

**Stateflow<sup>®</sup>**

Reference



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

R2019a



# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

*Stateflow*<sup>®</sup> Reference

© COPYRIGHT 2006–2019 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)         |
| 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)         |
| September 2017 | Online only | Revised for Version 9.0 (Release 2017b)         |
| March 2018     | Online only | Revised for Version 9.1 (Release 2018a)         |
| September 2018 | Online only | Revised for Version 9.2 (Release 2018b)         |
| March 2019     | Online only | Revised for Version 10.0 (Release 2019a)        |



|          |                                      |
|----------|--------------------------------------|
| <b>1</b> | <b>Functions – Alphabetical List</b> |
| <b>2</b> | <b>Operators – Alphabetical List</b> |
| <b>3</b> | <b>Block Reference</b>               |



# 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];
```



## See Also

sfgco | sfnew | sfgroot | stateflow

## Topics

*“Copy and Paste Stateflow Objects”*

*“Create Charts by 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;
```

### See Also

[sfnew](#) | [sfroot](#) | [stateflow](#)

### Topics

[“Create Charts by Using the Stateflow API”](#)  
[“Getting a Handle on Stateflow API Objects”](#)  
[“View Stateflow Graphical Objects”](#)

**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 Simulink model that contains an empty Stateflow block

## Syntax

```
sfnew
sfnew chart_type
sfnew model_name
sfnew chart_type model_name
```

## Description

`sfnew` creates an untitled Simulink model that contains an empty Stateflow 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` that contains an empty chart.

`sfnew chart_type model_name` creates a model called `model_name` that contains an empty block of type `chart_type`.

## Examples

### Untitled Model with Chart

Create an untitled model that contains an empty Stateflow chart that uses MATLAB as the action language.

```
sfnew
```

## Untitled Model with Truth Table

Create an untitled model called `MyModel` that contains an empty Stateflow truth table block.

```
sfnew -TT
```

## Named Model with Chart

Create a model called `MyModel` that contains an empty Stateflow chart that uses MATLAB as the action language.

```
sfnew 'MyModel'
```

## Named Model with Moore Chart

Create a model called `MyModel` that contains an empty Stateflow chart that uses Moore semantics.

```
sfnew -Moore 'MyModel'
```

## Input Arguments

### `chart_type` — Type of block

`-MATLAB` (default) | `-C` | `-Mealy` | `-Moore` | `-STT` | `-TT`

Type of Stateflow block to add to empty model, specified as one of these options.

| Option               | Description                                       |
|----------------------|---|
| <code>-MATLAB</code> | Chart that supports MATLAB expressions in actions |
| <code>-C</code>      | Chart that supports C expressions in actions      |
| <code>-Mealy</code>  | Chart that supports Mealy machine semantics       |

| Option | Description                                 |
|--------|---|
| -Moore | Chart that supports Moore machine semantics |
| -STT   | State Transition Table                      |
| -TT    | Truth Table                                 |

### **model\_name** — Name of model

character vector

Name of the Simulink model, specified as a character vector.

## Tips

- The default action language for new charts is 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”.
- To create a standalone chart that you can execute as a MATLAB object, open the Stateflow editor by using the `edit` function. For example, at the MATLAB Command Window, enter:

```
edit chart.sfx
```

For more information, see “Create Stateflow Charts for Execution as MATLAB Objects”.

## See Also

`sfhelp` | `sfprint` | `sfroot` | `sfsave` | `stateflow`

## Topics

“Differences Between MATLAB and C as Action Language Syntax”

“Overview of Mealy and Moore Machines”

“Reuse Combinatorial Logic by Defining Truth Tables”

“State Transition Tables in Stateflow”

**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, 'tiff', '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' | 'tiff'`

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

|                       |  |
|-----------------------|--|
| <code>'bitmap'</code> | Save the chart image to the clipboard as a bitmap (for Windows® operating systems only)  |
| <code>'jpg'</code>    | Generate a JPEG file   |
| <code>'meta'</code>   | Save the chart image to the clipboard as a metafile (for Windows operating systems only) |
| <code>'pdf'</code>    | Generate a PDF file  |
| <code>'png'</code>    | Generate a PNG file  |
| <code>'svg'</code>    | Generate an SVG file   |
| <code>'tiff'</code>   | 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 <i>chart_name.file_extension</i> . 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**

`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;
```

## See Also

sfclipboard | sfgco

## Topics

“Create Charts by 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;

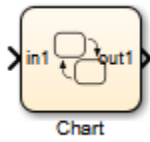
% Add a default transition to state A
dtA = Stateflow.Transition(chart1);
dtA.Destination = sA;
dtA.DestinationOClock = 0;
x = sA.Position(1)+sA.Position(3)/2;
y = sA.Position(2)-30;
dtA.SourceEndPoint = [x y];

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

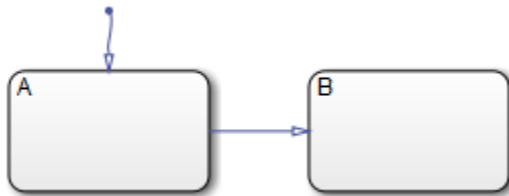
% Add an output out1
d2 = Stateflow.Data(chart1);
d2.Scope = 'Output';
d2.Name = 'out1';

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

Here is the resulting model:



Here is the resulting chart:



## See Also

[find](#) | [sfclose](#) | [sfnew](#) | [sfopen](#) | [sfroot](#)

## Topics

“Create Charts by Using the Stateflow API”

“Create Charts by Using a MATLAB Script”

**Introduced before R2006a**

## stateflow

Open Stateflow library window and create Simulink model that contains an empty chart

### Syntax

```
stateflow
```

### Description

`stateflow` creates an untitled Simulink model that contains an empty Stateflow 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.

### Tips

- To only create a Simulink model that contains an empty Stateflow block, use the `sfnew` function.
- To only open the Stateflow block library, use the `sflib` function.
- To create a standalone chart that you can execute as a MATLAB object, open the Stateflow editor by using the `edit` function. For example, at the MATLAB Command Window, enter:

```
edit chart.sfx
```

For more information, see “Create Stateflow Charts for Execution as MATLAB Objects”.

### See Also

```
edit | sflib | sfnew
```

**Introduced before R2006a**





# Operators — Alphabetical List

---

# after

Control chart execution with the `after` operator

## Syntax

```
after(n,E)  
after(n,time_unit)
```

## Description

`after(n,E)` returns `true` if the base event `E` has occurred at least `n` times since activation of the associated state. Otherwise, the operator returns `false`.

In a chart with no input events, `after(n,tick)` or `after(n,wakeup)` returns `true` if the chart has woken up `n` times or more since activation of the associated state.

The `after` operator resets the counter for `E` to 0 each time the associated state reactivates.

`after(n,time_unit)` returns `true` if `n` units of simulation time have elapsed since activation of the associated state. Otherwise, the operator returns `false`. Specify `time_unit` as seconds (`sec`), milliseconds (`msec`), or microseconds (`usec`).

The `after` operator resets the counter for `sec`, `msec`, and `usec` to 0 each time the associated state reactivates.

## Examples

### Event Based State Action (on after)

A status message appears during each CLK cycle, starting 5 clock cycles after activation of the state.

```
on after(5,CLK): status('on');
```

### Event Based Transition

A transition out of the associated state occurs only on broadcast of a ROTATE event, but no sooner than 10 CLK cycles after activation of the state.

```
ROTATE[after(10,CLK)]
```

### Absolute-Time Based State Action (on after)

After 12.3 seconds of simulation time since activation of the state, temp variable becomes LOW.

```
on after(12.3,sec): temp = LOW;
```

### Absolute-Time Based Transition

After 8 milliseconds of simulation time have passed since activation of the state, a transition out of the associated state occurs.

```
after(8,msec)
```

## Tips

- You can use quotation marks to enclose the keywords 'tick', 'wakeup', 'sec', 'msec', and 'usec'. For example, after(5, 'tick') is equivalent to after(5,tick).

## See Also

at | before | every

## Topics

“Control Chart Execution by Using Temporal Logic”

**Introduced in R2014b**

## ascii2str

Convert array of type `uint8` to string

### Syntax

```
dest = ascii2str(A)
```

### Description

`dest = ascii2str(A)` converts ASCII values in array `A` of type `uint8` to a string.

### Examples

#### Array of Type `uint8` to String

Return string "Hi!"

