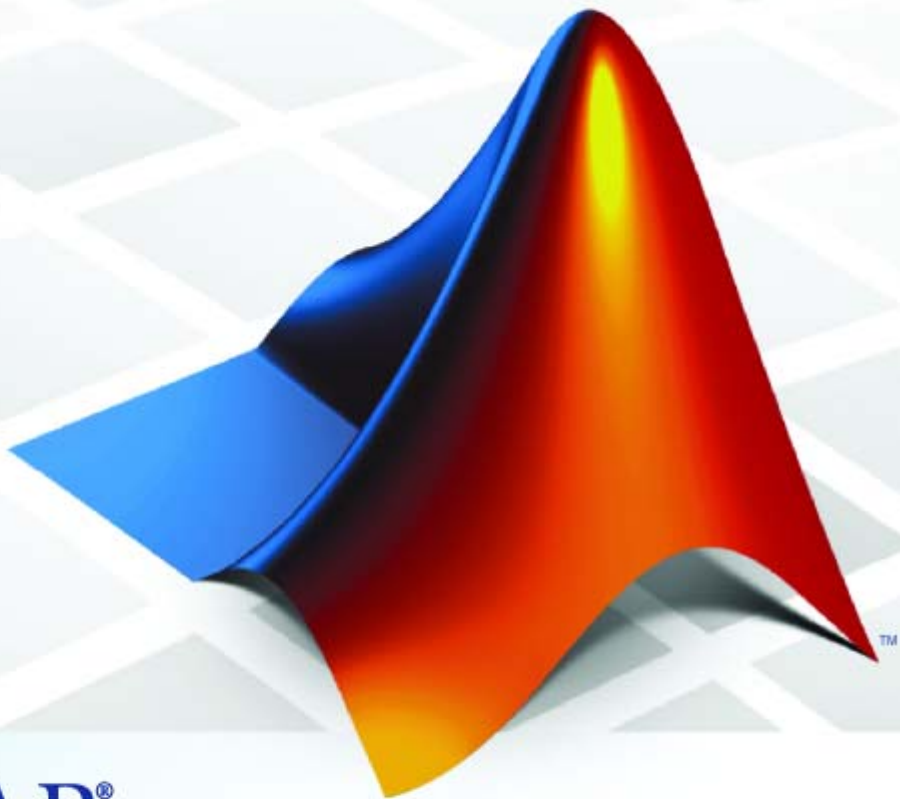


# Stateflow<sup>®</sup> and Stateflow<sup>®</sup> Coder<sup>™</sup> 7 Reference



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

## How to Contact The 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> and Stateflow<sup>®</sup> Coder<sup>™</sup> Reference*

© COPYRIGHT 2006-2009 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

The 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)



## Function Reference

<b>1</b>	<hr/>	
	Object Retrieval .....	1-2
	Chart Creation .....	1-2
	Chart Input/Output .....	1-2
	GUI .....	1-3
	Help .....	1-3

## Functions — Alphabetical List

<b>2</b>	<hr/>	
----------	-------	--

## Block Reference

<b>3</b>	<hr/>	
----------	-------	--

## Index

<hr/>	
-------	--



# Function Reference

---

Object Retrieval (p. 1-2)

Get objects in Stateflow® hierarchy

Chart Creation (p. 1-2)

Create Stateflow charts and truth tables

Chart Input/Output (p. 1-2)

Read and write Stateflow charts

GUI (p. 1-3)

Launch tools for defining and debugging Stateflow objects

Help (p. 1-3)

Get help on using Stateflow software

## Object Retrieval

<code>sfclipboard</code>	Get Stateflow clipboard object
<code>sfgco</code>	Get most recently selected objects in Stateflow chart
<code>sfroot</code>	Get Stateflow root object

## Chart Creation

<code>sfnew</code>	Create Simulink® model containing empty Stateflow block
<code>stateflow</code>	Create Simulink model containing empty Stateflow chart, and open Stateflow library window

## Chart Input/Output

<code>sfclose</code>	Close Stateflow chart
<code>sfopen</code>	Open Stateflow machine
<code>sfprint</code>	Print graphical view of Stateflow charts
<code>sfsave</code>	Save Stateflow machine in current directory



## GUI

<code>sfdebugger</code>	Open Stateflow Debugger
<code>sfexplr</code>	Start Model Explorer
<code>sflib</code>	Open Stateflow library window

## Help

<code>sfhelp</code>	Open Stateflow online help
---------------------	----------------------------



# Functions — Alphabetical List

---

# sfclipboard

---

**Purpose** Get Stateflow clipboard object

**Syntax** *object* = sfclipboard

**Description** *object* = sfclipboard returns a handle to the Stateflow clipboard object. Use the clipboard object to copy objects from one container object to another, as described in “Copying Objects” in the Stateflow API Reference.

