mathworks.com) Web  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab) Newsgroup  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html) Technical Support



[suggest@mathworks.com](mailto:suggest@mathworks.com) Product enhancement suggestions  
[bugs@mathworks.com](mailto:bugs@mathworks.com) Bug reports  
[doc@mathworks.com](mailto:doc@mathworks.com) Documentation error reports  
[service@mathworks.com](mailto:service@mathworks.com) Order status, license renewals, passcodes  
[info@mathworks.com](mailto:info@mathworks.com) Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



The MathWorks, Inc.  
3 Apple Hill Drive  
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

*Stateflow*<sup>®</sup> Reference

© COPYRIGHT 2006-2012 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

### **Revision History**

March 2006	Online only	New for Version 6.4 (Release 2006a)
September 2006	Online only	Revised for Version 6.5 (Release R2006b)
September 2007	Online only	Rereleased for Version 7.0 (Release 2007b)
March 2008	Online only	Revised for Version 7.1 (Release 2008a)
October 2008	Online only	Revised for Version 7.2 (Release 2008b)
March 2009	Online only	Rereleased for Version 7.3 (Release 2009a)
September 2009	Online only	Revised for Version 7.4 (Release 2009b)
March 2010	Online only	Rereleased for Version 7.5 (Release 2010a)
September 2010	Online only	Rereleased for Version 7.6 (Release 2010b)
April 2011	Online only	Rereleased for Version 7.7 (Release 2011a)
September 2011	Online only	Rereleased for Version 7.8 (Release 2011b)
March 2012	Online only	Revised for Version 7.9 (Release 2012a)



## Function Reference

---

**1**

Object Retrieval .....	1-2
Chart Creation .....	1-3
Chart Input/Output .....	1-4
GUI .....	1-5
Help .....	1-6

## Functions — Alphabetical List

---

**2**

## Block Reference

---

**3**

## Index

---



# Function Reference

---

Object Retrieval (p. 1-2)

Get objects in Stateflow<sup>®</sup> hierarchy

Chart Creation (p. 1-3)

Create Stateflow charts and truth tables

Chart Input/Output (p. 1-4)

Read and write Stateflow charts

GUI (p. 1-5)

Launch tools for defining and debugging Stateflow objects

Help (p. 1-6)

Get help on using Stateflow software

## Object Retrieval

sfclipboard

Stateflow clipboard object

sfgco

Recently selected objects in chart

sfrout

Root object



## Chart Creation

`sfnew`

Create model containing empty  
Stateflow block

`stateflow`

Create empty chart

## **Chart Input/Output**

sfclose

Close chart

sfopen

Open existing model

sfprint

Print graphical view of charts

sfsave

Save chart in current folder

## GUI

`sfdebugger`

Open Stateflow Debugger

`sfexplr`

Open Model Explorer

`sflib`

Open Stateflow library window

## Help

`sfhelp`

[Open Stateflow online help](#)

# Functions — Alphabetical List

---

# sfclipboard

---

<b>Purpose</b>	Stateflow clipboard object
<b>Syntax</b>	<code>object = sfclipboard</code>
<b>Description</b>	<code>object = sfclipboard</code> returns a handle to the Stateflow clipboard object, which you use to copy objects from one chart or state to another.
<b>Examples</b>	<p>Copy the <code>init</code> function from the <code>Init</code> chart to the <code>Pool</code> chart in the <code>sf_pool</code> model:</p> <pre>sf_pool; % Get handle to the root object rt = sfroot; % Get handle to 'init' function in Init chart f1 = rt.find('-isa','Stateflow.EMFunction','Name','init'); % Get handle to Pool chart chP = rt.find('-isa','Stateflow.Chart','Name','Pool'); % Get handle to the clipboard object cb = sfclipboard; % Copy 'init' function to the clipboard cb.copy(f1); % Paste 'init' function to the Pool chart cb.pasteTo(chP); % Get handle to newly pasted function f2 = chP.find('-isa','Stateflow.EMFunction','Name','init'); % Reset position of new function in the Pool chart f2.Position = [90 180 90 60];</pre>
<b>See Also</b>	<code>sfgco</code>   <code>sfnew</code>   <code>sfroot</code>   <code>stateflow</code>
<b>Tutorials</b>	<ul style="list-style-type: none"><li>• “Copying Objects”</li><li>• “Quick Start for the Stateflow API”</li></ul>
<b>How To</b>	<ul style="list-style-type: none"><li>• “Getting a Handle on Stateflow API Objects”</li><li>• “Accessing the Chart Object”</li></ul>

**Purpose** Close chart

**Syntax**  
sfclose  
sfclose('chart\_name')  
sfclose('all')

**Description** sfclose closes the current chart.  
sfclose('chart\_name') closes the chart called 'chart\_name'.  
sfclose('all') closes all open or minimized charts. 'all' is a literal string.

