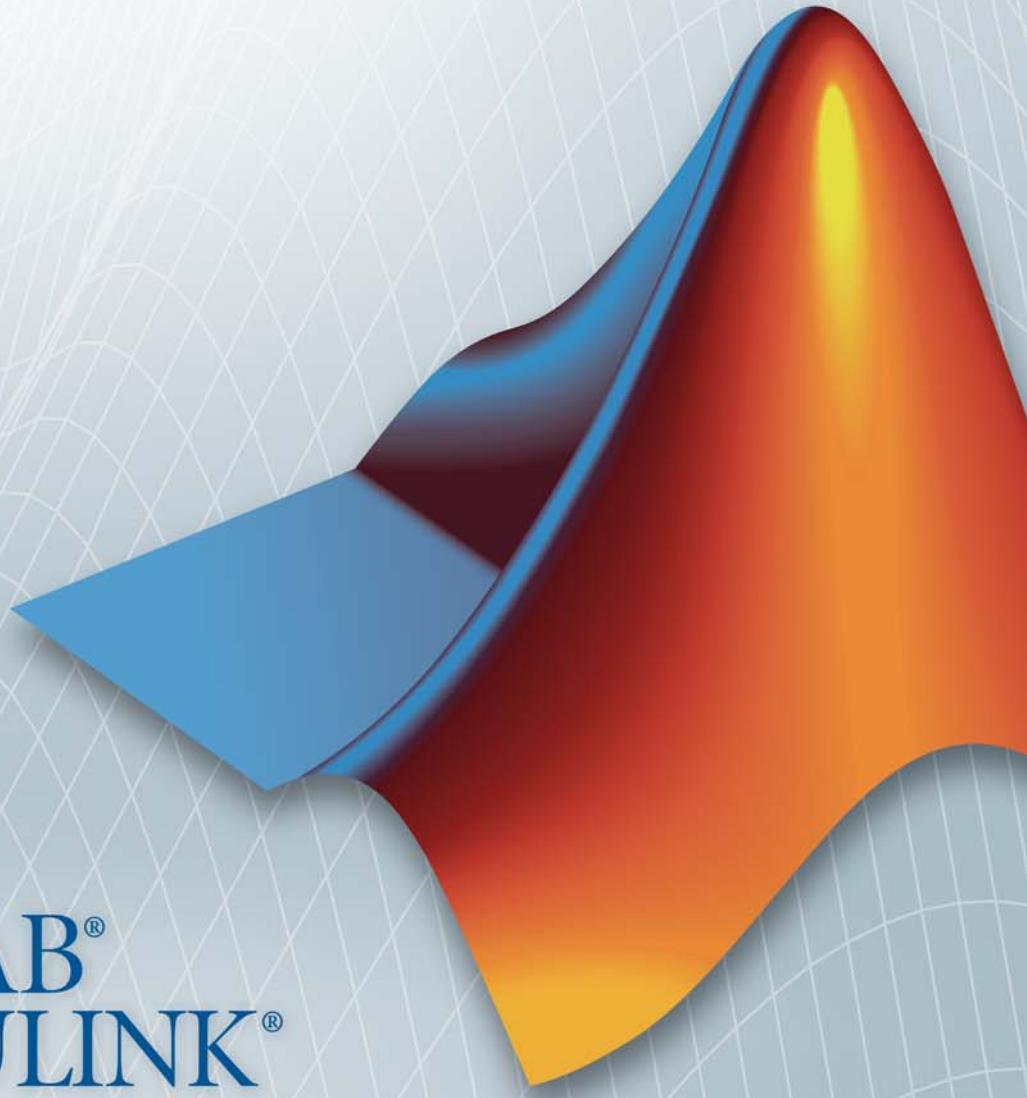
Stateflow®

Reference

R2012b



MATLAB®
&SIMULINK®

MathWorks®

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support

`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`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® Reference

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `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` 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)
September 2012	Online only	Revised for Version 8.0 (Release 2012b)