**See Also** sfgco, sfnew, sfroot, stateflow

**Purpose** Close Stateflow chart

**Syntax**

```
sfclose  
sfclose( 'Chart_Name' )  
sfclose( Chart_Handle )  
sfclose( 'All' )
```

**Arguments**

<i>'Chart_Name'</i>	Name of a Stateflow chart
<i>Chart_Handle</i>	Handle to a Stateflow chart
<i>'All'</i>	Literal string to close all open or minimized Stateflow charts

**Description**

*sfclose* closes the current Stateflow chart.

*sfclose*( 'Chart\_Name' ) closes the Stateflow chart named **Chart\_Name**.

*sfclose*( *Chart\_Handle* ) closes the Stateflow chart whose handle is *Chart\_Handle*.

*sfclose*( 'All' ) closes all open or minimized Stateflow charts.

**See Also** `sfopen`, `sfnew`, `stateflow`

# sfdebugger

---

**Purpose** Open Stateflow Debugger

**Syntax**

```
sfdebugger  
sfdebugger( 'Machine_Name' )  
sfdebugger( Machine_Handle )  
sfdebugger( Machine_Id )
```

## Arguments

<i>'Model_Name'</i>	String name of a Stateflow machine
<i>Machine_Handle</i>	Handle to a Stateflow machine
<i>Machine_Id</i>	ID of a Stateflow machine

## Description

*sfdebugger* opens the Stateflow Debugger for the currently selected Stateflow machine.

*sfdebugger*( 'Machine\_Name' ) opens the Stateflow Debugger for the Stateflow machine called **Machine\_Name**.

*sfdebugger*( *Machine\_Handle* ) opens the Stateflow Debugger for the Stateflow machine whose handle is *Model\_Handle*.

*sfdebugger*( *Machine\_Id* ) opens the Stateflow Debugger for the Stateflow machine whose Id is *Machine\_Id*.

## See Also

sfexplr, sfhelp, sflib

**Purpose** Start Model Explorer

**Syntax** sfexplr

**Description** sfexplr opens the Model Explorer. For more information, see “The Model Explorer” in the Simulink software documentation.

**See Also** sfdebugger, sfhelp, sflib

**Purpose** Get most recently selected objects in Stateflow chart

**Syntax** `object = sfgco`

**Description** `object = sfgco` returns a handle or vector of handles to the most recently selected objects in a Stateflow chart, as follows.

<b>If ...</b>	<b>sfgco returns ...</b>
No Stateflow charts are open, or no open charts were edited or otherwise manipulated	Empty matrix
There is no selection list	Handle to the Stateflow chart most recently clicked
You select one object in a Stateflow chart	Handle to the selected object
You select multiple objects in a Stateflow chart	Vector of handles to the selected objects
You select multiple objects in multiple Stateflow charts	Vector of handles to the most recently selected objects in the most recently selected chart

**See Also** `sfnew`, `stateflow`



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

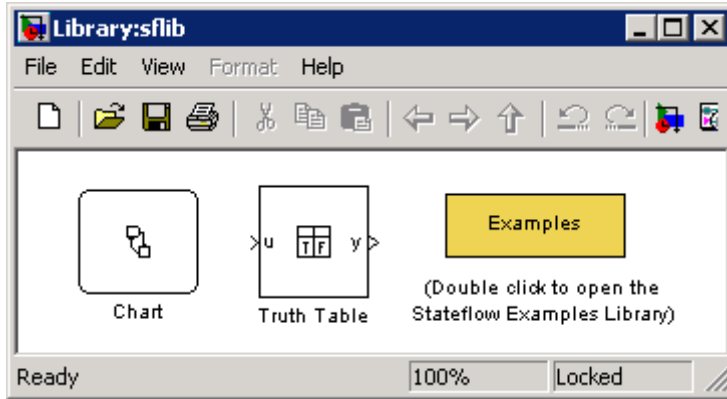
# sflib

---

**Purpose** Open Stateflow library window

**Syntax** sflib

**Description** sflib opens the Stateflow library window, as shown.



From this window, you can drag Stateflow charts and Truth Table blocks into Simulink models and access the Stateflow Examples Library.