**See Also** sfnew | sfoopen | stateflow

# sfdebugger

---

<b>Purpose</b>	Open Stateflow Debugger
<b>Syntax</b>	<code>sfdebugger</code> <code>sfdebugger('model_name')</code>
<b>Description</b>	<code>sfdebugger</code> opens the Stateflow Debugger for the current model. <code>sfdebugger('model_name')</code> opens the debugger for the Simulink® model called ' <i>model_name</i> '. Use this input argument to specify which model to debug when you have multiple models open.
<b>See Also</b>	<code>sfexplr</code>   <code>sfhelp</code>   <code>sflib</code>
<b>How To</b>	<ul style="list-style-type: none"><li>• “Using the Stateflow Debugger”</li></ul>



<b>Purpose</b>	Open Model Explorer
<b>Syntax</b>	<code>sfexplr</code>
<b>Description</b>	<code>sfexplr</code> opens the Model Explorer. A model does not need to be open.
<b>See Also</b>	<code>sfdebugger</code>   <code>sfhelp</code>   <code>sflib</code>
<b>How To</b>	<ul style="list-style-type: none"><li>• “Using the Model Explorer with Stateflow Objects”</li></ul>



- “Zooming a Chart Object with the API”

# sfhelp

---

<b>Purpose</b>	Open Stateflow online help
<b>Syntax</b>	sfhelp
<b>Description</b>	sfhelp opens the Stateflow online help in the MATLAB® Help browser.
<b>See Also</b>	sfdebugger   sfexplr   sfnew   stateflow

**Purpose** Open Stateflow library window

**Syntax** `sflib`

**Description** `sflib` opens the Stateflow library window. From this window, you can drag Chart and Truth Table blocks into Simulink models and access the Stateflow Examples Library.

**See Also** `sfdebugger` | `sfexplr` | `sfhelp` | `sfnew`

# sfnew

---

**Purpose** Create model containing empty Stateflow block

**Syntax**

```
sfnew
sfnew('chart_type')
sfnew('model_name')
sfnew('chart_type', 'model_name')
```

**Description** `sfnew` creates an untitled model with an empty chart that supports full semantics.

`sfnew('chart_type')` creates an untitled model that contains an empty block of type `chart_type`.

`sfnew('model_name')` creates a model called `model_name` with an empty chart that supports full semantics.

`sfnew('chart_type', 'model_name')` creates a model called `model_name` with an empty block of type `chart_type`.

**Input Arguments**

`chart_type`

Empty block to add to an empty model:

- |          |  |
|----------|--|
| -Classic | Use full chart semantics               |
| -Mealy   | Use only Mealy state machine semantics |
| -Moore   | Use only Moore state machine semantics |
| -TT      | Use a truth table                      |

`model_name`

Name of the model.

**Examples** Create a model called `MyModel` with an empty chart that uses only Mealy semantics:

```
sfnew(' -Mealy', 'MyModel')
```

---

Create a model called `MyModel` with an empty chart that uses only Moore semantics:

```
sfnew(' -Moore', 'MyModel')
```

## See Also

`sfhelp` | `sfprint` | `sfrout` | `sfsave` | `stateflow`

## How To

- “Creating an Empty State Chart”
- “Creating Mealy and Moore Charts”
- “Building a Model with a Stateflow Truth Table”

# sfopen

---

<b>Purpose</b>	Open existing model
<b>Syntax</b>	<code>sfopen</code>
<b>Description</b>	<code>sfopen</code> prompts you for a model file and opens the model that you select from your file system.
<b>See Also</b>	<code>sfclose</code>   <code>sfdebugger</code>   <code>sfexplr</code>   <code>sflib</code>   <code>sfnew</code>   <code>stateflow</code>