```
A[0] = 72;  
A[1] = 105;  
A[2] = 33;  
dest = ascii2str(A);
```

### Tips

- Use in Stateflow charts that use C as the action language.

### See Also

`str2ascii` | `strcpy`

### Topics

“Manage Textual Information by Using Strings”

“Share String Data with Custom C Code”

**Introduced in R2018b**

# at

Control chart execution with the `at` operator

## Syntax

`at(n,E)`

## Description

`at(n,E)` returns `true` only at the  $n^{\text{th}}$  occurrence of the base event `E` since activation of the associated state. Otherwise, the operator returns `false`.

In a chart with no input events, `at(n,tick)` or `at(n,wakeup)` returns `true` if the chart has woken up for the  $n^{\text{th}}$  time since activation of the associated state.

The `at` operator resets the counter for `E` to 0 each time the associated state reactivates.

## Examples

### Event Based State Action (on at)

A status message `on` appears at exactly 10 CLK cycles after activation of the state.

```
on at(10,CLK): status('on');
```

### Event Based Transition

A transition out of the associated state occurs only on broadcast of a `ROTATE` event, at exactly 10 CLK cycles after activation of the state.

ROTATE[at(10,CLK)]

### Tips

- You can use quotation marks to enclose the keywords 'tick' and 'wakeup'. For example, at(5, 'tick') is equivalent to at(5,tick).
- Use of at as an absolute-time temporal logic operator is not supported. Use the after operator instead. For more information, see “Use the after Operator to Replace the at Operator”.

### See Also

after | before | every

### Topics

“Control Chart Execution by Using Temporal Logic”

**Introduced in R2014b**



# before

Control chart execution with the before operator

## Syntax

```
before(n,E)  
before(n,time_unit)
```

## Description

`before(n,E)` returns `true` if the base event `E` has occurred fewer than `n` times since activation of the associated state. Otherwise, the operator returns `false`.

In a chart with no input events, `before(n,tick)` or `before(n,wakeup)` returns `true` if the chart has woken up fewer than `n` times since activation of the associated state.

The `before` operator resets the counter for `E` to 0 each time the associated state reactivates.

`before(n,time_unit)` returns `true` if fewer than `n` units of simulation time have elapsed since activation of the associated state. Otherwise, the operator returns `false`. Specify `time_unit` as seconds (`sec`), milliseconds (`msec`), or microseconds (`usec`).

The `before` operator resets the counter for `sec`, `msec`, and `usec` to 0 each time the associated state reactivates.

## Examples

### Event Based State Action (on before)

The `temp` variable increments once per CLK cycle until the state reaches the MAX limit.

```
on before(MAX,CLK): temp++;
```

### Event Based Transition

A transition out of the associated state occurs only on broadcast of a ROTATE event, but no later than 10 CLK cycles after activation of the state.

```
ROTATE[before(10,CLK)]
```

### Absolute-Time Based Transition

If the variable `temp` exceeds 75 and fewer than 12.34 seconds have elapsed since activation of the state, a transition out of the associated state occurs.

```
[temp > 75 && before(12.34,sec)]
```

## Tips

- You can use quotation marks to enclose the keywords 'tick', 'wakeup', 'sec', 'msec', and 'usec'. For example, `before(5, 'tick')` is equivalent to `before(5,tick)`.

## See Also

after | at | every

## Topics

“Control Chart Execution by Using Temporal Logic”

**Introduced in R2014b**

## count

Control chart execution with the count operator

### Syntax

count(C)

### Description

The count(C) operator returns a double value equivalent to the number of ticks after the conditional expression, C, becomes true. The count operator is reset if the conditional expression becomes false. If the count operator is used within a state, it is reset when the state that contains it is entered. If the count operator is used on a transition, it is reset when the source state for that transition is entered.

The value for count is dependent on the step size. Changing the solver or step size for your Simulink model affects the result of Stateflow charts that include the count operator.

To ensure that your Stateflow chart simulates without error, do not use count with these objects:

- Continuous time charts
- Graphical, MATLAB, or Simulink functions
- Simulink based states
- Transitions that can be reached from multiple states
- Default transitions

Use the count operator in charts that use C or MATLAB as the action language.

### Examples

### **Absolute-Time Based Transition**

The transition occurs when the value of `data` has been greater than or equal to 2 for longer than 5 ticks.

```
[count(data >= 2) > 5]
```

### **State Action**

When the state is exited, `x` is set to the number of ticks that `data` has been greater than 5.

```
ex: x = count(data>5)
```

### **See Also**

`duration` | `elapsed` | `temporalCount`

### **Topics**

“Control Chart Execution by Using Temporal Logic”

### **Introduced in R2019a**

# discard

Discard message

## Syntax

```
discard(message_name)
```

## Description

`discard(message_name)` discards a valid message. After a chart discards a message, it can remove another message from the queue in the same time step. A chart cannot access the data of a discarded message.

## Examples

### Discard Message in State Action

Check the queue for message M. If a message is present, remove it from the queue. If the message has a data value equal to 3, discard the message.

```
during:  
if receive(M) == true  
  if M.data == 3  
    discard(M);  
  end  
end
```

## See Also

receive

## Topics

“Control Message Activity in Stateflow Charts”

**Introduced in R2018b**

# duration

Control chart execution with the duration operator

## Syntax

```
duration(C)
```

## Description

`duration(C)` returns the number of seconds after the conditional expression, `C`, becomes true. The duration operator is reset if the conditional expression becomes false. If the duration operator is used within a state, it is reset when the state that contains it is entered. If the duration operator is used on a transition, it is reset when the source state for that transition is entered.

## Examples

### Absolute-Time Based Transition

The transition occurs when the value of `x` has been greater than or equal to `0` for longer than `0.1` seconds.

```
[duration(x >= 0) > 0.1]
```

## See Also

`count` | `elapsed` | `temporalCount`

## Topics

“Control Chart Execution by Using Temporal Logic”

“Control Oscillations by Using the duration Operator”

“Reduce Transient Signals by Using Debouncing Logic”

**Introduced in R2017a**



# elapsed

Control chart execution with the `elapsed` operator

## Syntax

```
elapsed(sec)
```

## Description

`elapsed(sec)` returns the simulation time in seconds (`sec`) that has elapsed since the activation of the associated state.

The `elapsed` operator resets the counter for `sec` to 0 each time the associated state reactivates.

## Examples

### Absolute-Time Based State Action

At the entry and during actions of the state, `y` is assigned the length of time that the state has been active.

```
en, du: y = elapsed(sec);
```

## Tips

- In state and transition actions, you can use quotation marks to enclose the keyword `'sec'`. For example:

```
y = elapsed('sec');
```

## **See Also**

count | duration | temporalCount

## **Topics**

“Control Chart Execution by Using Temporal Logic”

**Introduced in R2017a**

## every

Control chart execution with the every operator

### Syntax

```
every(n,E)  
every(n,time_unit)
```

### Description

`every(n,E)` returns `true` at every  $n^{\text{th}}$  occurrence of the base event `E` since activation of the associated state. Otherwise, the operator returns `false`.

In a chart with no input events, `every(n,tick)` or `every(n,wakeup)` returns `true` if the chart has woken up an integer multiple of `n` times since activation of the associated state.

The every operator resets the counter for `E` to 0 each time the associated state reactivates.

`every(n,time_unit)` returns `true` every `n` units of simulation time since activation of the associated state. Otherwise, the operator returns `false`. Specify `time_unit` as seconds (`sec`), milliseconds (`msec`), or microseconds (`usec`).

The every operator resets the counter for `sec`, `msec`, and `usec` to 0 each time the associated state reactivates.

Use of every as an absolute-time temporal logic operator is supported only in standalone charts for execution as MATLAB objects.

### Examples

### Event Based State Action (on every)

A status message on appears every 5 CLK cycles after activation of the state.

```
on every(5,CLK): status('on');
```

### Absolute-Time Based State Action (on every)

A status message is displayed every 2 seconds after activation of the state.

```
on every(2,sec): disp('Hello!');
```

## Tips

- You can use quotation marks to enclose the keywords 'tick', 'wakeup', 'sec', 'msec', and 'usec'. For example, `every(5, 'tick')` is equivalent to `every(5, tick)`.
- Use of `every` as an absolute-time temporal logic operator is supported only in standalone charts for execution as MATLAB objects. In a Simulink model, use an outer self-loop transition with the `after` operator instead. For more information, see “Use an Outer Self-Loop Transition with the after Operator to Replace the every Operator”.

## See Also

after | at | before

## Topics

“Control Chart Execution by Using Temporal Logic”

**Introduced in R2014b**

## forward

Forward message

### Syntax

```
forward(input_message_name,output_message_name)
```

### Description

`forward(input_message_name,output_message_name)` forwards a message from an input queue to an output port, or to and from local message queues. After a chart forwards a message, it can remove another message from the queue in the same time step.

### Examples

#### Forward an Input Message

Checks the input queue for message `M_in`. If a message is present, the chart removes the message from the queue and forwards it to the output port `M_out`. After the chart forwards the message, the message is no longer valid.