Contents

1

Function Reference

Object Retrieval

`sfclipboard`	Stateflow clipboard object
`sfgco`	Recently selected objects in chart
`sfroot`	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

2

Functions — Alphabetical List

sfclipboard

Purpose

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];
```

See Also

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

Tutorials

- "Copying Objects"
- "Quick Start for the Stateflow API"

How To

- "Getting a Handle on Stateflow API Objects"
- "Accessing the Chart Object"

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` | `sfopen` | `stateflow`

sfdebugger

Purpose

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`

How To

- "Debug Run-Time Errors in a Chart"

Purpose

Open Model Explorer

Syntax

```
sfexplr
```

Description

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

See Also

`sfdebugger | sfhelp | sflib`

How To

- "Using the Model Explorer with Stateflow Objects"

sfgco

Purpose

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;
```

See Also

`sfnew` | `sfroot` | `stateflow`

Tutorials

- "Quick Start for the Stateflow API"

How To

- "Getting a Handle on Stateflow API Objects"

- “Zooming a Chart Object with the API”

sfhelp

Purpose

Open Stateflow online help

Syntax

```
sfhelp
```

Description

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

See Also

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

Purpose

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`

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 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
`-MATLAB`	Use a chart that supports MATLAB expressions in Stateflow actions
`-STT`	Use a state transition 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` | `sfroot` | `sfsave` | `stateflow`

How To

- “Creating an Empty State Chart”
- “Creating Mealy and Moore Charts”
- “Building a Model with a Stateflow Truth Table”
- “C Syntax for States and Transitions”

sfopen

Purpose

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`

Purpose

Print graphical view of charts

Syntax

```
sfprint
sfprint(objects)
sfprint(objects,format)
sfprint(objects,format,output_option)
sfprint(objects,format,output_option,print_entire_chart)
```

Description

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

`sfprint(objects)` prints all charts in *`objects`* to a default printer.

`sfprint(objects,format)` prints all charts in *`objects`* in the specified *`format`* to a default printer.

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

`sfprint(objects,format,output_option,print_entire_chart)` prints all charts in *`objects`* in the specified *`format`* to the file or printer specified in *`output_option`*. Prints a complete or current view of charts as specified in *`print_entire_chart`*.

If the *`format`* argument is absent, `sfprint` generates a PostScript file to the default printer. If the *`output_option`* argument is absent, the name of the chart in the current folder becomes the output file name.

Input Arguments

objects

Objects containing charts to print:

`'literal_string'`	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
`'epsc'`	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:

`'`*`output_file_name`*`'`	Send output to a file called *`output_file_name`*
`'clipboard'`	Copy output to the clipboard

`'file'`	Send output to a default file with the name *`path_to_chart.file_extension`*, such as `sf_pool_Init.jpg`
`'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')
```

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

Purpose

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;
```

See Also

`sfclipboard | sfgco`

Tutorials

- "Quick Start for the Stateflow API"

How To

- "Getting a Handle on Stateflow API Objects"
- "Accessing the Chart Object"

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.

