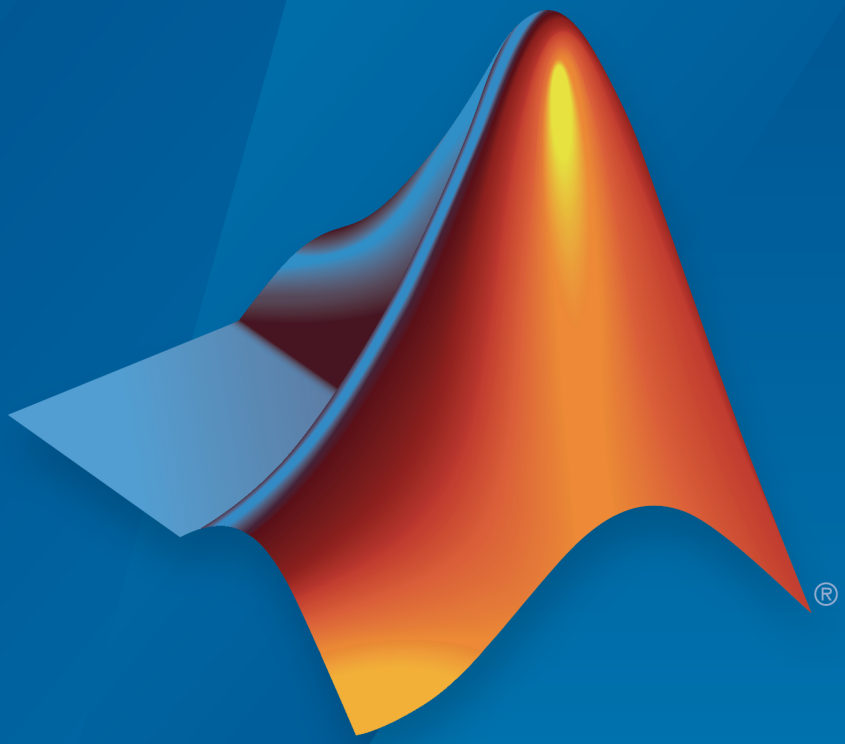
Stateflow®

Reference



MATLAB® & SIMULINK®

R2017a

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

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

Stateflow® Reference

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)
September 2012	Online only	Revised for Version 8.0 (Release 2012b)
March 2013	Online only	Revised for Version 8.1 (Release 2013a)
September 2013	Online only	Revised for Version 8.2 (Release 2013b)
March 2014	Online only	Revised for Version 8.3 (Release 2014a)
October 2014	Online only	Revised for Version 8.4 (Release 2014b)
March 2015	Online only	Revised for Version 8.5 (Release 2015a)
September 2015	Online only	Revised for Version 8.6 (Release 2015b)
October 2015	Online only	Rereleased for Version 8.5.1 (Release 2015aSP1)
March 2016	Online only	Revised for Version 8.7 (Release 2016a)
September 2016	Online only	Revised for Version 8.8 (Release 2016b)
March 2017	Online only	Revised for Version 8.9 (Release 2017a)

Contents

Functions — Alphabetical List

sfclipboard

Stateflow clipboard object

Syntax

```
object = sfclipboard
```

Description

`object = sfclipboard` returns a handle to the Stateflow® clipboard object, which you use to copy objects from one chart or state to another.

Examples

Copy the `init` function from the `Init` chart to the `Pool` chart in the `sf_pool` model:

```
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];
```

Tutorials

- "Copy Objects"

- "Create and Access Charts Using the Stateflow API"

See Also

`sfgco` | `sfnew` | `sfroot` | `stateflow`

Topics

"Copy Objects"
"Create and Access Charts Using the Stateflow API"
"Getting a Handle on Stateflow API Objects"
"Access the Chart Object"

Introduced before R2006a

sfclose

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 character vector.

See Also

`sfnew` | `sfopen` | `stateflow`

Introduced in R2006a

sfdebugger

Open Stateflow Debugger

Syntax

```
sfdebugger
sfdebugger('model_name')
```

Description

`sfdebugger` opens the Stateflow Debugger for the current model.

`sfdebugger('model_name')` opens the debugger for the Simulink® model called `'model_name'`. Use this input argument to specify which model to debug when you have multiple models open.