```
on M_in: forward(M_in,M_out);
```

### See Also

receive

### Topics

“Control Message Activity in Stateflow Charts”

**Introduced in R2018b**

# hasChanged

Detect change in data since last time step

## Syntax

```
tf = hasChanged(u)
```

## Description

`tf = hasChanged(u)` returns `true` if the value of `u` at the beginning of the current time step is different from the value of `u` at the beginning of the previous time step. If multiple input events occur in the same time step, `hasChanged` returns `true` when the value of `u` changes between input events.

The argument `u` can be:

- A scalar variable.
- A matrix or an element of a matrix. See “Supported Operations for Vectors and Matrices”.
- A structure or a field in a structure. See “Index and Assign Values to Stateflow Structures”.
- Any valid combination of structure fields or matrix indices.

Indices can be numbers or expressions that evaluate to a scalar value. If `u` is a matrix, `hasChanged` returns `true` if any element of `u` has changed value since the last time step or input event. If `u` is a structure, `hasChanged` returns `true` if any field of `u` has changed value since the last time step or input event.

The scope of `u` depends on the chart action language:

- MATLAB as the action language: Input only.
- C as the action language: Input, Output, Local, or Data Store Memory.

The argument `u` cannot be a nontrivial expression or a custom code variable.

# Examples

### Detect Change in Structure

Returns `true` if any field of the structure `struct` has changed value since the last time step or input event.

```
hasChanged(struct)
```

### Detect Change in Structure Field

Returns `true` if the structure field of `struct.field` has changed value since the last time step or input event.

```
hasChanged(struct.field)
```

### Detect Change in Matrix

Returns `true` if any element of the matrix `M` has changed value since the last time step or input event.

```
hasChanged(M)
```

### Detect Change in Matrix Element

Returns `true` if the element in row 1 and column 3 of the matrix `M` has changed value since the last time step or input event.

In charts that use MATLAB as the action language:

```
hasChanged(M(1,3))
```

In charts that use C as the action language:



hasChanged(M[0][2])

## Tips

- The `hasChanged` operator returns `false` if the chart writes to the data but does not change the data value.
- If you enable the chart option **Initialize Outputs Every Time Chart Wakes Up**, do not use an output as the argument of the `hasChanged` operator. With this option enabled, the `hasChanged` operator always returns `false`. For more information, see “Initialize Outputs Every Time Chart Wakes Up”.
- The `hasChanged` operator is supported only in Stateflow charts in Simulink models.

## See Also

`hasChangedFrom` | `hasChangedTo`

## Topics

“Detect Changes in Data Values”

“Supported Operations for Vectors and Matrices”

“Index and Assign Values to Stateflow Structures”

“Convert Scalars to Nonscalars by Using Scalar Expansion”

**Introduced in R2007a**

# hasChangedFrom

Detect change in data from specified value

## Syntax

```
tf = hasChangedFrom(u,v)
```

## Description

`tf = hasChangedFrom(u,v)` returns `true` if both of these conditions are true:

- The value of `u` at the beginning of the previous time step was equal to `v`.
- The value of `u` at the beginning of the current time step is not equal to `v`.

If multiple input events occur in the same time step, `hasChangedFrom` returns `true` when the value of `u` changes from the value `v` between input events.

The first argument `u` can be:

- A scalar variable.
- A matrix or an element of a matrix. See “Supported Operations for Vectors and Matrices”.
- A structure or a field in a structure. See “Index and Assign Values to Stateflow Structures”.
- Any valid combination of structure fields or matrix indices.

Indices can be numbers or expressions that evaluate to a scalar value.

The second argument `v` can be any expression that resolves to a value that is comparable with `u`:

- If `u` is a scalar, then `v` must resolve to a scalar value.
- If `u` is a matrix, then `v` must resolve to a matrix value with the same dimensions as `u`. The `hasChangedFrom` operator returns `true` if the previous value of `u` was equal to `v` and any element of `u` has changed value since the last time step or input event.

Alternatively, in a chart that uses C as the action language, *v* can resolve to a scalar value. The chart uses scalar expansion to compare *u* to a matrix whose elements are all equal to the value specified by *v*. See “Convert Scalars to NonScalars by Using Scalar Expansion”.

- If *u* is a structure, then *v* must resolve to a structure value whose field specification matches *u* exactly. The `hasChangedFrom` operator returns `true` if the previous value of *u* was equal to *v* and any field of *u* has changed value since the last time step or input event.

The scope of *u* depends on the chart action language:

- MATLAB as the action language: Input only.
- C as the action language: Input, Output, Local, or Data Store Memory.

The argument *u* cannot be a nontrivial expression or a custom code variable.

## Examples

### Detect Change in Structure

Returns `true` if the previous value of the structure `struct` was equal to `structValue` and any field of `struct` has changed value since the last time step or input event.

```
hasChangedFrom(struct,structValue)
```

### Detect Change in Structure Field

Returns `true` if the structure field `struct.field` has changed from the value 5 since the last time step or input event.

```
hasChangedFrom(struct.field,5)
```

### Detect Change in Matrix

Returns `true` if the previous value of the matrix `M` was equal to `matrixValue` and any element of `M` has changed value since the last time step or input event.

`hasChangedFrom(M, matrixValue)`

### Detect Change in Matrix Element

Returns `true` if the element in row 1 and column 3 of the matrix `M` has changed from the value 7 since the last time step or input event.

In charts that use MATLAB as the action language:

```
hasChangedFrom(M(1,3),7)
```

In charts that use C as the action language:

```
hasChangedFrom(M[0][2],7)
```

## Tips

- The `hasChangedFrom` operator returns `false` if the chart writes to the data but does not change the data value.
- If you enable the chart option **Initialize Outputs Every Time Chart Wakes Up**, do not use an output as the argument of the `hasChangedFrom` operator. With this option enabled, the `hasChangedFrom` operator always returns `false`. For more information, see “Initialize Outputs Every Time Chart Wakes Up”.
- The `hasChangedFrom` operator is supported only in Stateflow charts in Simulink models.

## See Also

`hasChanged` | `hasChangedTo`

## Topics

“Detect Changes in Data Values”

“Supported Operations for Vectors and Matrices”

“Index and Assign Values to Stateflow Structures”

“Convert Scalars to Nonscalars by Using Scalar Expansion”

**Introduced in R2007a**

# hasChangedTo

Detect change in data to specified value

## Syntax

```
tf = hasChangedTo(u,v)
```

## Description

`tf = hasChangedTo(u,v)` returns `true` if both of these conditions are true:

- The value of `u` at the beginning of the previous time step was not equal to `v`.
- The value of `u` at the beginning of the current time step is equal to `v`.

If multiple input events occur in the same time step, `hasChangedTo` returns `true` when the value of `u` changes to the value `v` between input events.

The first argument `u` can be:

- A scalar variable.
- A matrix or an element of a matrix. See “Supported Operations for Vectors and Matrices”.
- A structure or a field in a structure. See “Index and Assign Values to Stateflow Structures”.
- Any valid combination of structure fields or matrix indices.

Indices can be numbers or expressions that evaluate to a scalar value.

The second argument `v` can be any expression that resolves to a value that is comparable with `u`:

- If `u` is a scalar, then `v` must resolve to a scalar value.
- If `u` is a matrix, then `v` must resolve to a matrix value with the same dimensions as `u`. The `hasChangedTo` operator returns `true` if any element of `u` has changed value since the last time step or input event and the current value of `u` is equal to `v`.

Alternatively, in a chart that uses C as the action language, `v` can resolve to a scalar value. The chart uses scalar expansion to compare `u` to a matrix whose elements are all equal to the value specified by `v`. See “Convert Scalars to NonScalars by Using Scalar Expansion”.

- If `u` is a structure, then `v` must resolve to a structure value whose field specification matches `u` exactly. The `hasChangedTo` operator returns `true` if any field of `u` has changed value since the last time step or input event and the current value of `u` is equal to `v`.

The scope of `u` depends on the chart action language:

- MATLAB as the action language: Input only.
- C as the action language: Input, Output, Local, or Data Store Memory.

The argument `u` cannot be a nontrivial expression or a custom code variable.

## Examples

### Detect Change in Structure

Returns `true` if any field of `struct` has changed value since the last time step or input event and the current value of the structure `struct` is equal to `structValue`.

```
hasChangedTo(struct,structValue)
```

### Detect Change in Structure Field

Returns `true` if the structure field `struct.field` has changed to the value 5 since the last time step or input event.

