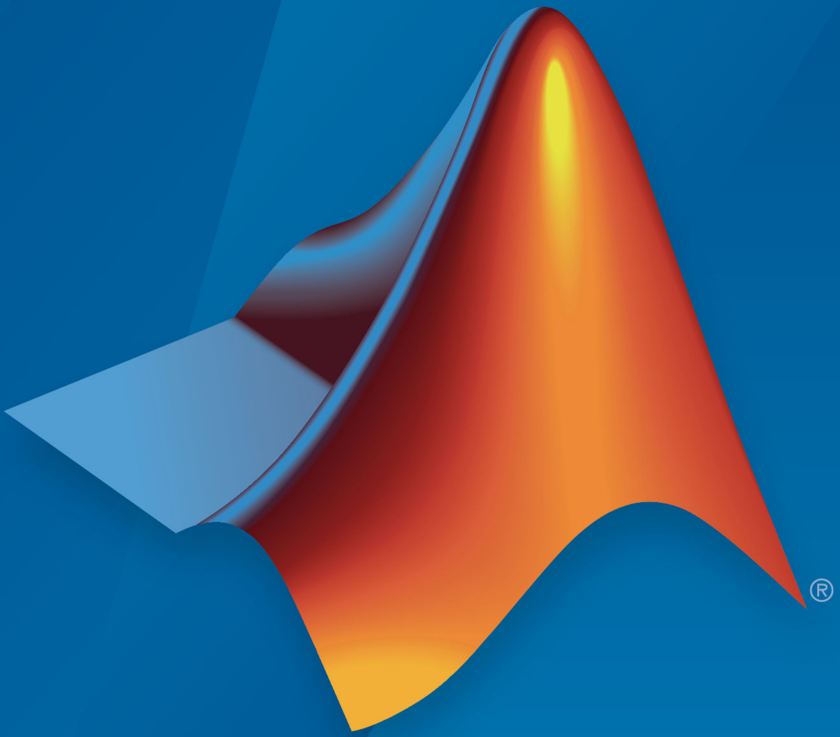
Stateflow®

Reference



MATLAB® & SIMULINK®

R2018b

MathWorks®

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)

Contents

Functions — Alphabetical List

sfclipboard

Stateflow clipboard object

Syntax

```
object = sfclipboard
```

Description

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

Examples

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

```
sf_pool;
% Get handle to the root object
rt = sfroot;
% Get handle to 'init' function in Init chart
f1 = rt.find('-isa','Stateflow.EMFunction','Name','init');
% Get handle to Pool chart
chP = rt.find('-isa','Stateflow.Chart','Name','Pool');
% Get handle to the clipboard object
cb = sfclipboard;
% Copy 'init' function to the clipboard
cb.copy(f1);
% Paste 'init' function to the Pool chart
cb.pasteTo(chP);
% Get handle to newly pasted function
f2 = chP.find('-isa','Stateflow.EMFunction','Name','init');
% Reset position of new function in the Pool chart
f2.Position = [90 180 90 60];
```

See Also

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

Topics

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

Introduced before R2006a

sfclose

Close chart

Syntax

```
sfclose
sfclose('chart_name')
sfclose('all')
```

Description

`sfclose` closes the current chart.

`sfclose('`*`chart_name`*`')` closes the chart called `'`*`chart_name`*`'`.

`sfclose('all')` closes all open or minimized charts. `'all'` is a literal character vector.