**See Also** sfdebugger, sfexplr, sfhelp, sfnew

**Purpose** Create Simulink model containing empty Stateflow block

**Syntax** `Model_Handle = sfnew('-Chart_Type', 'Machine_Name')`

## Arguments

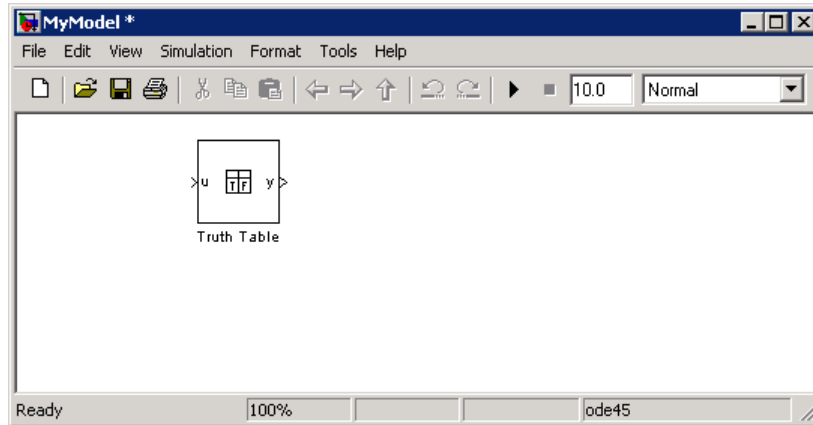
<i>Model_Handle</i>	Handle to the new Simulink model that will contain the Stateflow block
<i>Chart_Type</i>	Type of Stateflow block to add to the Simulink model. Enter <ul style="list-style-type: none"> <li>• '-Classic' for a chart that implements full Stateflow chart semantics (default)</li> <li>• '-Mealy' for a chart that implements Mealy state machine semantics</li> <li>• '-Moore' for a chart that implements Moore state machine semantics</li> <li>• '-TT' for a truth table</li> </ul> Optional.
<i>'Machine_Name'</i>	Name of the Stateflow machine (also becomes the model name). Optional.

**Description** `Model_Handle = sfnew('-Chart_Type', 'Machine_Name')` returns the handle to a new model named **Machine\_Name** that contains an empty Stateflow block of type *Chart\_Type*, and opens the new model on your desktop. If *Chart\_Type* is not specified, the default block is **Classic**. If *Machine\_Name* is not specified, the default name is **untitled**.

**Examples** Create a Simulink model called **MyModel** that contains an empty Stateflow truth table.

```
m = sfnew('-TT', 'MyModel')
```

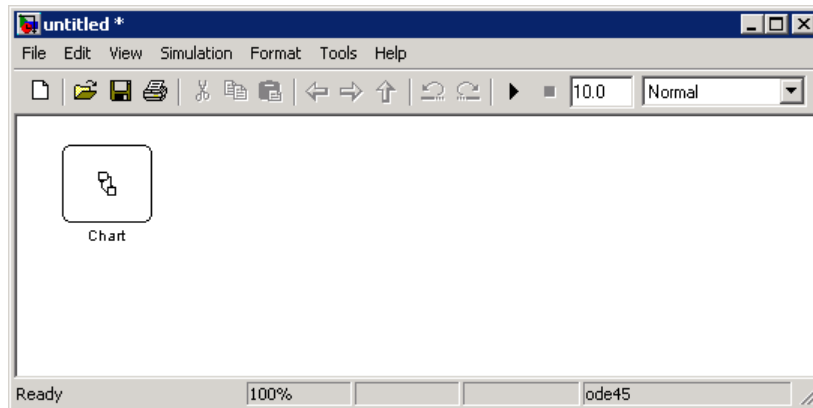
The new model looks like this:



Create an untitled Simulink model that contains an empty Stateflow chart.

```
m = sfnew
```

The new model looks like this:



**See Also**      sfhelp, sfprint, sfroot, sfsave, stateflow

# sfopen

---

<b>Purpose</b>	Open Stateflow machine
<b>Syntax</b>	sfopen
<b>Description</b>	sfopen prompts you for an .mdl file and opens the model that you select from your file system.
<b>See Also</b>	sfclose, sfdebugger, sfexplr, sflib, sfnew, stateflow

**Purpose**

Print graphical view of Stateflow charts

**Syntax**

```
sfprint
sfprint( objects, format, outputOption, printEntireChart )
```

**Arguments**

*objects*

Any of these object identifiers:

- String name of a Stateflow chart, or Simulink model, system, or block
- Handle to a Stateflow chart, or Simulink model, system, or block
- Cell array of names of and/or handles to a Stateflow chart, or Simulink model, system, or block
- Vector of handles to a Stateflow chart, or Simulink model, system, or block
- Simulink model construction commands `gcb`, `gcbh`, or `gcs`

*format*

Optional literal string that specifies the print destination:

- 'default' prints to a default printer
- 'ps' generates a PostScript file
- 'psc' generates a color PostScript file
- 'eps' generates an Encapsulated PostScript file
- 'epsc' generates a color Encapsulated PostScript file
- 'tif' generates a TIFF file
- 'jpg' generates a JPEG file

- 'png' generates a PNG file
  - 'meta' saves the Stateflow chart image to the clipboard as a metafile (for Windows® operating systems only)
  - 'bitmap' saves the Stateflow chart image to the clipboard as a bitmap (for Windows operating systems only)
- outputOption* Optional string that specifies an output file or printer:
- String that specifies the name of a file to which to write (file will be overwritten if more than one chart is printed)
  - 'promptForFile' prompts for file name interactively
  - 'printer' sends output to default printer (use only with 'default', 'ps', or 'eps' formats)
  - 'file' sends output to a default file, specified as *<path to object>.<device extension>*
  - 'clipboard' copies output to the clipboard
- printEntireChart* Optional Boolean argument:
- 1 (default) prints complete charts
  - 0 prints current view of charts

## Description

sfprint prints the current Stateflow chart to a default printer.

sfprint( *objects*, *format*, *outputOption*, *printEntireChart* )  
prints all Stateflow charts identified in *objects* in the specified *format* to the file or printer specified in *outputOption*. Prints a complete or



current view of charts as specified in *printEntireChart*. If the *format* argument is absent, the format defaults to 'ps' and output is sent to the default printer. If the *outputOption* argument is absent, the name of the Stateflow chart in the current directory is used as the output file name.

## Examples

Print the complete chart whose handle is *id* to a TIFF file called **myFilename**.

```
sfprint(id, 'tif', 'myFilename')
```

Print all Stateflow charts in the current system as a PostScript file to the default printer.

```
sfprint(gcs)
```

Print the current Stateflow block to a JPEG file whose name is specified by the user interactively.

```
sfprint(gcb, 'jpg', 'promptForFile')
```

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

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

Assume that you loaded a Simulink model named **myModel** that has two charts named **Chart1** and **Chart2**. Further, both **Chart1** and **Chart2** are represented by the Stateflow chart objects **ch1** and **ch2**, respectively.

This command...	Prints the graphical view of...
<code>sfprint('myModel')</code>	Both <b>Chart1</b> and <b>Chart2</b> to the default printer
<code>sfprint('myModel','ps')</code>	Both <b>Chart1</b> and <b>Chart2</b> to a PostScript file

# sfprint

---

This command...	Prints the graphical view of...
<code>sfprint(ch1.Id,'psc')</code>	<b>Chart1</b> to a color PostScript file
<code>sfprint([ch1.Id, ch2.Id])</code>	Both <b>Chart1</b> and <b>Chart2</b> to the default printer

## See Also

`sfhelp`, `sfnew`, `sfsave`, `stateflow`

**Purpose** Get Stateflow root object

**Syntax** `object = sfroot`

**Description** `object = sfroot` returns the handle to the top-level object in the Stateflow hierarchy of objects. Use the root object to access all other objects in Stateflow charts, as described in “Accessing the Model Object” in the Stateflow API Reference.