```
hasChangedTo(struct.field,5)
```

### Detect Change in Matrix

Returns `true` if any element of `M` has changed value since the last time step or input event and the current value of the matrix `M` is equal to `matrixValue`.

`hasChangedTo(M, matrixValue)`

### Detect Change in Matrix Element

Returns `true` if the element in row 1 and column 3 of the matrix `M` has changed to the value 7 since the last time step or input event.

In charts that use MATLAB as the action language:

```
hasChangedTo(M(1,3),7)
```

In charts that use C as the action language:

```
hasChangedTo(M[0][2],7)
```

## Tips

- The `hasChangedTo` operator returns `false` if the chart writes to the data but does not change the data value.
- If you enable the chart option **Initialize Outputs Every Time Chart Wakes Up**, do not use an output as the argument of the `hasChangedTo` operator. With this option enabled, the `hasChangedTo` operator always returns `false`. For more information, see “Initialize Outputs Every Time Chart Wakes Up”.
- The `hasChangedTo` operator is supported only in Stateflow charts in Simulink models.

## See Also

`hasChanged` | `hasChangedFrom`

## Topics

“Detect Changes in Data Values”

“Supported Operations for Vectors and Matrices”

“Index and Assign Values to Stateflow Structures”

“Convert Scalars to Nonscalars by Using Scalar Expansion”

**Introduced in R2007a**



## isvalid

Determine if message is valid

### Syntax

```
isvalid(message_name)
```

### Description

`isvalid(message_name)` checks if a message is valid. A message is valid if the chart has removed it from the queue and has not forwarded or discarded it.

### Examples

#### Check Message in State Action

Determine whether the message `M` is valid. If the message is valid and has a data value equal to 6, discard the message.

```
during:  
if isvalid(M) == true  
    if M.data == 6  
        discard(M);  
    end  
end
```

### Tips

- Use the `isvalid` operator to check if a message is valid in a Simulink model that contains more than one Stateflow chart.

## **See Also**

discard | forward | receive

## **Topics**

“Control Message Activity in Stateflow Charts”

**Introduced in R2018b**

# length

Determine length of message queue

## Syntax

```
length(message_name)
```

## Description

`length(message_name)` checks the number of messages in a message queue.

## Examples

### Check Queue Length in State Action

Check the queue for message M. If a message is present, remove it from the queue. If exactly seven messages remain in the queue, increment the value of x.

```
during:  
if receive(M) == true  
    if length(M) == 7  
        x = x+1;  
    end  
end
```

## See Also

overflowed | receive

## Topics

“Control Message Activity in Stateflow Charts”

**Introduced in R2018b**

# overflowed

Determine when message queue overflows

## Syntax

`overflowed(message_name)`

## Description

`overflowed(message_name)` checks whether a message is lost because it was sent to a queue that was already full. In each time step, the value of this operator is set when a chart adds a message to, or removes a message from, a queue. It is invalid to use the `overflowed` operator before sending or retrieving a message in the same time step.

- To check the overflow status of an input message queue, first remove a message from the queue.
- To check the overflow status of an output message queue, first add a message to the queue.
- To check the overflow status of a local message queue, first add a message to the queue or remove a message from the queue.

## Examples

### Check for Overflow in Transition

Check the input or local queue for message M. If a message is present and the queue has overflowed, transition occurs.

`M[overflowed(M)]`

### Check for Overflow in State Action

Check the input or local queue for message M. If a message is present and the queue has overflowed, increment the value of x.

```
on M:  
if overflowed(M) == true  
    x = x+1;  
end
```

### Check for Overflow After Sending Message

Send message and check for overflow. If the queue overflows, increment the value of x.

```
entry:  
M.data = 3;  
send(M);  
if overflowed(M) == true  
    x = x+1;  
end
```

## Tips

- By default, when a message queue overflows, simulation stops with an error. To prevent a run-time error and allow the overflowed operator to dynamically react to dropped messages, set the value of the **Queue Overflow Diagnostic** property to Warning or None. For more information, see “Queue Overflow Diagnostic”.

## See Also

length | receive | send

## Topics

“Control Message Activity in Stateflow Charts”

“Set Properties for a Message”

**Introduced in R2018b**

# receive

Extract message from queue

## Syntax

```
receive(message_name)
```

## Description

`receive(message_name)` extracts an input or local message from queue. If a valid message exists, `receive` returns `true`. If a valid message does not exist but there is a message in the queue, the chart removes the message from the queue and `receive` returns `true`. If a valid message does not exist and there are no messages in the queue, `receive` returns `false`.

## Examples

### Extract Message in State Action

Check the queue for message `M` and increment the value of `x` if both of these conditions are true:

- A message is present in the queue.
- The data value of the message is equal to 3.

If a message is not present or if the data value is not equal to 3, then the value of `x` does not change. If a message is present, the chart removes it from the queue regardless of whether `x` is modified.

```
during:  
if receive(M) && M.data == 3
```

```
    x = x+1;  
end
```

### See Also

send

### Topics

“Control Message Activity in Stateflow Charts”

**Introduced in R2018b**



# send

Broadcast message or event

## Syntax

```
send(message_name)
send(output_event_name)
send(local_event_name, state_name)
send(state_name.local_event_name)
```

## Description

`send(message_name)` sends a local or output message.

`send(output_event_name)` sends an output event.

`send(local_event_name, state_name)` broadcasts a local event to `state_name` and any offspring of that state in the hierarchy.

`send(state_name.local_event_name)` broadcasts a local event to its parent state `state_name` and any offspring of that state in the hierarchy.

## Examples

### Broadcast Message

Send a local or output message `M` with a data value of 3.

```
M.data = 3;
send(M);
```

### **Broadcast Output Event**

Send an output event *e*.

```
send(e);
```

### **Broadcast Directed Local Event**

Send a local event *e* to state *A.A1* and any of its substates.

```
send(e,A.A1);
```

### **Broadcast by Using Qualified Event Name**

Send a local event *e* to its parent state *A* and any of its substates.

```
send(A.e);
```

## **See Also**

receive

## **Topics**

“Control Message Activity in Stateflow Charts”

“Activate a Simulink Block by Sending Output Events”

“Broadcast Local Events to Synchronize Parallel States”

## **Introduced in R2018b**

## str2ascii

Convert string to array of type `uint8`

### Syntax

```
A = str2ascii(str,n)
```

### Description

`A = str2ascii(str,n)` returns array of type `uint8` containing ASCII values for the first `n` characters in `str`, where `n` is a positive integer.

Use of variables or expressions for `n` is not supported.

### Examples

#### String to ASCII Values

Return `uint8` array `{-72,101,108,108,111}`.

```
A = str2ascii("Hello",5);
```

### Tips

- Use in Stateflow charts that use C as the action language.
- Enclose literal strings with single or double quotes.

### See Also

`ascii2str`

## **Topics**

“Manage Textual Information by Using Strings”

“Share String Data with Custom C Code”

**Introduced in R2018a**

# str2double

Convert string to double precision value

## Syntax

```
X = str2double(str)
```

## Description

`X = str2double(str)` converts the text in string `str` to a double-precision value.

`str` contains text that represents a number. Text that represents a number can contain:

- Digits
- A decimal point
- A leading + or - sign
- An e preceding a power of 10 scale factor

If `str2double` cannot convert text to a number, then it returns a NaN value.

## Examples

### String Containing Decimal Notation

Return a value of -12.345.

```
X = str2double("-12.345");
```

### String Containing Exponential Notation

Return a value of 123400.

```
X = strtodouble("1.234e5");
```

### Tips

- Use in Stateflow charts that use C as the action language.
- Enclose literal strings with single or double quotes.

### See Also

tostring

### Topics

“Manage Textual Information by Using Strings”

**Introduced in R2018a**

# strcat

Concatenate strings

## Syntax

```
dest = strcat(s1, ..., sN)
```

## Description

`dest = strcat(s1, ..., sN)` concatenates strings `s1, ..., sN`.

## Examples

### Concatenation of Strings

Concatenate strings to form "Stateflow".

```
s1 = "State";  
s2 = "flow";  
dest = strcat(s1,s2)
```

## Tips

- Use in Stateflow charts that use C as the action language.
- Enclose literal strings with single or double quotes.

## See Also

strcpy | substr

**Topics**

“Manage Textual Information by Using Strings”

**Introduced in R2018b**



# strcmp

Compare strings

## Syntax

```
tf = strcmp(s1,s2)
s1 == s2
s1 != s2
tf = strcmp(s1,s2,n)
```

## Description

`tf = strcmp(s1,s2)` compares strings `s1` and `s2`. Returns `0` if the two strings are identical. Otherwise returns a nonzero integer.

- The sign of the output value depends on the lexicographic ordering of the input strings `s1` and `s2`.
- The magnitude of the output value depends on the compiler that you use. This value can differ in simulation and generated code.

Strings are considered identical when they have the same size and content.

This operator is consistent with the C library function `strcmp` or the C++ function `string.compare`, depending on the compiler that you select for code generation. The operator behaves differently than the function `strcmp` in MATLAB.

`s1 == s2` is an alternative way to execute `strcmp(s1,s2) == 0`.

`s1 != s2` is an alternative way to execute `strcmp(s1,s2) != 0`.

`tf = strcmp(s1,s2,n)` returns `0` if the first `n` characters in `s1` and `s2` are identical.

## Examples

### Comparison by Using strcmp

Return a value of 0 (strings are equal).