<b>Purpose</b>	Print graphical view of charts						
<b>Syntax</b>	<pre>sfprint sfprint(objects) sfprint(objects,format) sfprint(objects,format,output_option) sfprint(objects,format,output_option,print_entire_chart)</pre>						
<b>Description</b>	<p>sfprint prints the current chart to a default printer.</p> <p>sfprint(<i>objects</i>) prints all charts in <i>objects</i> to a default printer.</p> <p>sfprint(<i>objects,format</i>) prints all charts in <i>objects</i> in the specified <i>format</i> to a default printer.</p> <p>sfprint(<i>objects,format,output_option</i>) prints all charts in <i>objects</i> in the specified <i>format</i> to the file or printer specified in <i>output_option</i>.</p> <p>sfprint(<i>objects,format,output_option,print_entire_chart</i>) prints all charts in <i>objects</i> in the specified <i>format</i> to the file or printer specified in <i>output_option</i>. Prints a complete or current view of charts as specified in <i>print_entire_chart</i>.</p> <p>If the <i>format</i> argument is absent, sfprint generates a PostScript file to the default printer. If the <i>output_option</i> argument is absent, the name of the chart in the current folder becomes the output file name.</p>						
<b>Input Arguments</b>	<p><i>objects</i></p> <p>Objects containing charts to print:</p> <table border="0" style="margin-left: 2em;"> <tr> <td style="vertical-align: top;"><i>'literal_string'</i></td> <td>Path name of a chart, model, subsystem, or block</td> </tr> <tr> <td style="vertical-align: top;">gcb</td> <td>Command that specifies the current block of the model</td> </tr> <tr> <td style="vertical-align: top;">gcs</td> <td>Command that specifies the current system of the model</td> </tr> </table>	<i>'literal_string'</i>	Path name of a chart, model, subsystem, or block	gcb	Command that specifies the current block of the model	gcs	Command that specifies the current system of the model
<i>'literal_string'</i>	Path name of a chart, model, subsystem, or block						
gcb	Command that specifies the current block of the model						
gcs	Command that specifies the current system of the model						

## format

Format of the image to print:

'bitmap'	Save the chart image to the clipboard as a bitmap (for Windows® operating systems only)
'default'	Print image to your default printer
'eps'	Generate an encapsulated PostScript file
'eps'	Generate a color encapsulated PostScript file
'jpg'	Generate a JPEG file
'meta'	Save the chart image to the clipboard as a metafile (for Windows operating systems only)
'png'	Generate a PNG file
'ps'	Generate a PostScript file
'psc'	Generate a color PostScript file
'tif'	Generate a TIFF file

## output\_option

Name of the file or printer:

' <i>output_file_name</i> '	Send output to a file called <i>output_file_name</i>
'clipboard'	Copy output to the clipboard

'file'	Send output to a default file with the name <i>path_to_chart.file_extension</i> , such as <i>sf_pool_Init.jpg</i>
'promptForFile'	Prompt for file name interactively
'printer'	Send output to the default printer (use only with 'default', 'ps', or 'eps' formats)

`print_entire_chart`

View of charts to print:

1	Print complete charts (default)
0	Print current view of charts

## Examples

Print all charts in the current system as a PostScript file to your default printer:

```
sfprint(gcs)
```

---

Print the complete chart whose path is 'sf\_pool/Pool' in JPEG format:

```
sfprint('sf_pool/Pool', 'jpg')
```

---

Print the complete chart whose path is 'sf\_car/shift\_logic' in TIFF format using the name myFile:

```
sfprint('sf_car/shift_logic', 'tif', 'myFile')
```

---

# sfprint

---

Print the current view of all charts in the current system in PNG format using default file names:

```
sfprint(gcs, 'png', 'file', 0)
```

## See Also

`gcb` | `gcs` | `sfhelp` | `sfnew` | `sfsave` | `stateflow`

## How To

- “Printing Stateflow Charts”

<b>Purpose</b>	Root object
<b>Syntax</b>	<code>object = sfroot</code>
<b>Description</b>	<code>object = sfroot</code> returns a handle to the top-level object in the Stateflow hierarchy of objects. Use the root object to access all other objects in your charts when using the API.
<b>Examples</b>	<p>Zoom in on a state in your chart:</p> <pre>old_sf_car; % Get handle to the root object rt = sfroot; % Find the state with the name 'first' myState = rt.find('-isa','Stateflow.State','Name','first'); % Zoom in on that state in the chart myState.fitToView;</pre>
<b>See Also</b>	<code>sfclipboard</code>   <code>sfgco</code>
<b>Tutorials</b>	<ul style="list-style-type: none"> <li>• “Quick Start for the Stateflow API”</li> </ul>
<b>How To</b>	<ul style="list-style-type: none"> <li>• “Getting a Handle on Stateflow API Objects”</li> <li>• “Accessing the Chart Object”</li> </ul>

# sfsave

---

**Purpose** Save chart in current folder

**Syntax**

```
sfsave  
sfsave('model_name')  
sfsave('model_name','new_model_name')  
sfsave('Defaults')
```

**Description** sfsave saves the chart in the current model.

sfsave('model\_name') saves the chart in the model called 'model\_name'.

sfsave('model\_name','new\_model\_name') saves the chart in 'model\_name' to 'new\_model\_name'.

sfsave('Defaults') saves the settings of the current model as defaults. 'Defaults' is a literal string.

The model must be open and the current folder must be writable.