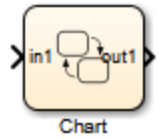
Examples

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

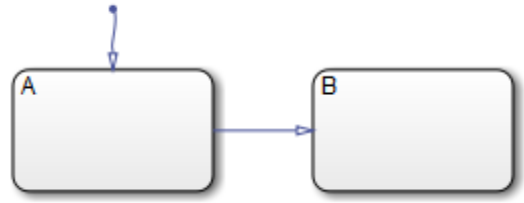
```
bdclose('all');

% 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 block library. From this library, you can drag Stateflow blocks into models or access the Stateflow Examples Library.

See Also

`sflib` | `sfnew`

How To

- "Creating an Empty State Chart"

3

Block Reference

MATLAB Chart

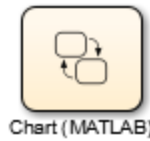
Purpose

Finite state machine that uses MATLAB expressions to implement control logic

Library

Stateflow

Description



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 operating 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.

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.

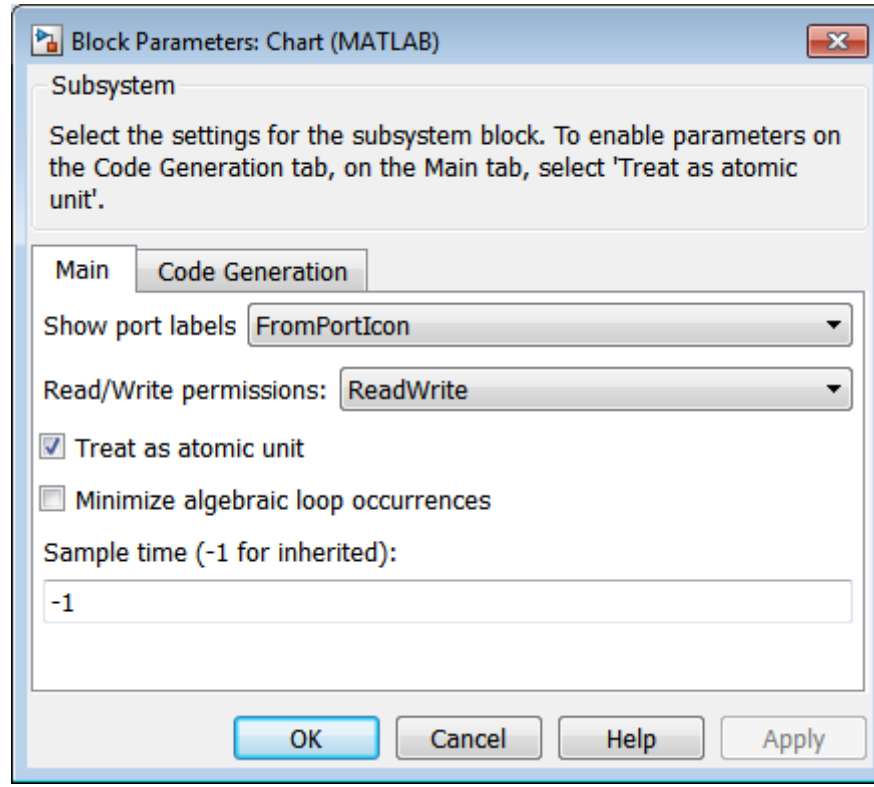