```
tf = strcmp("abc", "abc");
```

Return a nonzero value (strings are not equal).

```
tf = strcmp("abc", "abcd");
```

### Comparison by Using ==

Return a value of true.

```
"abc" == "abc";
```

### Comparison by Using !=

Return a value of true.

```
"abc" != "abcd";
```

### Comparison of Substrings

Return a value of 0 (substrings are equal).

```
tf = strcmp("abc", "abcd", 3);
```

## Tips

- Use in Stateflow charts that use C as the action language.
- Enclose literal strings with single or double quotes.

## See Also

substr

## **Topics**

“Manage Textual Information by Using Strings”

**Introduced in R2018b**

# strcpy

Assign string value

## Syntax

```
dest = src  
strcpy(dest,src)
```

## Description

`dest = src` assigns string `src` to `dest`.

`strcpy(dest,src)` is an alternative way to execute `dest = src`.

## Examples

### Assignment by Using =

Assign string data to `s1` and `s2`.

```
s1 = 'hello';  
s2 = "good bye";
```

### Assignment by Using strcpy

Assign string data to `s3` and `s4`.

```
strcpy(s3, 'howdy');  
strcpy(s4, "so long");
```

## Tips

- Use in Stateflow charts that use C as the action language.
- Source and destination strings must refer to different symbols.
- Enclose literal strings with single or double quotes.

## See Also

### Topics

“Manage Textual Information by Using Strings”

**Introduced in R2018b**

# strlen

Determine length of string

## Syntax

```
L = strlen(str)
```

## Description

`L = strlen(str)` returns the number of characters in the string `str`.

## Examples

### Number of Characters in String

Return a value of 9.

```
L = strlen("Stateflow");
```

## Tips

- Use in Stateflow charts that use C as the action language.
- Enclose literal strings with single or double quotes.

## See Also

### Topics

“Manage Textual Information by Using Strings”

**Introduced in R2018b**

# substr

Extract substring from string

## Syntax

```
dest = substr(str,i,n)
```

## Description

`dest = substr(str,i,n)` returns the substring of length `n` starting at the `i`-th character of string `str`. Use zero-based indexing.

## Examples

### Extract Substring

Extract substring "Stateflow" from a longer string.

```
str = "Stateflow rule the waves";  
dest = substr(str,0,9);
```

## Tips

- Use in Stateflow charts that use C as the action language.
- Use zero-based indexing.
- Enclose literal strings with single or double quotes.

## See Also

`strcat` | `strcpy` | `strlen`



## **Topics**

“Manage Textual Information by Using Strings”

**Introduced in R2018b**

# temporalCount

Control chart execution with the `temporalCount` operator

## Syntax

```
temporalCount(E)  
temporalCount(time_unit)
```

## Description

`temporalCount(E)` increments by 1 and returns a positive integer value for each occurrence of the base event `E` that takes place after activation of the associated state. Otherwise, the operator returns a value of 0.

In a chart with no input events, `temporalCount(tick)` or `temporalCount(wakeup)` returns the number of times that the chart has woken up since activation of the associated state.

The `temporalCount` operator resets the counter for `E` to 0 each time the associated state reactivates.

`temporalCount(time_unit)` counts and returns the number of units simulation time that have elapsed since activation of the associated state. Specify `time_unit` as seconds (`sec`), milliseconds (`msec`), or microseconds (`usec`).

The `temporalCount` operator resets the counter for `sec`, `msec` and `usec` to 0 each time the associated state reactivates.

## Examples

### State Action (during)

This action counts and returns the integer number of ticks that have elapsed since activation of the state. Then, the action assigns to the variable `y` the value of the `mm` array whose index is the value that the `temporalCount` operator returns.

```
du: y = mm[temporalCount(tick)];
```

### State Action (exit)

This action counts and returns the number of seconds of simulation time that pass between activation and deactivation of the state.

```
ex: y = temporalCount(sec);
```

## Tips

- You can use quotation marks to enclose the keywords `'tick'`, `'wakeup'`, `'sec'`, `'msec'`, and `'usec'`. For example, `temporalCount('tick')` is equivalent to `temporalCount(tick)`.

## See Also

`count` | `duration` | `elapsed`

## Topics

“Control Chart Execution by Using Temporal Logic”

**Introduced in R2008a**

# tostring

Convert numeric value to string

## Syntax

```
dest = tostring(X)
```

## Description

`dest = tostring(X)` converts numeric, Boolean, or enumerated data `X` to a string.

## Examples

### Numeric Value to String

Convert numeric value to string "1.2345".

```
dest = tostring(1.2345);
```

### Boolean Value to String

Convert Boolean value to string "true".

```
dest = tostring(1==1);
```

### Enumerated Value to String

Convert enumerated value to string "RED".