See Also

`sfexplr` | `sfhelp` | `sflib`

Topics

"Debug Run-Time Errors in a Chart"

Introduced in R2006a

sfexplr

Open Model Explorer

Syntax

```
sfexplr
```

Description

`sfexplr` opens the Model Explorer. A model does not need to be open.

See Also

`sfdebugger` | `sfhelp` | `sflib`

Topics

"Use the Model Explorer with Stateflow Objects"

Introduced in R2006a

sfgco

Recently selected objects in chart

Syntax

```
object = sfgco
```

Description

`object = sfgco` returns a handle or vector of handles to the most recently selected objects in a chart.

Output Arguments

`object`

Handle or vector of handles to the most recently selected objects in a chart

Empty matrix	No charts are open, or you have no edited charts.
Handle to the chart most recently clicked	You clicked in a chart, but did not select any objects.
Handle to the selected object	You selected one object in a chart.
Vector of handles to the selected objects	You selected multiple objects in a chart.
Vector of handles to the most recently selected objects in the most recently selected chart	You selected multiple objects in multiple charts.

Examples

Zoom in on a state after clicking it:

```
myState = sfgco;
```

```
% Zoom in on the selected state
myState.fitToView;
```

Tutorials

- "Create and Access Charts Using the Stateflow API"

See Also

`sfnew` | `sfroot` | `stateflow`

Topics

"Create and Access Charts Using the Stateflow API"
"Getting a Handle on Stateflow API Objects"
"Zoom a Chart Object Using the API"

Introduced before R2006a

sfhelp

Open Stateflow online help

Syntax

```
sfhelp
```

Description

`sfhelp` opens the Stateflow online help in the MATLAB® Help browser.

See Also

`sfdebugger` | `sfexplr` | `sfnew` | `stateflow`

Introduced before R2006a

sflib

Open Stateflow library window

Syntax

```
sflib
```

Description

`sflib` opens the Stateflow block library. From this library, you can drag Stateflow blocks into Simulink models and access the Stateflow Examples Library.

See Also

`sfdebugger` | `sfexplr` | `sfhelp` | `sfnew`

Introduced in R2006a

sfnew

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. Stateflow sets the default action language for new charts to MATLAB. To change the default action language to C, use the command `sfpref('ActionLanguage','C')`. For more information, see "Modify the Action Language for a Chart".

`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 with the default action language.

`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:

`-MATLAB`	Use a chart that supports MATLAB expressions in Stateflow actions

`-C`	Use a chart that supports C expressions in Stateflow actions
`-Mealy`	Use a chart that supports only Mealy state machine semantics
`-Moore`	Use a chart that supports only Moore state machine semantics
`-TT`	Use a truth table
`-STT`	Use a state transition table

`model_name`

Name of the model.

Examples

Create a untitled model with an empty chart that uses MATLAB as the action language:

```
sfnew()
```

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` | `sfroot` | `sfsave` | `stateflow`

Topics

“Model Event-Driven System”
“Create Mealy and Moore Charts”
“Build Model with Stateflow Truth Table”
“Syntax for States and Transitions”

Introduced before R2006a

sfopen

Open existing model

Syntax

```
sfopen
```

Description

`sfopen` prompts you for a model file and opens the model that you select from your file system.

See Also

`sfclose` | `sfdebugger` | `sfexplr` | `sflib` | `sfnew` | `stateflow`

Introduced in R2006a

sfprint

Print graphical view of charts

Syntax

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

Description

`sfprint` prints the current chart to the default printer.

`sfprint(objects)` prints all charts specified by `objects` to the default printer.

`sfprint(objects,format)` prints all charts specified by `objects` in the specified `format` to output files. Each output file matches the name of the chart and the file extension matches the `format`.

`sfprint(objects,format,outputOption)` prints all charts specified by `objects` in the specified `format` to the file or printer specified in `outputOption`.

`sfprint(objects,format,outputOption,wholeChart)` prints all charts specified by `objects` in the specified `format` to the file or printer specified in `outputOption`. As specified in `wholeChart`, prints either a complete or current view.

Examples

Print open chart

```
sfprint
```

Prints current chart to the default printer.

Print all charts specified in path

```
sfprint('sf_car/shift_logic');
```

Prints the chart with the path 'sf_car/shift_logic' to the default printer.