A MATLAB Chart block uses MATLAB expressions to implement control logic. For more information, see "MATLAB as the Action Language for Stateflow Charts".

Data Type Support

The MATLAB Chart block accepts input 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.

Floating-point and Boolean inputs pass through the block unchanged.

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 **Sample time** parameter
Scalar Expansion	N/A

Dimensionalized	Yes
Zero-Crossing Detection	No, because this block does not support continuous-time modeling.

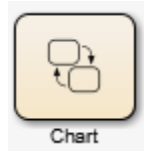
Purpose

Finite state machine that uses C expressions to implement control logic

Library

Stateflow

Description



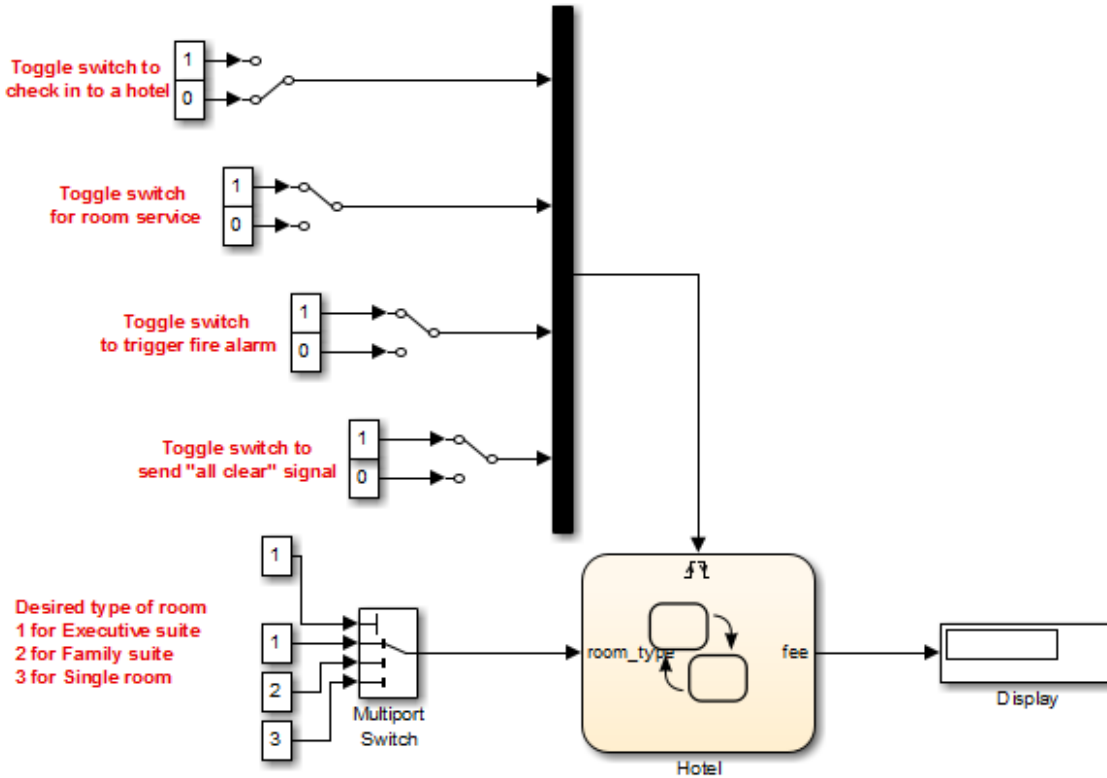
A *finite state machine* is a representation of an event-driven (reactive) system. In an event-driven system, the system responds by making a transition from one state (mode) to another. This action occurs in response to an event, as long as the condition defining the change is true.