**Examples** Develop a script to create a baseline chart and save it in a new model:

```
bdclose('all');  
  
% Create an empty chart in a new model  
sfnew;  
  
% Get root object  
rt = sfroot;  
  
% Get model  
m = rt.find('-isa','Simulink.BlockDiagram');  
  
% Get chart  
chart1 = m.find('-isa','Stateflow.Chart');  
  
% Create two states, A and B, in the chart  
sA = Stateflow.State(chart1);
```

```
sA.Name = 'A';
sA.Position = [50 50 100 60];
sB = Stateflow.State(chart1);
sB.Name = 'B';
sB.Position = [200 50 100 60];

% Add a transition from state A to state B
tAB = Stateflow.Transition(chart1);
tAB.Source = sA;
tAB.Destination = sB;
tAB.SourceOClock = 3;
tAB.DestinationOClock = 9;

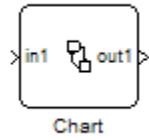
% Add a default transition to state A
dtA = Stateflow.Transition(chart1);
dtA.Destination = sA;
dtA.DestinationOClock = 0;
x = sA.Position(1)+sA.Position(3)/2;
y = sA.Position(2)-30;
dtA.SourceEndPoint = [x y];

% Add an input in1
d1 = Stateflow.Data(chart1);
d1.Scope = 'Input';
d1.Name = 'in1';

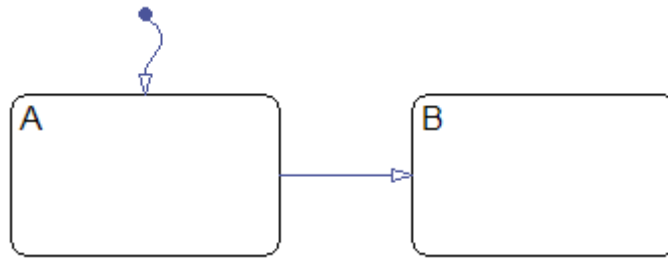
% Add an output out1
d2 = Stateflow.Data(chart1);
d2.Scope = 'Output';
d2.Name = 'out1';

% Save the chart in a model called "NewModel"
% in current folder
sfsave('untitled', 'NewModel');
```

Here is the resulting model:



Here is the resulting chart:



## See Also

`sfopen` | `sfclose` | `sfroot` | `sfnew` | `find`

## Tutorials

- “Quick Start for the Stateflow API”

## How To

- “Creating a MATLAB Script of API Commands”



**Purpose** Create empty chart

**Syntax** `stateflow`

**Description** `stateflow` creates an untitled model that contains an empty chart. The function also opens the Stateflow library window. From this window, you can drag Chart and Truth Table blocks into models or access the Stateflow Examples Library.

**See Also** `sflib` | `sfnew`

**How To**

- “Creating an Empty State Chart”