**See Also** `sfnew`, `sfgco`, `sfclipboard`, `stateflow`

**Purpose** Save Stateflow machine in current directory

**Syntax**

```
sfsave  
sfsave( Model_Handle )  
sfsave( Model_Handle, 'New_Model_Name' )  
sfsave( Machine_Handle )  
sfsave( 'Model_Name' )  
sfsave( 'Defaults' )
```

## Arguments

<i>Model_Handle</i>	Handle to a Simulink model that contains a Stateflow block
' <i>New_Model_Name</i> '	Name to assign to the model being saved
<i>Machine_Handle</i>	Handle to a Stateflow machine
' <i>Model_Name</i> '	Name of a Simulink model that contains a Stateflow block
' <i>Defaults</i> '	Literal string used to save current settings as defaults

## Description

sfsave saves the current Stateflow machine in the current directory.

sfsave( *Model\_Handle* ) saves the Simulink model specified by *Model\_Handle* in the current directory.

sfsave( *Model\_Handle*, '*New\_Model\_Name*' ) saves the Simulink model specified by *Model\_Handle* as **New\_Model\_Name** in the current directory.

sfsave( *Machine\_Handle* ) saves the Simulink model that contains the Stateflow machine specified by *Machine\_Handle* in the current directory.

sfsave( '*Model\_Name*' ) saves the Simulink model called **Model\_Name** in the current directory.

`sfsave( 'Defaults' )` saves the settings of the current Stateflow machine as defaults.

**Examples**

Save the model whose handle is `m` as **MyModel** in the current directory.

```
sfsave(m, 'MyModel')
```

Save the model that contains a Stateflow machine whose handle is `sf` in the current directory.

```
sfsave(sf)
```

**See Also**

`sfclose`, `sfnew`, `sfoopen`, `sfprint`

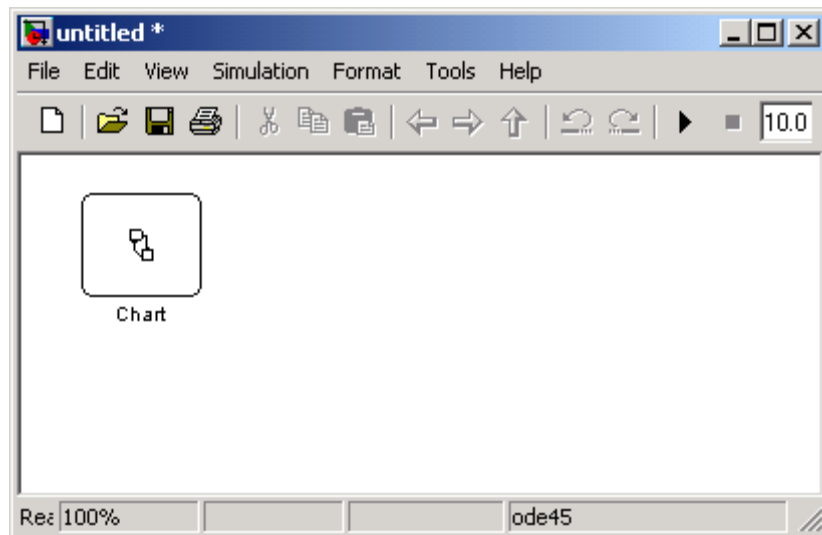
# stateflow

---

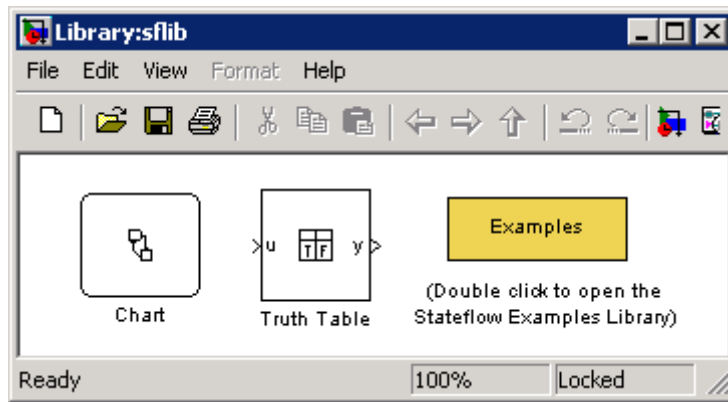
**Purpose** Create Simulink model containing empty Stateflow chart, and open Stateflow library window

**Syntax** stateflow

**Description** stateflow creates a new Simulink model that is preconfigured with an empty Stateflow chart, as shown.



The function also opens the Stateflow library window.