```
dest = tostring(RED);
```

## Tips

- Use in Stateflow charts that use C as the action language.

## See Also

str2double | strcpy

## Topics

“Manage Textual Information by Using Strings”

**Introduced in R2018b**



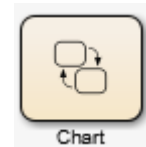
# Block Reference

---

## 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 transition occurs if 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.

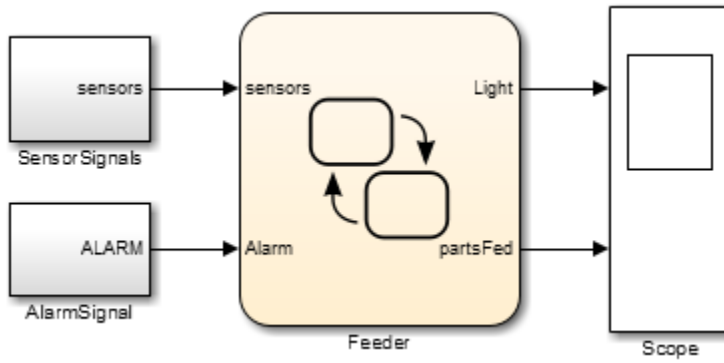
For example, you can use Stateflow charts to control a physical plant in response to events such as a temperature and pressure sensors, clocks, and user-driven events.

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.

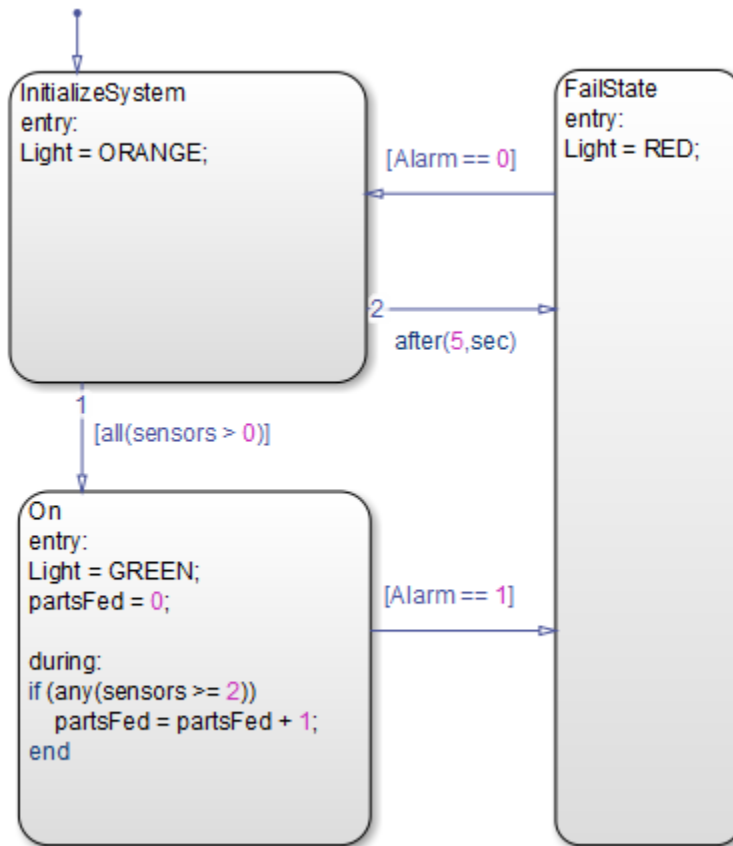
A Stateflow chart can use MATLAB or C as the action language to implement control logic.

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.





To open the chart, double-click the Feeder block in the model.



For a tutorial on this model, see “Model an Assembly Line Feeder”.

## Ports

### Input

#### Port\_1 — Input port

scalar | vector | matrix

When you create input data in the Symbols window, Stateflow creates input ports. The input data that you create has a corresponding input port that appears once you create data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point` | `enumerated` | `bus`

## Output

### Port\_1 — Output port

`scalar` | `vector` | `matrix`

When you create output data in the Symbols window, Stateflow creates output ports. The output data that you create has a corresponding output port that appears once you create data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point` | `enumerated` | `bus`

## Parameters

Parameters on the Code Generation tab require Simulink Coder™ or Embedded Coder®.

### Main

#### Show port labels — Select how to display port labels

`FromPortIcon` (default) | `FromPortBlockName` | `SignalName`

Select how to display port labels on the Chart block icon.

`none`

Do not display port labels.

`FromPortIcon`

If the corresponding port icon displays a signal name, display the signal name on the Chart block. Otherwise, display the port block name.

`FromPortBlockName`

Display the name of the corresponding port block on the Chart block.

### SignalName

If a signal name exists, display the name of the signal connected to the port on the Chart block. Otherwise, display the name of the corresponding port block.

#### **Programmatic Use**

**Parameter:** ShowPortLabels

**Type:** character vector

**Value:** 'FromPortIcon' | 'FromPortBlockName' | 'SignalName'

**Default:** 'FromPortIcon'

#### **Read/Write permissions — Select access to contents of chart**

ReadWrite (default) | ReadOnly | NoReadOrWrite

Control user access to the contents of the chart.

### ReadWrite

Enable opening and modification of chart contents.

### ReadOnly

Enable opening but not modification of the chart. If the chart resides in a block library, you can create and open links to the chart and can make and modify local copies of the chart but you cannot change the permissions or modify the contents of the original library instance.

### NoReadOrWrite

Disable opening or modification of chart. If the chart resides in a library, you can create links to the chart in a model but you cannot open, modify, change permissions, or create local copies of the chart.

#### **Programmatic Use**

**Parameter:** Permissions

**Type:** character vector

**Value:** 'ReadWrite' | 'ReadOnly' | 'NoReadOrWrite'

**Default:** 'ReadWrite'

#### **Treat as atomic unit — Control execution of a subsystem as one unit**

off (default) | on

When determining the execution order of block methods, causes Simulink to treat the chart as a unit.

off

When determining block method execution order, treat all blocks in the chart as being at the same level in the model hierarchy as the chart. This hierarchy treatment can cause the execution of methods of blocks in the chart to be interleaved with the execution of methods of blocks outside the chart.

on

When determining the execution order of block methods, treat the chart as a unit. For example, when Simulink needs to compute the output of the chart, Simulink invokes the output methods of all the blocks in the chart before invoking the output methods of other blocks at the same level as the chart block.

### Dependency

If you select this parameter, you enable the **Minimize algebraic loop occurrences**, **Sample time**, and **Function packaging** parameters. **Function packaging** requires the Simulink Coder software.

### Programmatic Use

**Parameter:** TreatAsAtomicUnit

**Type:** character vector

**Value:** 'off' | 'on'

**Default:** 'off'

### See also

- “Generate Reusable Code for Atomic Subcharts”

### Minimize algebraic loop occurrences — Control elimination of algebraic loops

off (default) | on

off

Do not try to eliminate any artificial algebraic loops that include the atomic subchart.

on

Try to eliminate any artificial algebraic loops that include the atomic subchart.

### Dependency

To enable this parameter, select the **Treat as atomic unit** parameter.

**Programmatic Use****Parameter:** MinAlgLoopOccurrences**Type:** character vector**Value:** 'off' | 'on'**Default:** 'off'**Sample time — Specify time interval****-1 (default) | [Ts 0]**

Specify whether all blocks in this chart must run at the same rate or can run at different rates.

- If the blocks in the chart can run at different rates, specify the chart sample time as inherited (-1).
- If all blocks must run at the same rate, specify the sample time corresponding to this rate as the value of the **Sample time** parameter.
- If any of the blocks in the chart specify a different sample time (other than -1 or `inf`), Simulink displays an error message when you update or simulate the model. For example, suppose all the blocks in the chart must run 5 times a second. To ensure this time, specify the sample time of the chart as 0.2. In this example, if any of the blocks in the chart specify a sample time other than 0.2, -1, or `inf`, Simulink displays an error when you update or simulate the model.

-1

Specify inherited sample time. If the blocks in the chart can run at different rates, use this sample time.

[Ts 0]

Specify periodic sample time.

**Dependency**

To enable this parameter, select the **Treat as atomic unit** parameter.

**Programmatic Use****Parameter:** SystemSampleTime**Type:** character vector**Value:** '-1' | '[Ts 0]'**Default:** '-1'

## Treat as grouped when propagating variant conditions — Control treating subsystem as unit

on (default) | off

When propagating variant conditions from Variant Source blocks or to Variant Sink blocks, causes Simulink to treat the chart as a unit.

on

Simulink treats the chart as a unit when propagating variant conditions from Variant Source blocks or to Variant Sink blocks. For example, when Simulink computes the variant condition of the chart, it propagates that condition to all the blocks in the chart.

off

Simulink treats all blocks in the chart as being at the same level in the model hierarchy as the chart itself when determining their variant condition.

### Programmatic Use

**Parameter:** TreatAsGroupedWhenPropagatingVariantConditions

**Type:** character vector

**Value:** 'on' | 'off'

**Default:** 'on'

## Code Generation

### Function packaging — Select code format

Auto (default) | Inline | Nonreusable function | Reusable function

Select the generated code format for an atomic (nonvirtual) subchart.

#### Auto

Simulink Coder chooses the optimal format for your system based on the type and number of instances of the chart that exist in the model.

#### Inline

Simulink Coder inlines the chart unconditionally.

#### Nonreusable function

Simulink Coder explicitly generates a separate function in a separate file. Charts with this setting generate functions that might have arguments depending on the

“Function interface” (Simulink) parameter setting. You can name the generated function and file using parameters “Function name” (Simulink) and “File name (no extension)” (Simulink). These functions are not reentrant.

### Reusable function

Simulink Coder generates a function with arguments that allows reuse of chart code when a model includes multiple instances of the chart.

This option generates a function with arguments that allows chart code to be reused in the generated code of a model reference hierarchy that includes multiple instances of a chart across referenced models. In this case, the chart must be in a library.

### Tips

- When you want multiple instances of a chart represented as one reusable function, you can designate each one of them as **Auto** or as **Reusable function**. It is best to use one because using both creates two reusable functions, one for each designation. The outcomes of these choices differ only when reuse is not possible. Selecting **Auto** does not allow for control of the function or file name for the chart code.
- The **Reusable function** and **Auto** options both determine whether multiple instances of a chart exist and the code can be reused. The options behave differently when it is impossible to reuse the code. In this case, **Auto** yields inlined code, or if circumstances prohibit inlining, separate functions for each chart instance.
- If you select the **Reusable function** while your generated code is under source control, set **File name options** to **Use subsystem name**, **Use function name**, or **User specified**. Otherwise, the names of your code files change whenever you modify your model, which prevents source control on your files.

### Dependency

- This parameter requires Simulink Coder.
- To enable this parameter, select **Treat as atomic unit**.
- Setting this parameter to **Nonreusable function** or **Reusable function** enables the following parameters:
  - **Function name options**
  - **File name options**
  - Memory section for initialize/terminate functions (requires Embedded Coder and an ERT-based system target file)



- Memory section for execution functions (requires Embedded Coder and an ERT-based system target file)
- Setting this parameter to `Nonreusable function` enables **Function with separate data** (requires a license for Embedded Coder and an ERT-based system target file).

**Programmatic Use****Parameter:** `RTWSystemCode`**Type:** character vector**Value:** `'Auto' | 'Inline' | 'Nonreusable function' | 'Reusable function'`**Default:** `'Auto'`

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

### PLC Code Generation

Generate Structured Text code using Simulink® PLC Coder™.

### Fixed-Point Conversion

Convert floating-point algorithms to fixed point using Fixed-Point Designer™.

## See Also

**Introduced in R2013b**

## Sequence Viewer

Display messages, events, states, transitions, and functions between blocks during simulation

**Library:** SimEvents  
Stateflow