# Block Reference

---

# Stateflow Chart

---

**Purpose** Classic, Mealy, and Moore finite state machines for implementing control logic

**Library** Stateflow

**Description**



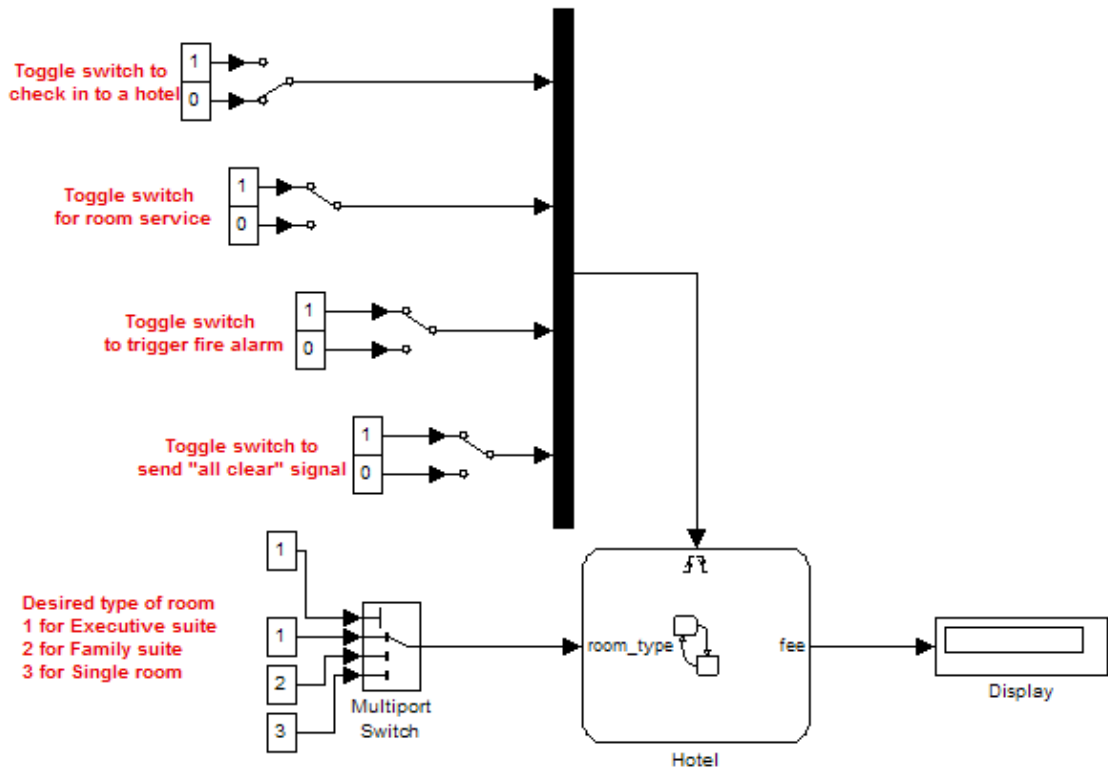
Chart

A *finite state machine* is a representation of an event-driven (reactive) system. In an event-driven system, the system responds by making a transition from one state (mode) to another. This action occurs in response to an event, as long as the condition defining the change is true.

A Stateflow chart is a graphical representation of a finite state machine, where *states* and *transitions* form the basic elements of the system. You can also represent stateless flow graphs. To add your control logic to a Simulink model, use a Stateflow block.

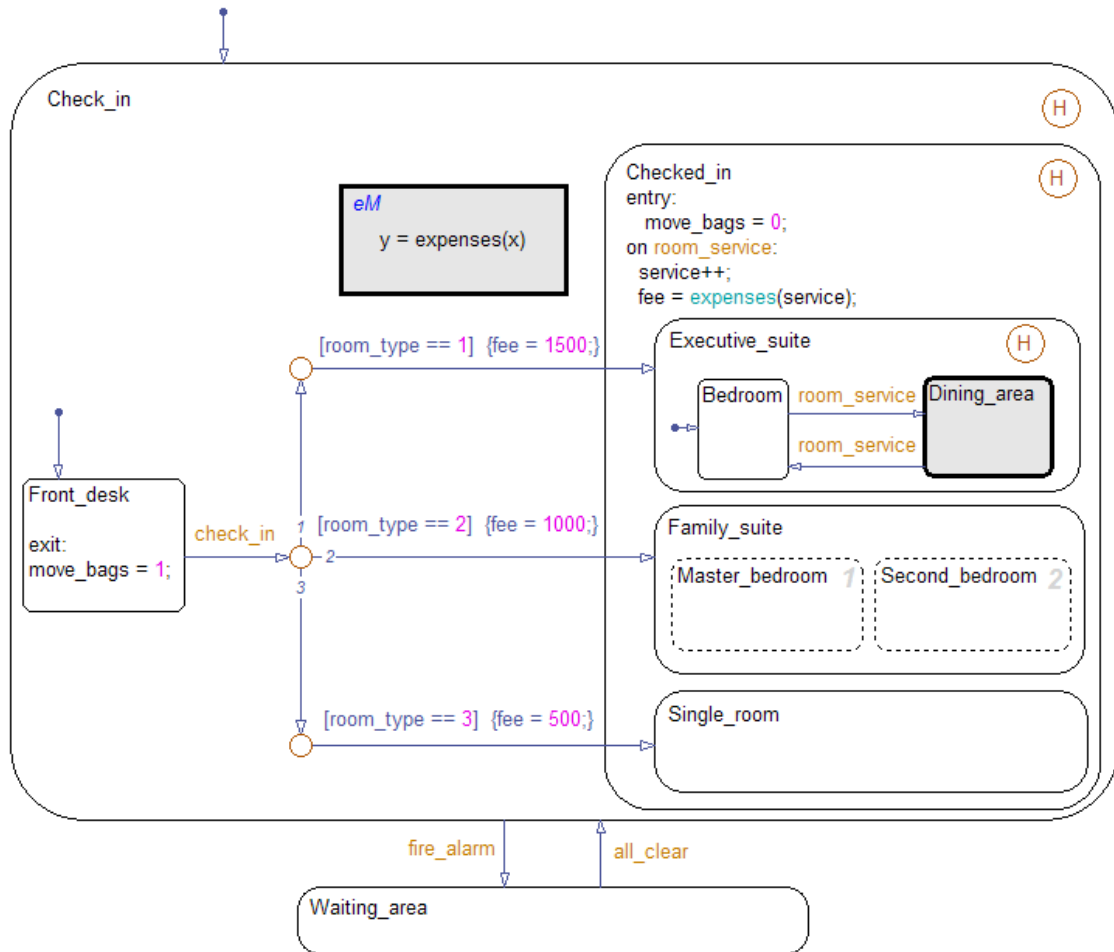
You can use Stateflow charts to control a physical plant in response to events such as a temperature or pressure sensor, or clock or user-driven events. For example, you can use a state machine to represent the automatic transmission of a car. The transmission has these operating states: park, reverse, neutral, drive, and low. As the driver shifts from one position to another, the system makes a transition from one state to another, for example, from park to reverse.