From this window, you can drag other Stateflow charts and Truth Table blocks into Simulink models and access the Stateflow Examples Library.

**See Also**

`sflib`, `sfnew`, `sfroot`





# Block Reference

---

# Stateflow Chart

---

**Purpose** A version of a finite state machine for controlling a physical plant

**Library** Stateflow

**Description**



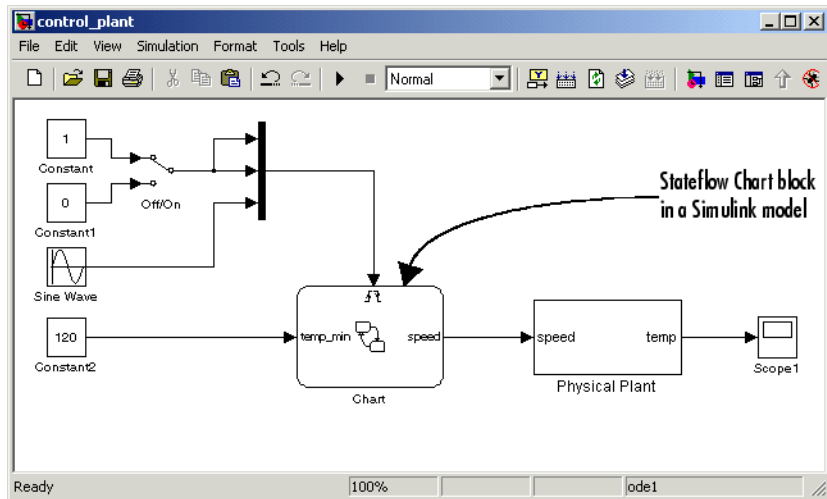
Chart1

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 prescribed state in response to an event, provided that 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 building blocks of the system. You can also represent stateless flow graphs. To add your control logic to a Simulink model, use a Stateflow block.

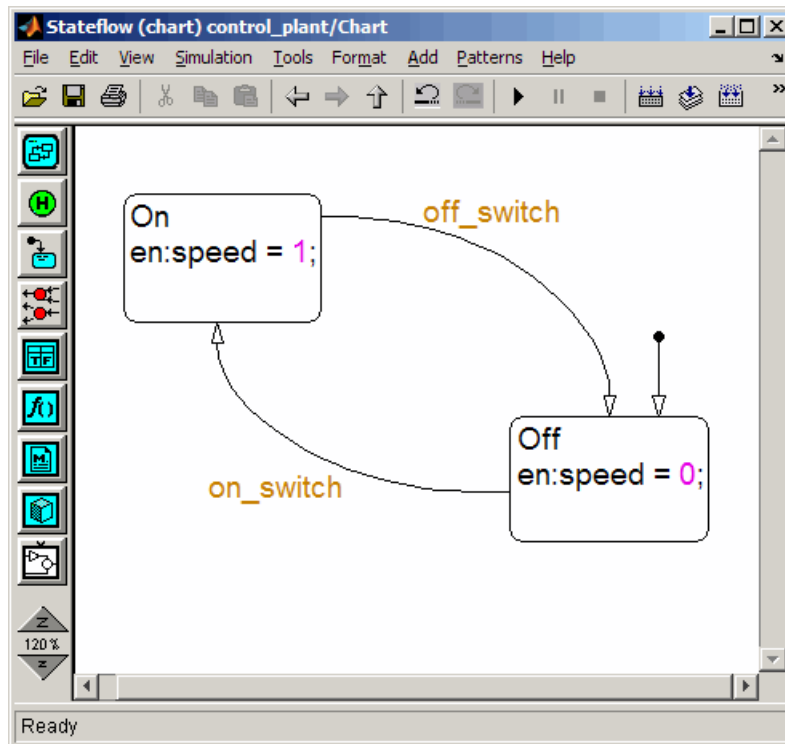
You can use Stateflow charts to control a physical plant in response to events such as a temperature or pressure sensor, or clock or user-driven events. For example, you can use a state machine to represent the automatic transmission of a car. The transmission has these operating states: park, reverse, 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 diagram shows a simple Simulink model that has a Stateflow block named Chart (default) that responds to input from a manual switch.



If you double-click the Stateflow block in the Simulink model, the Stateflow chart that programs the Stateflow block appears in the Stateflow Editor.