Print chart specified in path to a JPG file format.

```
sfprint('sf_car/shift_logic','jpg')
```

Prints a copy of the chart 'sf_car/shift_logic' in JPG format to the file 'sf_car_shift_logic.jpg'.

Print chart in TIFF format to the clipboard.

```
sfprint(gcs,'tif','clipboard')
```

Prints the chart in the current system to the clipboard in TIFF format.

Print the current view of a chart.

```
sfprint('sf_car/shift_logic','png','file',0)
```

Prints the current view of 'sf_car/shift_logic' in a PNG format to the file 'sf_car_shift_logic.png'.

Input Arguments

`objects` — Identifier of charts to print

`gcb` (default) | `gcs` | character vector

Identifier of charts to print. Use:

- `gcb` to specify the current block of the model.
- `gcs` to specify the current system of the model.
- a character vector to specify the path of a chart, model, subsystem, or block.

Example: `sfprint(gcs)`

Prints all the charts in the current system to the default printer.

Example: `sfprint('sf_pool/Pool')`

Prints the complete chart with the path `'sf_pool/Pool'` to the default printer.

`format` — Output format of printed charts

`'bitmap' | 'jpg' | 'meta' | 'pdf' | 'png' | 'svg' | 'tif'`

Output format of the printed charts specified as one of these values:

`'bitmap'`	Save the chart image to the clipboard as a bitmap (for Windows® operating systems only)
`'jpg'`	Generate a JPEG file
`'meta'`	Save the chart image to the clipboard as a metafile (for Windows operating systems only)
`'pdf'`	Generate a PDF file
`'png'`	Generate a PNG file
`'svg'`	Generate an SVG file
`'tif'`	Generate a TIFF file

Example: `sfprint('sf_car/shift_logic','jpg')`

Prints the complete chart with the path `'sf_car/shift_logic'` in a JPEG format to a file in the current folder named `'sf_car_shift_logic.jpg'`.

Data Types: `char`

`outputOption` — Name of the printer or output file

`'file'` (default) | character vector | `'clipboard'` | `'promptForFile'` | `'printer'`

Name of the output file or printer specified as one of these values:

`'file'`	Send output to a default file with the name *`chart_name.file_extension`*. The file name is the name of the chart, with an extension that matches the output format.
character vector	Specify the name of the output file with a character vector.

`'clipboard'`	Copy output to the clipboard
`'promptForFile'`	Prompts the user interactively for path and file name.
`'printer'`	Send output to the default printer (use only with `'ps'`, or `'eps'` formats)

Example: `sfprint('sf_car/shift_logic','png','myFile')`

Prints the complete chart whose path is `'sf_car/shift_logic'` in the PNG format to a file in the current folder with the name `'myFile'.png`.

Example: `sfprint('sf_car/shift_logic,'pdf','promptForFile')`

Prints all charts in the current block of the model in PDF format. A dialog box opens for each chart to prompt you for the path and name of the output file.

Data Types: `char`

`wholeChart` — View of charts to print

1 (default) | 0

View of charts to print specified as a integer of value 0 or 1. A value of 1 prints the complete views of all the charts, whereas a value of 0 prints the current views of all the charts.

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

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

See Also

See Also

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

Introduced before R2006a

sfroot

Root object

Syntax

```
object = sfroot
```

Description

object = `sfroot` 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.

Examples

Zoom in on a state in your chart:

```
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;
```

Tutorials

- "Create and Access Charts Using the Stateflow API"

See Also

`sfclipboard` | `sfgco`

Topics

"Create and Access Charts Using the Stateflow API"
"Getting a Handle on Stateflow API Objects"

"Access the Chart Object"

Introduced before R2006a

sfsave

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.

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

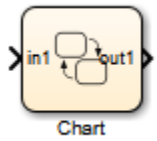
Examples

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

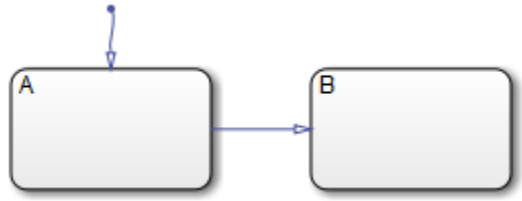
```
bdclose('all');

% 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:



Tutorials

- "Create and Access Charts Using the Stateflow API"

See Also

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

Topics

"Create and Access Charts Using the Stateflow API"
"Create a MATLAB Script of API Commands"

Introduced before R2006a

stateflow

Create empty chart

Syntax