The following block diagram contains a Chart block named Hotel that responds to inputs from several Manual Switch blocks.



# Stateflow Chart

If you double-click the Chart block in the model, the chart appears.



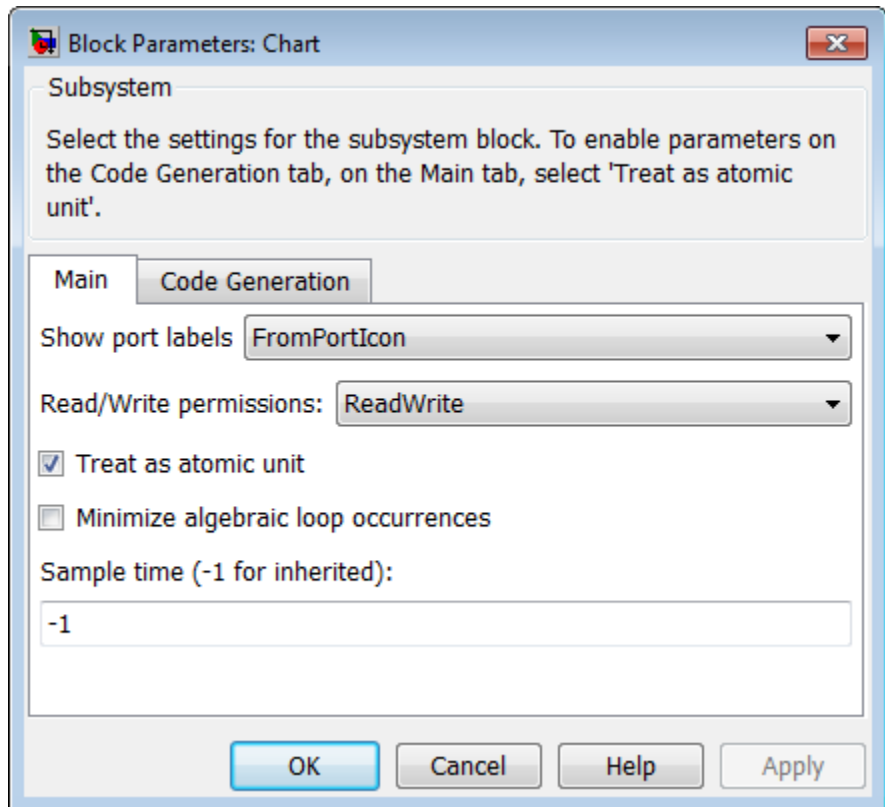
By default, the Chart block uses *Classic* chart semantics. You can also specify Mealy or Moore semantics for your state chart. For more information, see “Building Mealy and Moore Charts”.

## Data Type Support

The Chart block accepts inputs of any type including two-dimensional matrices, fixed-point data, and enumerated data. Floating-point inputs pass through the block unchanged. Boolean inputs are treated as uint8 signals.

You can declare local data of any type or size.

## Parameters and Dialog Box



For a description of the block parameters, see the Subsystem block reference page in the Simulink documentation.

# Stateflow Chart

---

## Characteristics

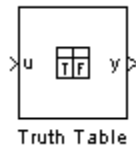
Direct Feedthrough	Yes
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	N/A
Dimensionalized	Yes
Zero-Crossing Detection	Yes, if enabled for continuous-time systems  For more information, see “Modeling Continuous-Time Systems in Stateflow Charts” in the Stateflow documentation.



**Purpose** Represent logical decision-making behavior with conditions, decisions, and actions

**Library** Stateflow

**Description**



The Truth Table block is a truth table function that uses only MATLAB action language. Use this block when you want to use truth table logic directly in a Simulink model. This block requires a Stateflow license.

When you add a Truth Table block directly to a model instead of calling truth table functions from a Stateflow chart, these advantages apply:

- It is a more direct approach, especially if your model requires only a single truth table.
- You can define truth table inputs and outputs to have inherited types and sizes.

The Truth Table block works with a subset of the MATLAB language that is optimized for generating embeddable C code. This block generates content as MATLAB code. As a result, you can take advantage of other tools to debug your Truth Table block during simulation. For more information, see “Debugging a MATLAB Function in a Chart”.