### Description

The Sequence Viewer block displays messages, events, states, transitions, and functions between certain blocks during simulation. The blocks that you can display are called lifeline blocks and include:

- Subsystems
- Referenced models
- Blocks that contain messages, such as Stateflow charts
- Blocks that call functions or generate events, such as Function Caller, Function-Call Generator, and MATLAB Function blocks
- Blocks that contain functions, such as Function-Call Subsystem and Simulink Function blocks

To see states, transitions, and events for lifeline blocks in a referenced model, you must have a Sequence Viewer block in the referenced model. Without a Sequence Viewer block in the referenced model, you can see only messages and functions for lifeline blocks in the referenced model.

### Parameters

#### **Time Precision for Variable Step — Adjust time increment precision**

3 (default)

When using a variable step solver, change this parameter to adjust the time precision for the sequence viewer.

**History — Maximum number of events to keep in viewer**

5000 (default)

**See Also**

“Use the Sequence Viewer Block to Visualize Messages, Events, and Entities” (SimEvents)

**Introduced in R2015b**

# State Transition Table

Represent modal logic in tabular format

**Library:** Stateflow



## Description

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

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 automatically generated content as you edit the table.

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

## Ports

### Input

**Port\_1 – Input port**

scalar | vector | matrix

When you create input data in the Symbols window, Stateflow creates input ports. The input data that you create has a corresponding input port that appears once you create data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point` | `enumerated` | `bus`

## Output

### Port\_1 — Output port

`scalar` | `vector` | `matrix`

When you create output data in the Symbols window, Stateflow creates output ports. The output data that you create has a corresponding output port that appears once you create data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point` | `enumerated` | `bus`

## Parameters

Parameters on the Code Generation tab require Simulink Coder or Embedded Coder.

### Main

#### Show port labels — Select how to display port labels

`FromPortIcon` (default) | `FromPortBlockName` | `SignalName`

Select how to display port labels on the Chart block icon.

`none`

Do not display port labels.

`FromPortIcon`

If the corresponding port icon displays a signal name, display the signal name on the Chart block. Otherwise, display the port block name.

`FromPortBlockName`

Display the name of the corresponding port block on the Chart block.

### SignalName

If a signal name exists, display the name of the signal connected to the port on the Chart block. Otherwise, display the name of the corresponding port block.

#### **Programmatic Use**

**Parameter:** ShowPortLabels

**Type:** character vector

**Value:** 'FromPortIcon' | 'FromPortBlockName' | 'SignalName'

**Default:** 'FromPortIcon'

#### **Read/Write permissions — Select access to contents of chart**

ReadWrite (default) | ReadOnly | NoReadOrWrite

Control user access to the contents of the chart.

### ReadWrite

Enable opening and modification of chart contents.

### ReadOnly

Enable opening but not modification of the chart. If the chart resides in a block library, you can create and open links to the chart and can make and modify local copies of the chart but you cannot change the permissions or modify the contents of the original library instance.

### NoReadOrWrite

Disable opening or modification of chart. If the chart resides in a library, you can create links to the chart in a model but you cannot open, modify, change permissions, or create local copies of the chart.

#### **Programmatic Use**

**Parameter:** Permissions

**Type:** character vector

**Value:** 'ReadWrite' | 'ReadOnly' | 'NoReadOrWrite'

**Default:** 'ReadWrite'

#### **Treat as atomic unit — Control execution of a subsystem as one unit**

off (default) | on

When determining the execution order of block methods, causes Simulink to treat the chart as a unit.

off

When determining block method execution order, treat all blocks in the chart as being at the same level in the model hierarchy as the chart. This hierarchy treatment can cause the execution of methods of blocks in the chart to be interleaved with the execution of methods of blocks outside the chart.

on

When determining the execution order of block methods, treat the chart as a unit. For example, when Simulink needs to compute the output of the chart, Simulink invokes the output methods of all the blocks in the chart before invoking the output methods of other blocks at the same level as the chart block.

### Dependency

If you select this parameter, you enable the **Minimize algebraic loop occurrences**, **Sample time**, and **Function packaging** parameters. **Function packaging** requires the Simulink Coder software.

### Programmatic Use

**Parameter:** TreatAsAtomicUnit

**Type:** character vector

**Value:** 'off' | 'on'

**Default:** 'off'

### See also

- “Generate Reusable Code for Atomic Subcharts”

### Minimize algebraic loop occurrences — Control elimination of algebraic loops

off (default) | on

off

Do not try to eliminate any artificial algebraic loops that include the atomic subchart.

on

Try to eliminate any artificial algebraic loops that include the atomic subchart.

### Dependency

To enable this parameter, select the **Treat as atomic unit** parameter.

**Programmatic Use****Parameter:** MinAlgLoopOccurrences**Type:** character vector**Value:** 'off' | 'on'**Default:** 'off'**Sample time — Specify time interval****-1** (default) | [Ts 0]

Specify whether all blocks in this chart must run at the same rate or can run at different rates.

- If the blocks in the chart can run at different rates, specify the chart sample time as inherited (-1).
- If all blocks must run at the same rate, specify the sample time corresponding to this rate as the value of the **Sample time** parameter.
- If any of the blocks in the chart specify a different sample time (other than -1 or `inf`), Simulink displays an error message when you update or simulate the model. For example, suppose all the blocks in the chart must run 5 times a second. To ensure this time, specify the sample time of the chart as 0.2. In this example, if any of the blocks in the chart specify a sample time other than 0.2, -1, or `inf`, Simulink displays an error when you update or simulate the model.

**-1**

Specify inherited sample time. If the blocks in the chart can run at different rates, use this sample time.

**[Ts 0]**

Specify periodic sample time.

**Dependency**

To enable this parameter, select the **Treat as atomic unit** parameter.

**Programmatic Use****Parameter:** SystemSampleTime**Type:** character vector**Value:** '-1' | '[Ts 0]'**Default:** '-1'



## Treat as grouped when propagating variant conditions — Control treating subsystem as unit

on (default) | off

When propagating variant conditions from Variant Source blocks or to Variant Sink blocks, causes Simulink to treat the chart as a unit.

on

Simulink treats the chart as a unit when propagating variant conditions from Variant Source blocks or to Variant Sink blocks. For example, when Simulink computes the variant condition of the chart, it propagates that condition to all the blocks in the chart.

off

Simulink treats all blocks in the chart as being at the same level in the model hierarchy as the chart itself when determining their variant condition.

### Programmatic Use

**Parameter:** TreatAsGroupedWhenPropagatingVariantConditions

**Type:** character vector

**Value:** 'on' | 'off'

**Default:** 'on'

## Code Generation

### Function packaging — Select code format

Auto (default) | Inline | Nonreusable function | Reusable function

Select the generated code format for an atomic (nonvirtual) subchart.

Auto

Simulink Coder chooses the optimal format for your system based on the type and number of instances of the chart that exist in the model.

Inline

Simulink Coder inlines the chart unconditionally.

Nonreusable function

Simulink Coder explicitly generates a separate function in a separate file. Charts with this setting generate functions that might have arguments depending on the

“Function interface” (Simulink) parameter setting. You can name the generated function and file using parameters “Function name” (Simulink) and “File name (no extension)” (Simulink). These functions are not reentrant.

#### Reusable function

Simulink Coder generates a function with arguments that allows reuse of chart code when a model includes multiple instances of the chart.

This option generates a function with arguments that allows chart code to be reused in the generated code of a model reference hierarchy that includes multiple instances of a chart across referenced models. In this case, the chart must be in a library.

#### Tips

- When you want multiple instances of a chart represented as one reusable function, you can designate each one of them as **Auto** or as **Reusable function**. It is best to use one because using both creates two reusable functions, one for each designation. The outcomes of these choices differ only when reuse is not possible. Selecting **Auto** does not allow for control of the function or file name for the chart code.
- The **Reusable function** and **Auto** options both try to determine if multiple instances of a chart exist and if the code can be reused. The difference between the options' behavior is that when reuse is not possible. In this case, **Auto** yields inlined code, or if circumstances prohibit inlining, separate functions for each chart instance.
- If you select the **Reusable function** while your generated code is under source control, set **File name options** to **Use subsystem name**, **Use function name**, or **User specified**. Otherwise, the names of your code files change whenever you modify your model, which prevents source control on your files.

#### Dependency

- This parameter requires Simulink Coder.
- To enable this parameter, select **Treat as atomic unit**.
- Setting this parameter to **Nonreusable function** or **Reusable function** enables the following parameters:
  - **Function name options**
  - **File name options**
  - Memory section for initialize/terminate functions (requires Embedded Coder and an ERT-based system target file)

- Memory section for execution functions (requires Embedded Coder and an ERT-based system target file)
- Setting this parameter to `Nonreusable function` enables **Function with separate data** (requires a license for Embedded Coder and an ERT-based system target file).