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.

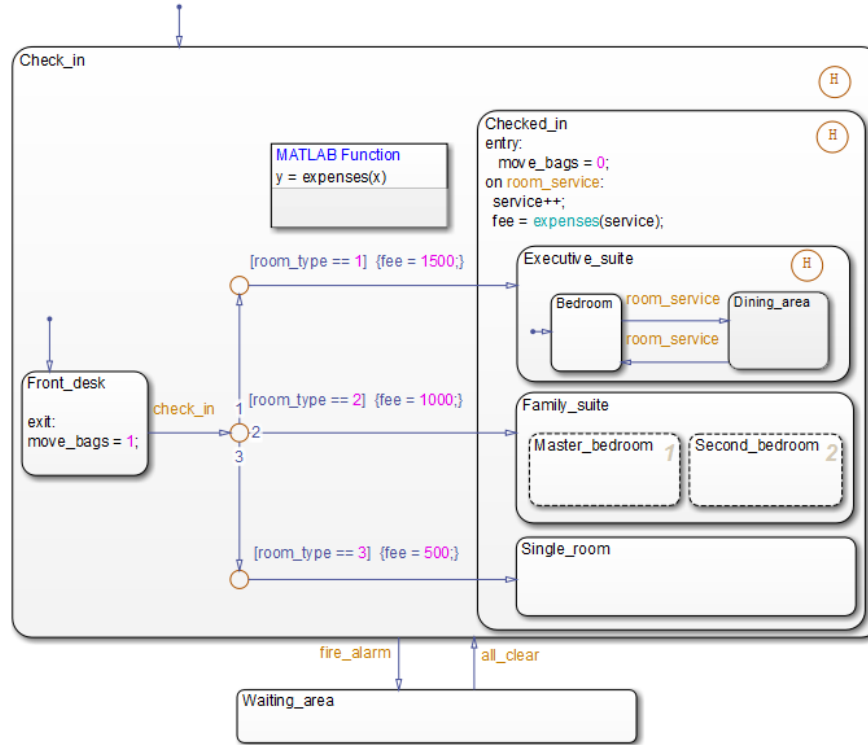
You can use Stateflow charts to control a physical plant in response to events such as a temperature or pressure sensor, or clock or user-driven events. For example, you can use a state machine to represent the automatic transmission of a car. The transmission has these operating states: park, reverse, 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.

A C Chart block uses C expressions to implement control logic.

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



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



By default, the C Chart block uses *Classic* chart semantics, but you can also specify Mealy or Moore semantics, as described in "Creating Mealy and Moore Charts".

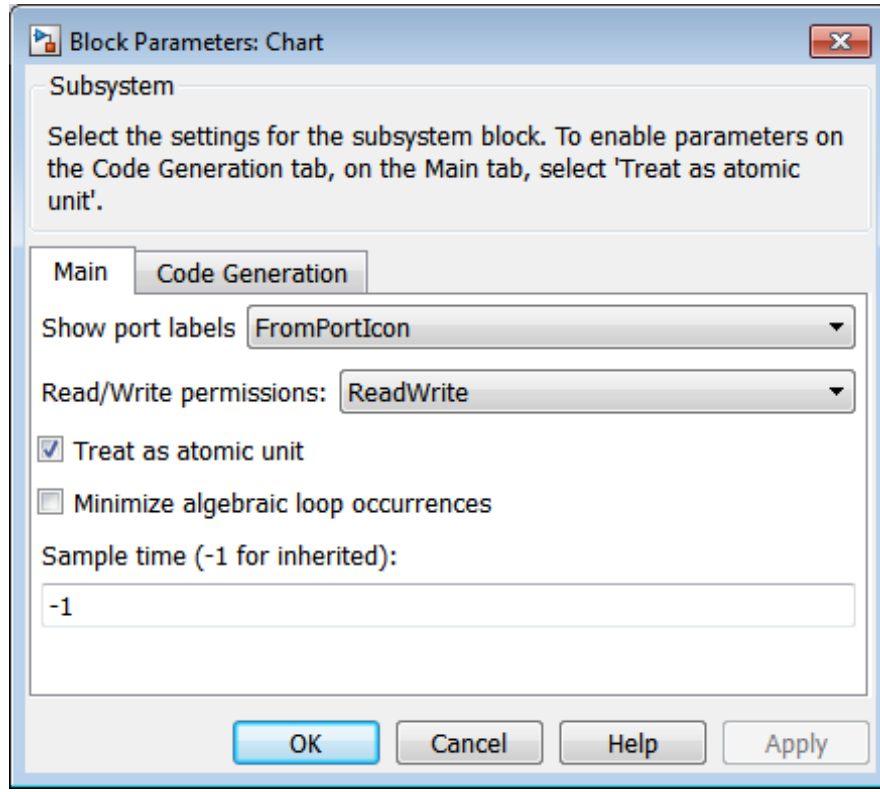