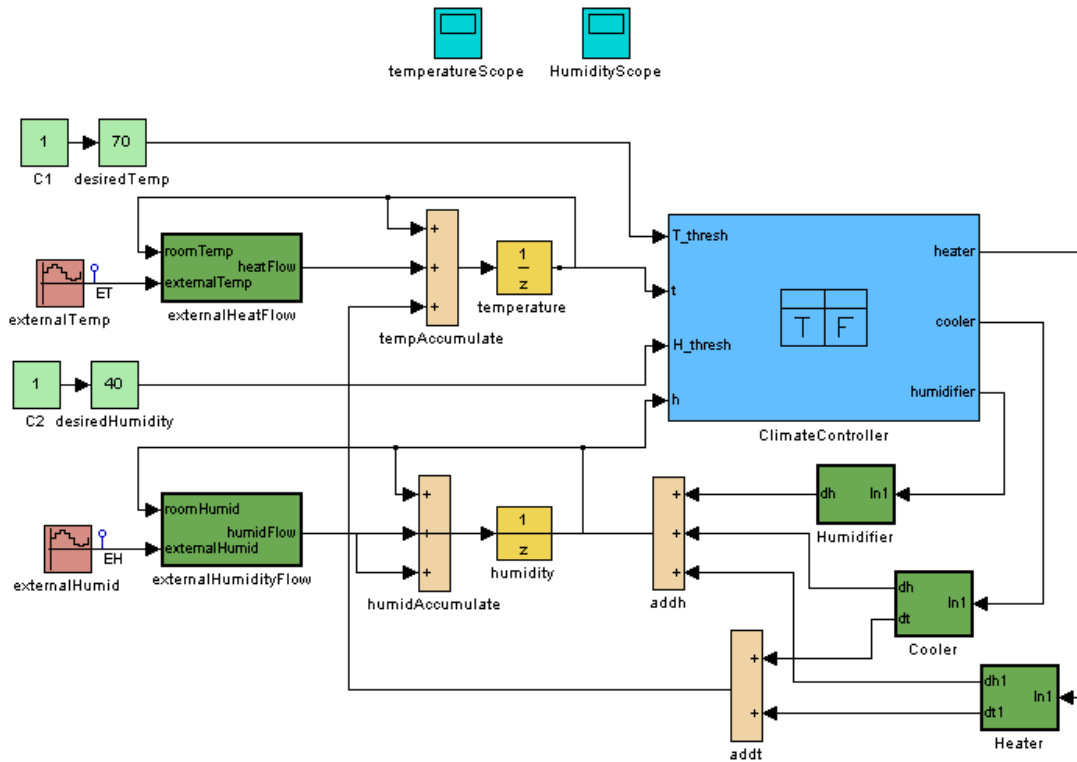
For purely logical behavior, truth tables are easier to program and maintain than graphical functions. Truth tables also provide diagnostics that indicate whether you have too few (underspecified) or too many (overspecified) decisions for the conditions you specify. For an introduction to truth tables, see “Truth Table Functions for Decision-Making Logic”.

The following model, `sf_climate_control`, shows a home environment controller that attempts to maintain a selected temperature and

# Truth Table

humidity. The model has a Truth Table block, `ClimateController`, that responds to changes in room temperature (input `t`) and humidity (input `h`).

Home climate (temperature and humidity) controller using Truth Table



## Truth Table Editor

If you double-click the Truth Table block in `sf_climate_control`, the Truth Table Editor opens to display its conditions, actions, and decisions. Here is the display for the Truth Table block named `ClimateController`.

**Condition Table**

	Description	Condition	D1	D2	D3	D4
1	Hot	$t > T\_thresh$	T	T	-	-
2	Dry	$h < H\_thresh$	T	-	T	-
		Actions: Specify a row from the Action Table	CoolOn, HumidOn	CoolOn	HeatOn, HumidOn	HeatOn

**Action Table**

#	Description	Action
1	Turn On Cooling (This implicitly reduces humidity)	CoolOn: cooler = 1; heater = 0; humidifier = 0;
2	Turn On Heater (This implicitly reduces humidity)	HeatOn: heater = 1; cooler = 0; humidifier = 0;
3	Turn On Humidifier	HumidOn: humidifier = 1;

The inputs  $t$  and  $h$  define the conditions, and the outputs  $heater$ ,  $cooler$ , and  $humidifier$  define the actions for this Truth Table block. For more details, refer to the demo description for this model.

Using the Truth Table Editor, you can:

- Enter and edit conditions, actions, and decisions
- Add or modify Stateflow data and ports using the Ports and Data Manager
- Run diagnostics to detect parser errors
- View generated content after simulation



# Truth Table


For more information about the Truth Table Editor, see “Truth Table Editor Operations”.

## Ports and Data Manager

To add or edit data in a Truth Table block, open the Ports and Data Manager by selecting **Add > Edit Data/Ports** in the Truth Table Editor.

Using the Ports and Data Manager, you can add these elements to a Truth Table block.