**Programmatic Use****Parameter:** `RTWSystemCode`**Type:** character vector**Value:** `'Auto' | 'Inline' | 'Nonreusable function' | 'Reusable function'`**Default:** `'Auto'`

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

### PLC Code Generation

Generate Structured Text code using Simulink® PLC Coder™.

### Fixed-Point Conversion

Convert floating-point algorithms to fixed point using Fixed-Point Designer™.

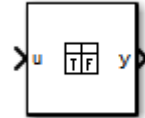
## See Also

**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. When you want to use truth table logic directly in a Simulink model, use this block. This block requires Stateflow.

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 than creating a truth table within a Stateflow chart, especially if your model requires only a single truth table.
- You can define truth table inputs and outputs with 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.

If you double-click the Truth Table block, the Truth Table Editor opens to display its conditions, actions, and decisions.

Using the Truth Table Editor, you can:

- Enter and edit conditions, actions, and decisions.
- Add or modify Stateflow data and ports by 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 Operations”.

## Ports

### Input

#### **u — Input port**

scalar | vector | matrix

When you create input data in the Symbols window, Stateflow creates input ports. The input data that you create has a corresponding input port that appears once you create data.

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | Boolean | fixed point | enumerated | bus

### Output

#### **y — Output port**

scalar | vector | matrix

When you create output data in the Symbols window, Stateflow creates output ports. The output data that you create has a corresponding output port that appears once you create data.

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | Boolean | fixed point | enumerated | bus

## Parameters

Parameters on the Code Generation tab require Simulink Coder or Embedded Coder.

### Main

#### **Show port labels — Select how to display port labels**

FromPortIcon (default) | FromPortBlockName | SignalName

Select how to display port labels on the Chart block icon.

none

Do not display port labels.

FromPortIcon

If the corresponding port icon displays a signal name, display the signal name on the Chart block. Otherwise, display the port block name.

FromPortBlockName

Display the name of the corresponding port block on the Chart block.

SignalName

If a signal name exists, display the name of the signal connected to the port on the Chart block. Otherwise, display the name of the corresponding port block.

**Programmatic Use**

**Parameter:** ShowPortLabels

**Type:** character vector

**Value:** 'FromPortIcon' | 'FromPortBlockName' | 'SignalName'

**Default:** 'FromPortIcon'

**Read/Write permissions — Select access to contents of chart**

ReadWrite (default) | ReadOnly | NoReadOrWrite

Control user access to the contents of the chart.

ReadWrite

Enable opening and modification of chart contents.

ReadOnly

Enable opening but not modification of the chart. If the chart resides in a block library, you can create and open links to the chart and can make and modify local copies of the chart but you cannot change the permissions or modify the contents of the original library instance.

NoReadOrWrite

Disable opening or modification of chart. If the chart resides in a library, you can create links to the chart in a model but you cannot open, modify, change permissions, or create local copies of the chart.

**Programmatic Use**

**Parameter:** Permissions

**Type:** character vector

**Value:** 'ReadWrite' | 'ReadOnly' | 'NoReadOrWrite'

**Default:** 'ReadWrite'

### **Treat as atomic unit – Control execution of a subsystem as one unit**

off (default) | on

When determining the execution order of block methods, causes Simulink to treat the chart as a unit.

off

When determining block method execution order, treat all blocks in the chart as being at the same level in the model hierarchy as the chart. This hierarchy treatment can cause the execution of methods of blocks in the chart to be interleaved with the execution of methods of blocks outside the chart.

on

When determining the execution order of block methods, treat the chart as a unit. For example, when Simulink needs to compute the output of the chart, Simulink invokes the output methods of all the blocks in the chart before invoking the output methods of other blocks at the same level as the chart block.

### **Dependency**

If you select this parameter, you enable the **Minimize algebraic loop occurrences**, **Sample time**, and **Function packaging** parameters. **Function packaging** requires the Simulink Coder software.

### **Programmatic Use**

**Parameter:** TreatAsAtomicUnit

**Type:** character vector

**Value:** 'off' | 'on'

**Default:** 'off'

### **See also**

- “Generate Reusable Code for Atomic Subcharts”

### **Minimize algebraic loop occurrences – Control elimination of algebraic loops**

off (default) | on

off

Do not try to eliminate any artificial algebraic loops that include the atomic subchart.

on

Try to eliminate any artificial algebraic loops that include the atomic subchart.

### Dependency

To enable this parameter, select the **Treat as atomic unit** parameter.

### Programmatic Use

**Parameter:** MinAlgLoopOccurrences

**Type:** character vector

**Value:** 'off' | 'on'

**Default:** 'off'

### Sample time — Specify time interval

-1 (default) | [Ts 0]

Specify whether all blocks in this chart must run at the same rate or can run at different rates.

- If the blocks in the chart can run at different rates, specify the chart sample time as inherited (-1).
- If all blocks must run at the same rate, specify the sample time corresponding to this rate as the value of the **Sample time** parameter.
- If any of the blocks in the chart specify a different sample time (other than -1 or `inf`), Simulink displays an error message when you update or simulate the model. For example, suppose all the blocks in the chart must run 5 times a second. To ensure this time, specify the sample time of the chart as 0.2. In this example, if any of the blocks in the chart specify a sample time other than 0.2, -1, or `inf`, Simulink displays an error when you update or simulate the model.

-1

Specify inherited sample time. If the blocks in the chart can run at different rates, use this sample time.

[Ts 0]

Specify periodic sample time.



## Dependency

To enable this parameter, select the **Treat as atomic unit** parameter.

### Programmatic Use

**Parameter:** SystemSampleTime

**Type:** character vector

**Value:** '-1' | '[Ts 0]'

**Default:** '-1'

### Treat as grouped when propagating variant conditions — Control treating subsystem as unit

on (default) | off

When propagating variant conditions from Variant Source blocks or to Variant Sink blocks, causes Simulink to treat the chart as a unit.

on

Simulink treats the chart as a unit when propagating variant conditions from Variant Source blocks or to Variant Sink blocks. For example, when Simulink computes the variant condition of the chart, it propagates that condition to all the blocks in the chart.

off

Simulink treats all blocks in the chart as being at the same level in the model hierarchy as the chart itself when determining their variant condition.

### Programmatic Use

**Parameter:** TreatAsGroupedWhenPropagatingVariantConditions

**Type:** character vector

**Value:** 'on' | 'off'

**Default:** 'on'

## Code Generation

### Function packaging — Select code format

Auto (default) | Inline | Nonreusable function | Reusable function

Select the generated code format for an atomic (nonvirtual) subchart.

### Auto

Simulink Coder chooses the optimal format for your system based on the type and number of instances of the chart that exist in the model.

### Inline

Simulink Coder inlines the chart unconditionally.

### Nonreusable function

Simulink Coder explicitly generates a separate function in a separate file. Charts with this setting generate functions that might have arguments depending on the “Function interface” (Simulink) parameter setting. You can name the generated function and file using parameters “Function name” (Simulink) and “File name (no extension)” (Simulink). These functions are not reentrant.

### Reusable function

Simulink Coder generates a function with arguments that allows reuse of chart code when a model includes multiple instances of the chart.

This option generates a function with arguments that allows chart code to be reused in the generated code of a model reference hierarchy that includes multiple instances of a chart across referenced models. In this case, the chart must be in a library.

### Tips

- When you want multiple instances of a chart represented as one reusable function, you can designate each one of them as **Auto** or as **Reusable function**. It is best to use one because using both creates two reusable functions, one for each designation. The outcomes of these choices differ only when reuse is not possible. Selecting **Auto** does not allow for control of the function or file name for the chart code.
- The **Reusable function** and **Auto** options both try to determine if multiple instances of a chart exist and if the code can be reused. The difference between the options' behavior is that when reuse is not possible. In this case, **Auto** yields inlined code, or if circumstances prohibit inlining, separate functions for each chart instance.
- If you select the **Reusable function** while your generated code is under source control, set **File name options** to **Use subsystem name**, **Use function name**, or **User specified**. Otherwise, the names of your code files change whenever you modify your model, which prevents source control on your files.

### Dependency

- This parameter requires Simulink Coder.

- To enable this parameter, select **Treat as atomic unit**.
- Setting this parameter to `Nonreusable function` or `Reusable function` enables the following parameters:
  - **Function name options**
  - **File name options**
  - Memory section for initialize/terminate functions (requires Embedded Coder and an ERT-based system target file)
  - Memory section for execution functions (requires Embedded Coder and an ERT-based system target file)
- Setting this parameter to `Nonreusable function` enables **Function with separate data** (requires a license for Embedded Coder and an ERT-based system target file).

**Programmatic Use****Parameter:** RTWSystemCode**Type:** character vector**Value:** 'Auto' | 'Inline' | 'Nonreusable function' | 'Reusable function'**Default:** 'Auto'

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

### PLC Code Generation

Generate Structured Text code using Simulink® PLC Coder™.

### Fixed-Point Conversion

Convert floating-point algorithms to fixed point using Fixed-Point Designer™.

## **See Also**

**Introduced before R2006a**