Data Type Support

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

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.

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 Enable Zero-Crossing Detection".

State Transition Table

Purpose

Represent modal logic in tabular format

Library

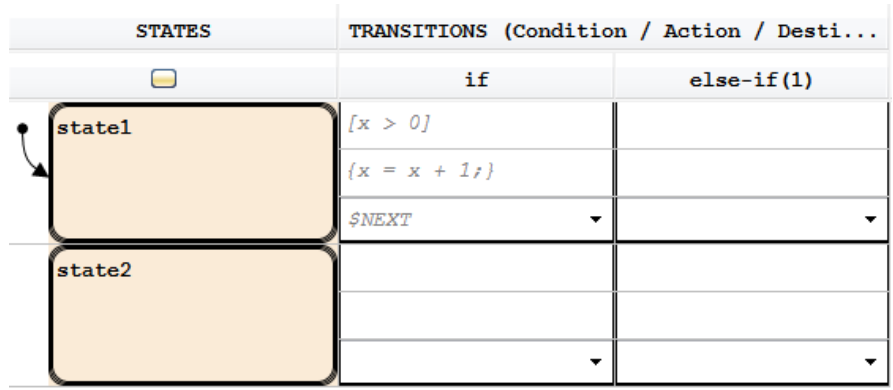
Stateflow

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 Editor Operations" in the Stateflow documentation.

Adding Data and Events

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

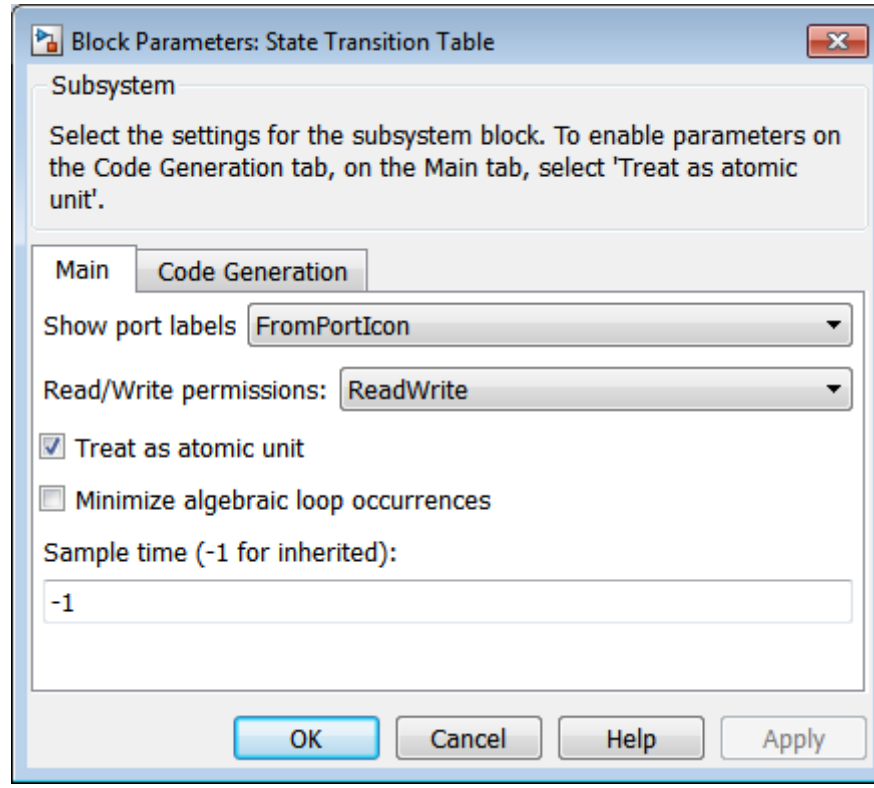
Element	Menu	Description
Inputs and outputs	**Table > Add Inputs & Outputs > Data Input from Simulink** **Table > Add Inputs & Outputs > Data Output to Simulink**	You can add inputs from the model and outputs to the model.
Data	**Table > Add Other Elements**	You can add these types of data: • Local • Constant • Parameter • Data store memory