# Stateflow Chart



During simulation of the Simulink model, you can interactively debug Stateflow charts in animation mode. Stateflow charts generate efficient C code for simulation targets, and also for embedded targets.

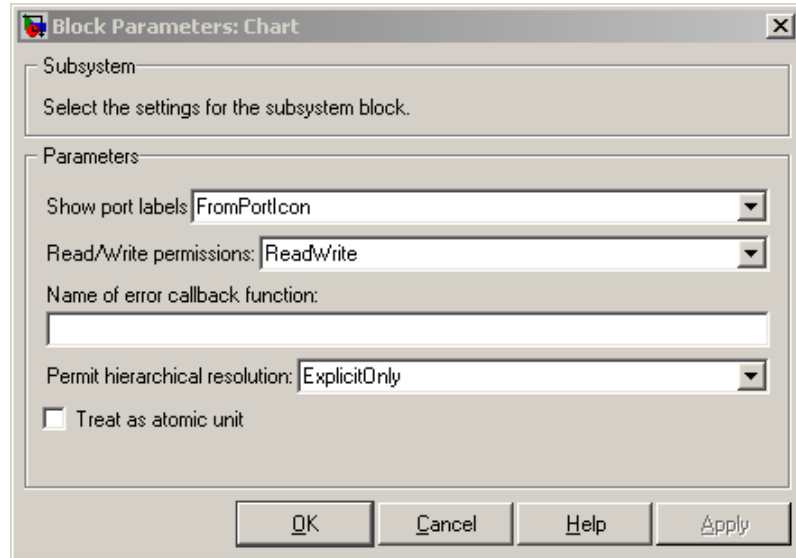
For an introduction to using Stateflow charts in Simulink models, see the Stateflow Getting Started Guide.

## Data Type Support

The Stateflow 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.

For a discussion on the variable types supported by Embedded MATLAB™ functions in Simulink models, refer to the Simulink software documentation.

You can declare local data of any type or size.



## Parameters and Dialog Box

---

**Note** It is highly recommended that the default settings for the block parameters of an Embedded MATLAB Function block not be changed.

---

## Characteristics

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

# Stateflow Chart

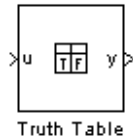
---

Dimensionalized	Yes
Zero Crossing	No

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

**Library** Stateflow

**Description**



The Truth Table block is an Embedded MATLAB truth table function that you can add to a Simulink model directly. The Truth Table block requires a Stateflow software license.

When you add a Truth Table block directly to a Simulink 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 supports the Embedded MATLAB language subset for programming conditions and actions, and generates content as Embedded MATLAB code. Embedded MATLAB functions work with a subset of the MATLAB language that is optimized for generating embeddable C code.

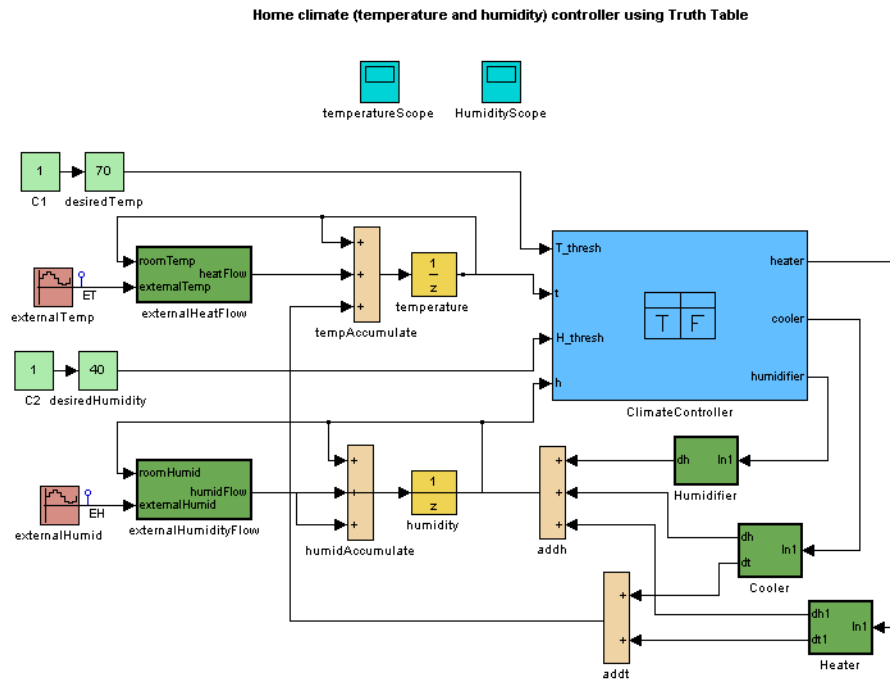
As a result, you can take advantage of Embedded MATLAB tools to debug your Truth Table block during simulation. For more information, see “Debugging an Embedded MATLAB Function”.