```
stateflow
```

Description

`stateflow` creates an untitled model that contains an empty chart. The function also opens the Stateflow block library. From this library, you can drag Stateflow blocks into models or access the Stateflow Examples Library.

See Also

`sflib` | `sfnew`

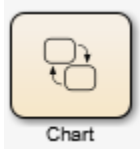
Introduced before R2006a

2

Block Reference

Chart

Implement control logic with finite state machine



Library

Stateflow

Description

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

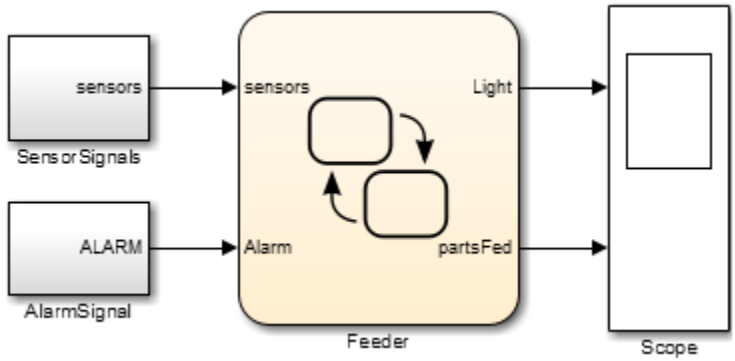
A Stateflow chart is a graphical representation of a finite state machine. *States* and *transitions* form the basic elements of the system. You can also represent stateless flow charts.

For example, you can use Stateflow charts to control a physical plant in response to events such as a temperature and pressure sensors, clocks, and user-driven events.

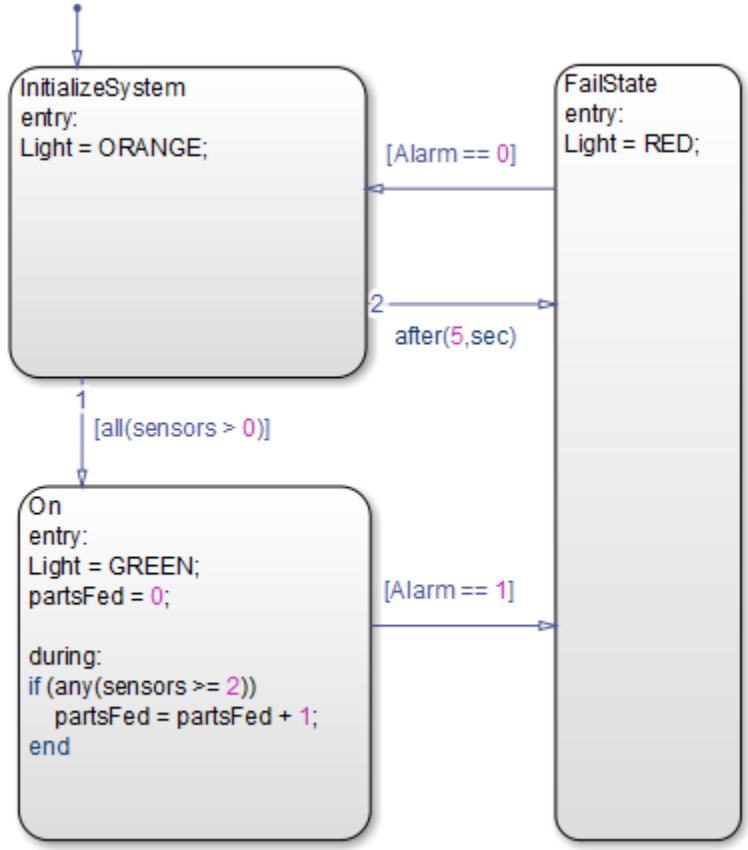
You can also use a state machine to represent the automatic transmission of a car. The transmission has these operating states: park, reverse, neutral, drive, and low. As the driver shifts from one position to another, the system makes a transition from one state to another, for example, from park to reverse.

A Stateflow Chart can use MATLAB or C as the action language to implement control logic.

This block diagram represents a machine on an assembly line that feeds raw material to other parts of the line. It contains a chart, `Feeder`, with MATLAB as the action language.



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



For a tutorial on this model, see “Model Event-Driven System”.

Data Type Support

The Chart block accepts input signals of any data type that Simulink supports, including fixed-point data and enumerated data types. For a description of data types that Simulink supports, refer to the Simulink documentation.