Element	Tool	Description
Data		You can add these types of data: <ul style="list-style-type: none"><li>• Local</li><li>• Constant</li><li>• Parameter</li><li>• Data store memory</li></ul>
Input trigger		An <i>input trigger</i> causes a Truth Table block to execute when a Simulink control signal changes or through a Simulink block that outputs function-call events. You can use one of these input triggers: <ul style="list-style-type: none"><li>• Rising edge</li><li>• Falling edge</li><li>• Either rising or falling edge</li><li>• Function call</li></ul>

Element	Tool	Description
		For more information, see “Defining Events”.
Function-call output		A <i>function-call output</i> triggers a function call to a subsystem. For more information, see “Function-Call Subsystems” in the Simulink documentation.

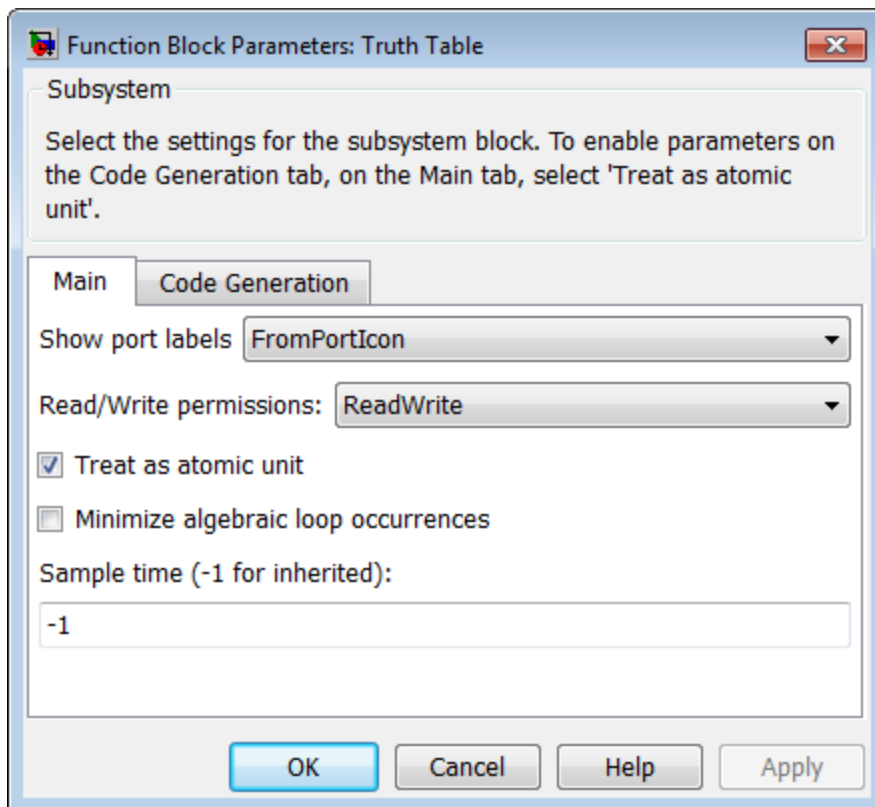
## Data Type Support

The Truth Table block accepts signals of any data type that Simulink supports, including fixed-point and enumerated data types. The block also accepts frame-based signals. Truth Table blocks work with frame-based signals in the same way as MATLAB Function blocks (see “Working with Frame-Based Signals” in the Simulink documentation).

For a discussion of data types that Simulink supports, refer to the Simulink documentation.

# Truth Table

## Parameters and Dialog Box



For a description of the block parameters, see the Subsystem block reference page in the Simulink documentation.

## Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	N/A

## Truth Table

---

Dimensionalized	Yes
Zero-Crossing Detection	No

# Truth Table

---



## F

### functions

- sfclipboard 2-2
- sfclose 2-3
- sfdebugger 2-4
- sfexplr 2-5
- sfgco 2-6
- sfhelp 2-8
- sflib 2-9
- sfnew 2-10
- sfopen 2-12
- sfprint 2-13
- sfroot 2-17
- sfsave 2-18
- stateflow 2-21

## P

Ports and Data Manager 3-10

## S

- sfclipboard function
  - reference 2-2
- sfclose function
  - reference 2-3
- sfdebugger function
  - reference 2-4

- sfexplr function
  - reference 2-5
- sfgco function
  - reference 2-6
- sfhelp function
  - reference 2-8
- sflib function
  - reference 2-9
- sfnew function
  - reference 2-10
- sfopen function
  - reference 2-12
- sfprint function
  - reference 2-13
- sfroot function
  - reference 2-17
- sfsave function
  - reference 2-18
- stateflow function
  - reference 2-21

## T

- Truth Table block
  - Ports and Data Manager 3-10
  - Truth Table Editor 3-8
- Truth Table Editor 3-8