Element	Menu	Description
Input events	**Table > 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	**Table > 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" .

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 and Dialog Box



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

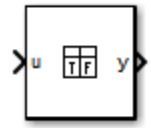
Dimensionalized	Yes
Zero-Crossing Detection	No, because this block does not support continuous-time modeling.

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 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:

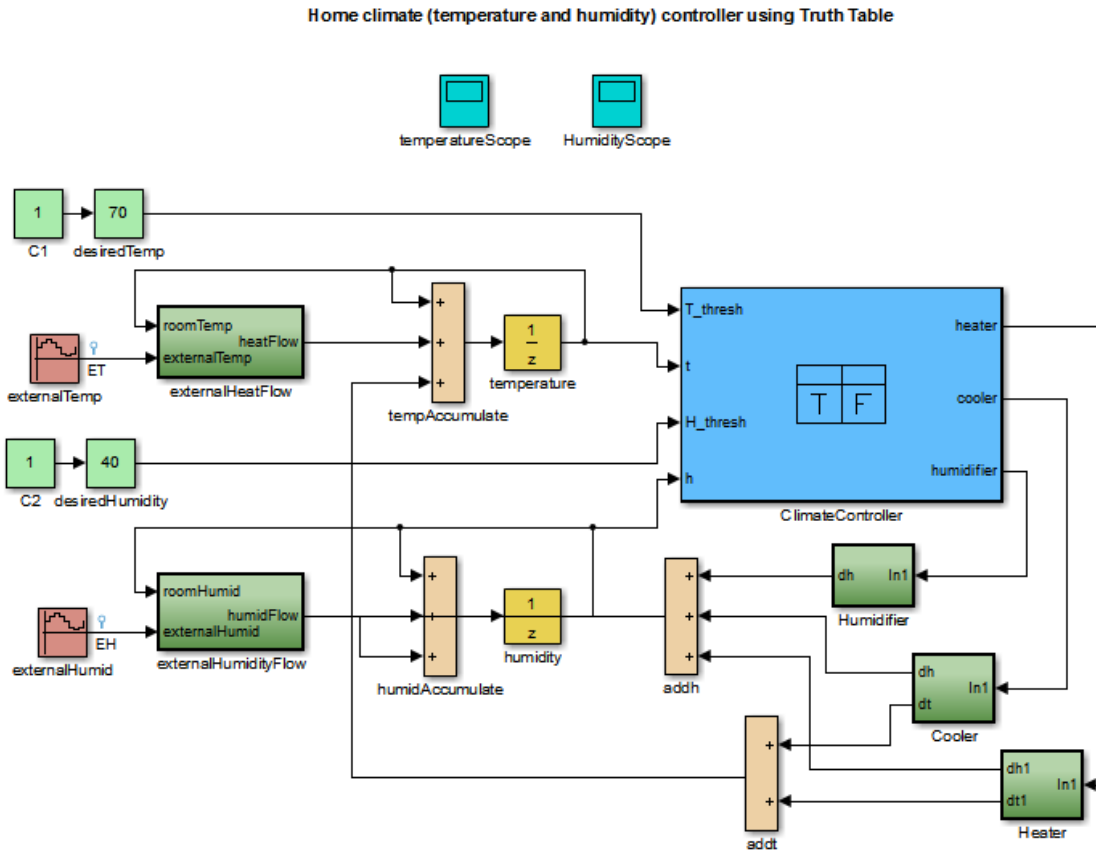
- It is a more direct approach, especially if your model requires only a single truth table.
- You can define truth table inputs and outputs to have inherited types and sizes.

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.