Floating-point inputs pass through the block unchanged. Boolean inputs to charts that use MATLAB as the action language pass directly as Boolean outputs. Boolean inputs to charts that use C as the action language are treated as `double` type.

You can declare local data of any type or size.

Parameters

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

Characteristics

Direct Feedthrough	Yes, for Classic and Mealy charts. No, for Moore charts.
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	N/A
Dimensionalized	Yes
Zero-Crossing Detection	Yes, if enabled for continuous-time systems. For more information, see “When to Disable Zero-Crossing Detection”.

Introduced in R2013b

Message Viewer

Display message or events between blocks during simulation

Library

Stateflow, SimEvents®, Simulink Test™

Description

The Message Viewer block displays messages or events between certain blocks during simulation. The blocks that you can display messages and events for are called *lifeline blocks* and include:

- Subsystems
- Stateflow charts
- Blocks that contain messages, for example, Stateflow charts.

Parameters

History

Specify maximum number of events to keep in viewer.

Settings

Default: 5000

See Also

Topics

"Work with Message Viewer"
"Work with Message Viewer" (SimEvents)
"Work with Message Viewer" (Simulink Test)

Introduced in R2015b

State Transition Table

Represent modal logic in tabular format

Library

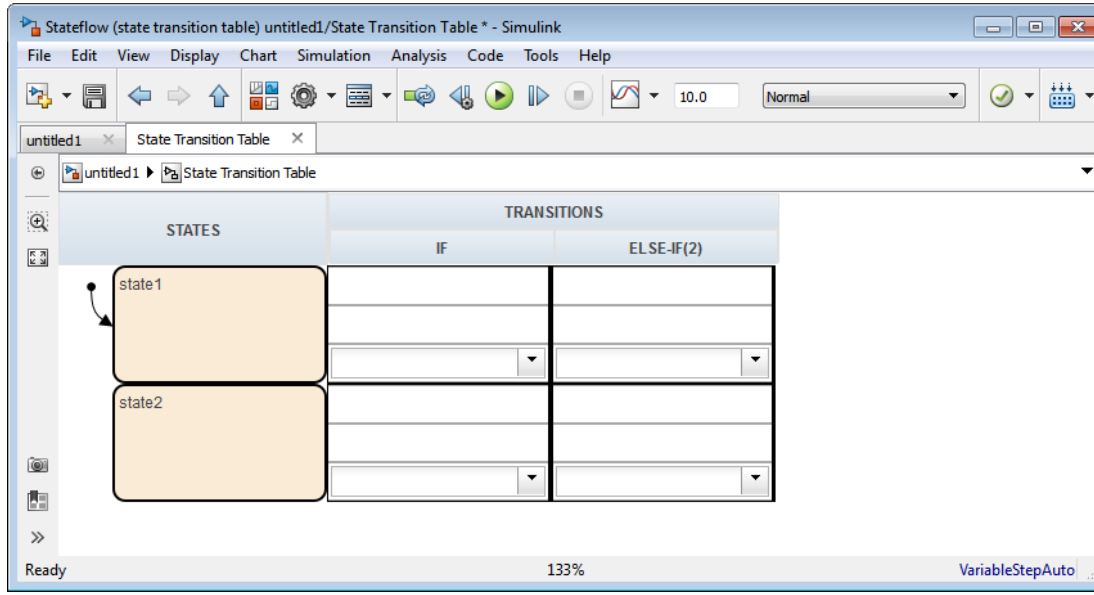
Stateflow

Description

Use this block when you want to represent modal logic in tabular format. The State Transition Table block uses only MATLAB as the action language.

State Transition Table Editor

If you double-click the State Transition Table block in `sflib`, the State Transition Table Editor shows the default layout of state-to-state transitions.



Using the State Transition Table Editor, you can:

- Add states and enter state actions
- Add hierarchy among your states
- Enter conditions and actions for state-to-state transitions
- Specify default transitions, inner transitions, and self-loop transitions
- Add input or output data and events
- Set breakpoints for debugging
- Run diagnostics to detect parser errors
- View auto-generated content as you edit the table

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

Adding Data and Events

You can add data and events from the State Transition Table Editor:

Element	Menu	Description
Inputs and outputs	**Chart > Add Inputs & Outputs > Data Input from Simulink** **Chart > Add Inputs & Outputs > Data Output to Simulink**	You can add inputs from the model and outputs to the model.
Data	**Chart > Add Other Elements**	You can add these types of data: • Local • Constant • Parameter • Data store memory
Input events	**Chart > Add Inputs & Outputs > Event Input from Simulink**	An *input event* causes a State Transition 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: • Rising edge • Falling edge • Either rising or falling edge • Function call
Output events	**Chart > Add Inputs & Outputs > Event Output to Simulink**	A *output event* triggers a function call to a subsystem. You can use one of these output triggers: • Function call • Either rising or falling edge For more information, see “Function-Call Subsystems” (Simulink) .

Data Type Support

The State Transition Table block accepts input signals of any data type that Simulink supports, including fixed-point and enumerated data types.

Parameters

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

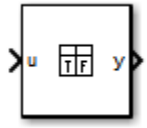
Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	N/A
Dimensionalized	Yes
Zero-Crossing Detection	Yes, if enabled for continuous-time systems For more information, see “When to Disable Zero-Crossing Detection”.