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

# Truth Table

too many (overspecified) decisions for the conditions you specify. For an introduction to truth tables, see “Truth Table Functions”.

This figure shows a Simulink model (`sf_climate_control.mdl`) of a home environment controller that attempts to maintain a selected temperature and humidity. The model has a Truth Table block (`ClimateController`) that responds to changes in room temperature (input `t`) and humidity (input `h`).



## Truth Table Editor

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



The screenshot shows the Truth Table Editor for a climate controller block. The interface includes a menu bar (File, Edit, Settings, Add, Help) and a toolbar with various icons. The main content area is divided into two sections:

**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 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

If you want to add or edit data in a Truth Table block, open the Ports and Data Manager by clicking the **Edit Data/Ports** button in the Truth Table Editor toolbar:



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

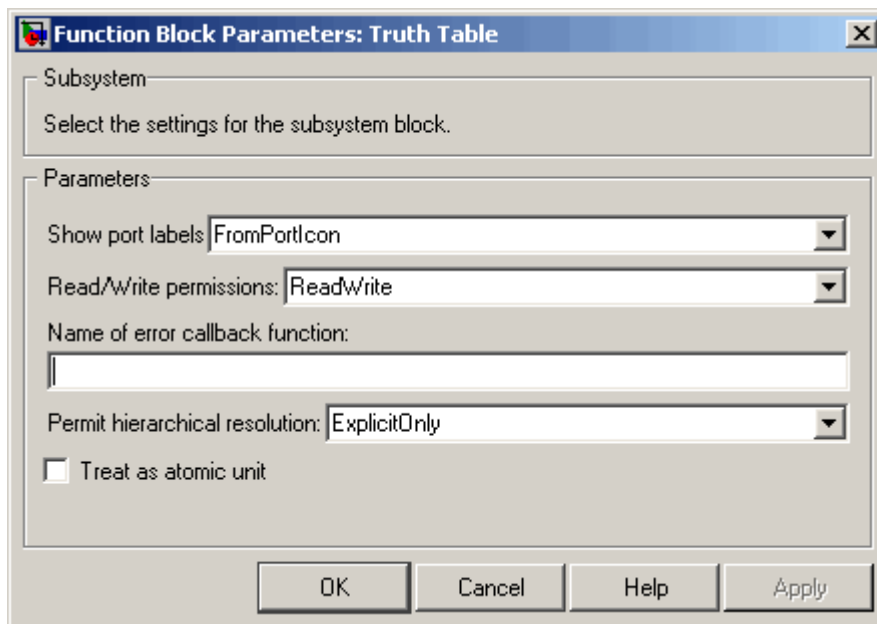
The Truth Table block accepts signals of any data type supported by Simulink models, including fixed-point data types, enumerated data types, and frame-based signals. Truth Table blocks work with frame-based signals in the same way as Embedded MATLAB Function blocks (see “Working with Frame-Based Signals”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

For a discussion of data types supported by Simulink models, refer to the Simulink documentation.

# Truth Table

## Parameters and Dialog Box

Right-click over a Truth Table block, and from the submenu, select **Subsystem Parameters**.



## Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	N/A
Dimensionalized	Yes
Zero Crossing	No

## F

### functions

- sfclipboard 2-2
- sfclose 2-3
- sfdebugger 2-4
- sfexplr 2-5
- sfgco 2-6
- sfhelp 2-7
- sflib 2-8
- sfnew 2-9
- sfopen 2-12
- sfprint 2-13
- sfroot 2-17
- sfsave 2-18
- stateflow 2-20

## 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 2-7
- sflib function
  - reference 2-8
- sfnew function
  - reference 2-9
- sfopen function
  - reference 2-12
- sfprint function
  - reference 2-13
- sfroot function
  - reference 2-17
- sfsave function
  - reference 2-18
- stateflow function
  - reference 2-20

## T

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