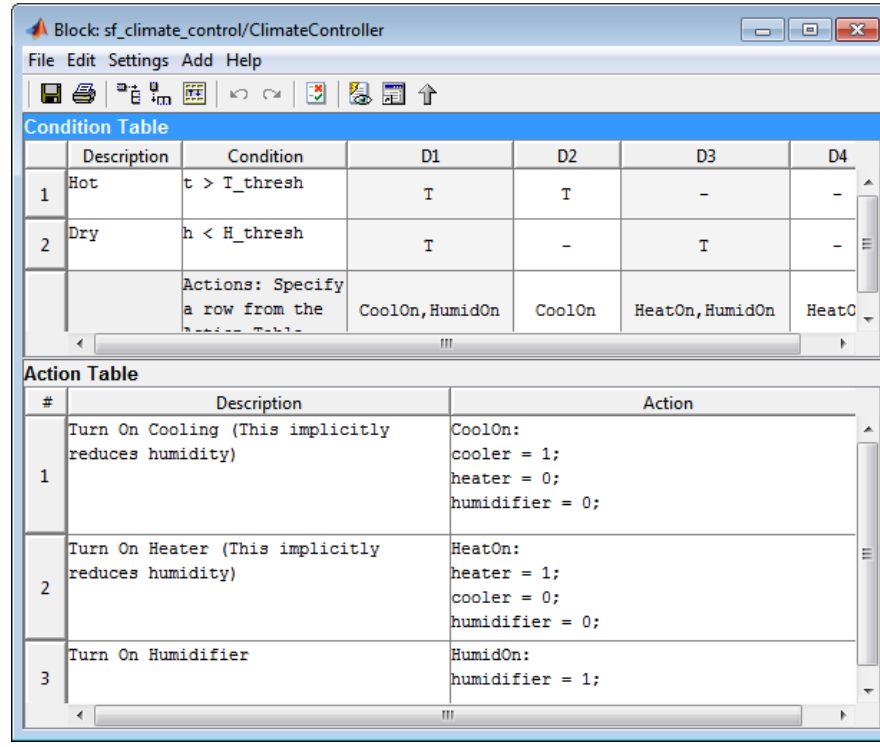
The following model, `sf_climate_control`, shows a home environment controller that attempts to maintain a selected temperature and humidity. The model has a Truth Table block, `ClimateController`,

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

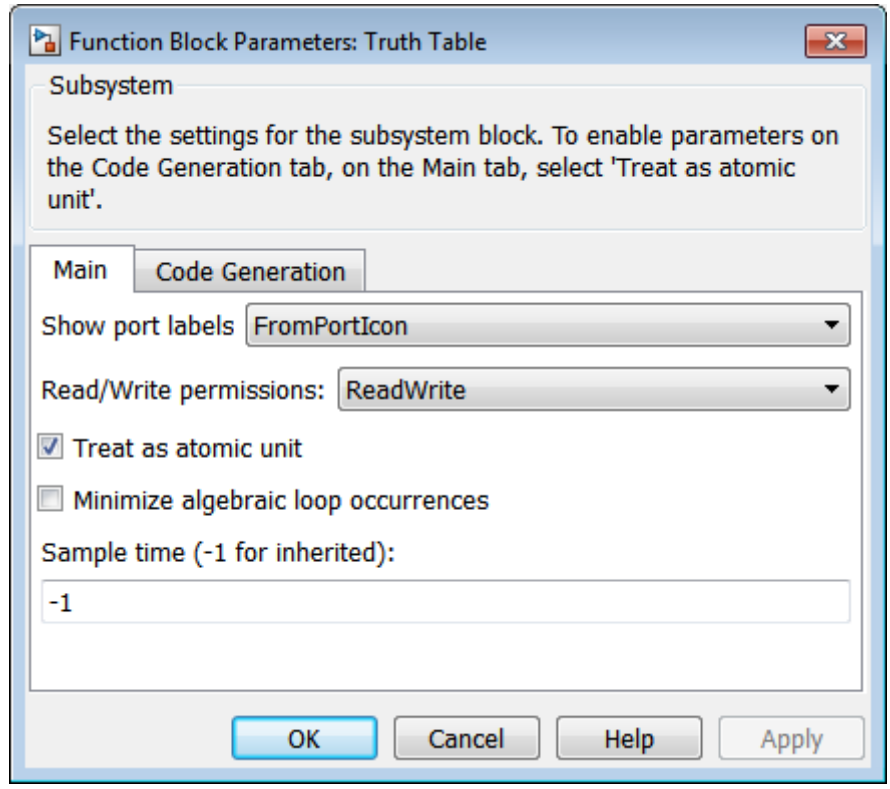
Element	Tool	Description
		For more information, see "How to Define Events".
Function-call output	*fx*	A *function-call output* triggers a function call to a subsystem. For more information, see "Function-Call Subsystems" in the Simulink documentation.

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 "Add Frame-Based Signals" in the Simulink documentation).

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

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 **Sample time** parameter
Scalar Expansion	N/A

Dimensionalized	Yes
Zero-Crossing Detection	No

Index