Introduced in R2012b

Truth Table

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



Library

Stateflow

Description

The Truth Table block is a truth table function that uses MATLAB as the 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:

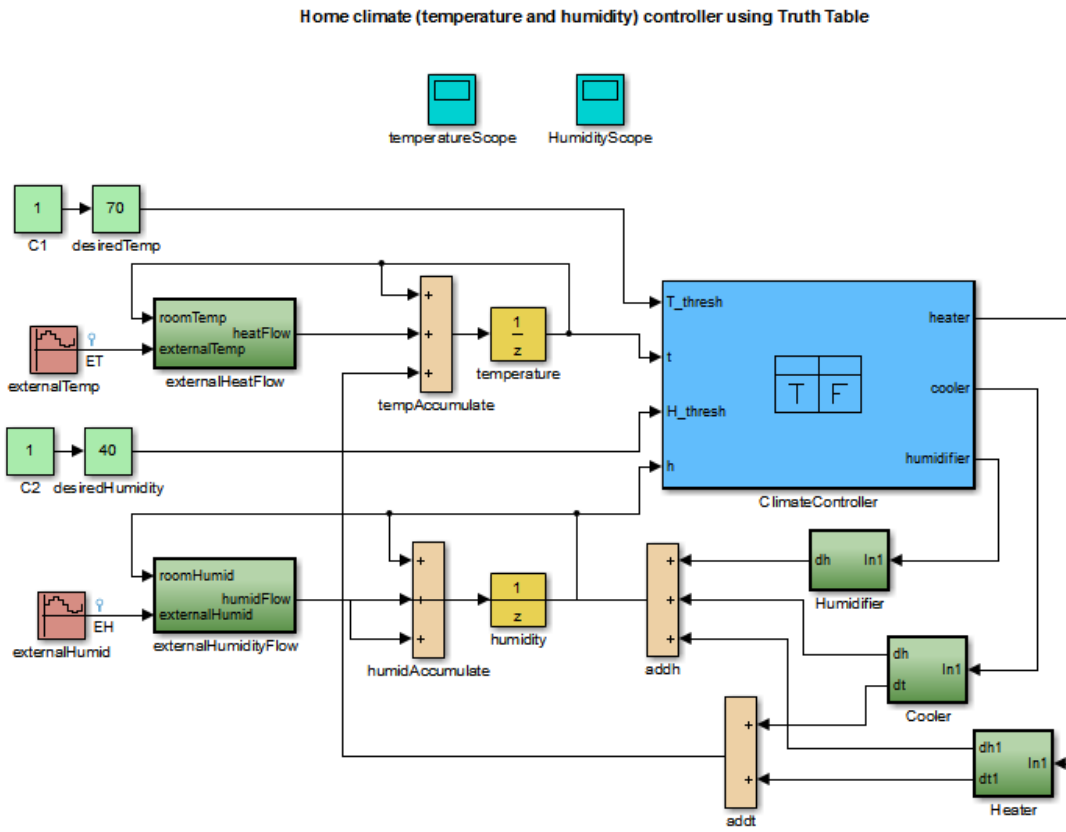
- It is a more direct approach, especially if your model requires only a single truth table.
- You can define truth table inputs and outputs to have inherited types and sizes.

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 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.

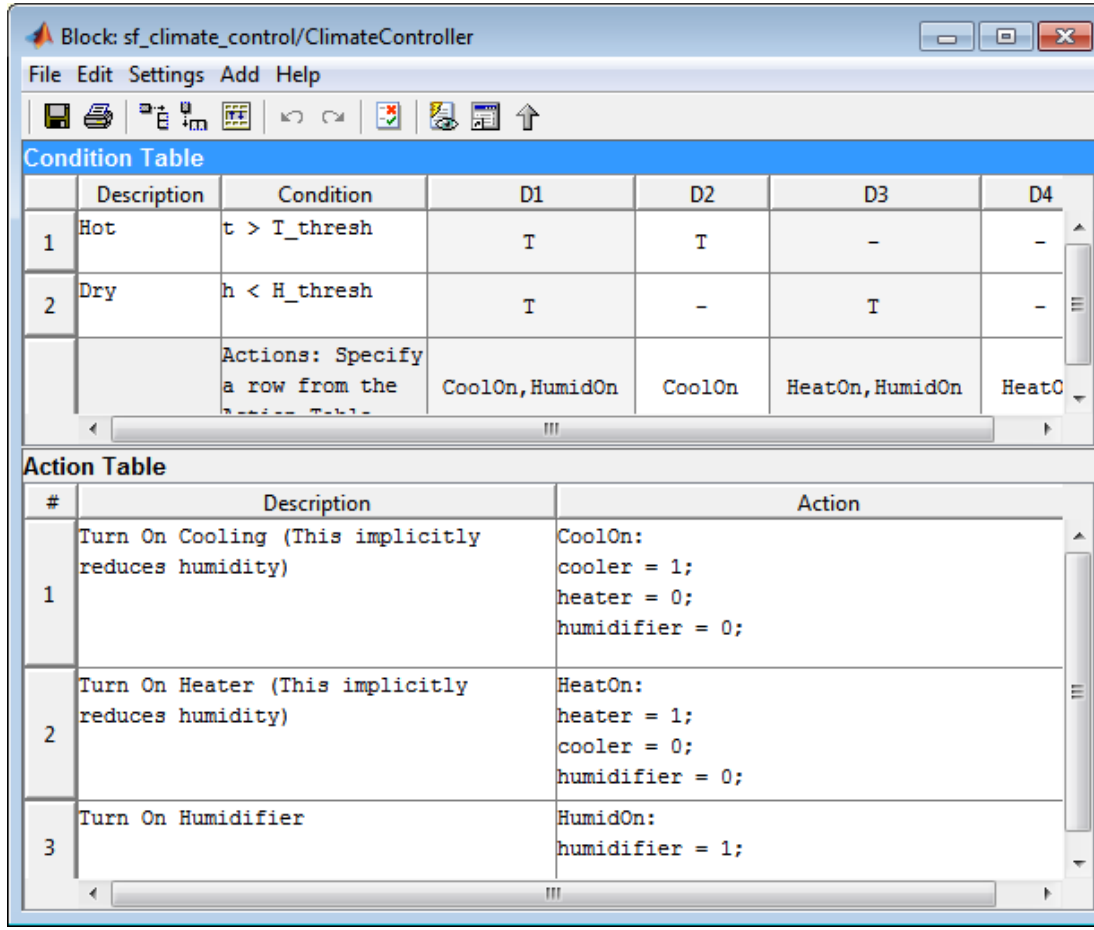
The following model, `sf_climate_control`, shows a home environment controller that attempts to maintain a selected temperature and humidity. The model has a Truth Table

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 `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`.



The inputs `t` and `h` define the conditions, and the outputs `heater`, `cooler`, and `humidifier` define the actions for this Truth Table block.

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

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: • Local • Constant • Parameter • Data store memory
Input trigger		An *input trigger* 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: • Rising edge • Falling edge • Either rising or falling edge • Function call For more information, see “Define Events”.
Function-call output		A *function-call output* triggers a function call to a subsystem. For more information, see “Function-Call Subsystems” (Simulink) 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.

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

Parameters

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

Characteristics

Direct Feedthrough	Yes
Sample Time	Sample based
Scalar Expansion	N/A
Dimensionalized	Yes
Zero-Crossing Detection	No

Introduced before R2006a