See Also

`sfnew` | `sfopen` | `stateflow`

Introduced in R2006a

sfdebugger

Open Stateflow Debugger

Syntax

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

Description

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

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

See Also

`sfexplr` | `sfhelp` | `sflib`

Topics

"Debug Run-Time Errors in a Chart"

Introduced in R2006a

sfexplr

Open Model Explorer

Syntax

```
sfexplr
```

Description

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

See Also

`sfdebugger` | `sfhelp` | `sflib`

Topics

“Use the Model Explorer with Stateflow Objects”

Introduced in R2006a

sfgco

Recently selected objects in chart

Syntax

`object = sfgco`

Description

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

Output Arguments

`object`

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

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

Examples

Zoom in on a state after clicking it:

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

See Also

`sfnew` | `sfroot` | `stateflow`

Topics

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

Introduced before R2006a

sfhelp

Open Stateflow online help

Syntax

`sfhelp`

Description

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

See Also

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

Introduced before R2006a

sflib

Open Stateflow library window

Syntax

```
sflib
```

Description

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

See Also

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

Introduced in R2006a

sfnew

Create model containing empty Stateflow block

Syntax

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

Description

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

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

`sfnew('model_name')` creates a model called *`model_name`* with an empty chart with the default action language.

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

Input Arguments

`chart_type`

Empty block to add to an empty model:

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

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

`model_name`

Name of the model.

Examples

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

```
sfnew()
```

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

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

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

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

See Also

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

Topics

“Model Event-Driven System”
“Create Mealy and Moore Charts”
“Define a Truth Table Function”
“Syntax for States and Transitions”

Introduced before R2006a

sfopen

Open existing model

Syntax

`sfopen`

Description

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

See Also

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

Introduced in R2006a

sfprint

Print graphical view of charts

Syntax

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

Description

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

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

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

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

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

Examples

Print open chart

```
sfprint
```

Prints current chart to the default printer.

Print all charts specified in path

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

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

Print chart specified in path to a JPG file format.

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

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

Print chart in TIFF format to the clipboard.

```
sfprint(gcs,'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:

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

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

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

Data Types: `char`

`outputOption` — Name of the printer or output file
`'file'` (default) | character vector | `'clipboard'` | `'promptForFile'` | `'printer'`

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

`'file'`	Send output to a default file with the name *`chart_name.file_extension`*. The file name is the name of the chart, with an extension that matches the output format.
character vector	Specify the name of the output file with a character vector.
`'clipboard'`	Copy output to the clipboard
`'promptForFile'`	Prompts the user interactively for path and file name.
`'printer'`	Send output to the default printer (use only with `'ps'`, or `'eps'` formats)

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

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

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

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

Data Types: `char`

`wholeChart` — View of charts to print
1 (default) | 0

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

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

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

See Also

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

Introduced before R2006a

sfsave

Save chart in current folder

Syntax

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

Description

`sfsave` saves the chart in the current model.

`sfsave('model_name')` saves the chart in the model called `'model_name'`.

`sfsave('model_name','new_model_name')` saves the chart in `'model_name'` to `'new_model_name'`.

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

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

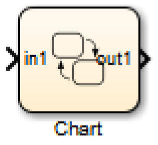
Examples

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

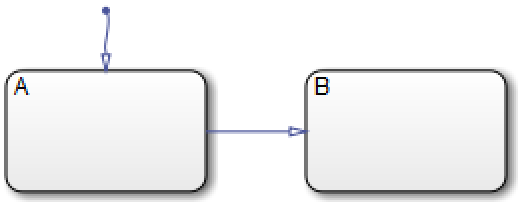
```
bdclose('all');

% Create an empty chart in a new model
sfnew;

% Get root object
rt = sfroot;
```

```
% Get model
m = rt.find('-isa','Simulink.BlockDiagram');

% Get chart
chart1 = m.find('-isa','Stateflow.Chart');

% Create two states, A and B, in the chart
sA = Stateflow.State(chart1);
sA.Name = 'A';
sA.Position = [50 50 100 60];
sB = Stateflow.State(chart1);
sB.Name = 'B';
sB.Position = [200 50 100 60];

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

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

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

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

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

Here is the resulting model:



Here is the resulting chart:



See Also

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

Topics

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

Introduced before R2006a

stateflow

Create empty chart

Syntax

```
stateflow
```

Description

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

See Also

`sflib` | `sfnew`

Introduced before R2006a

Operators — Alphabetical List

after

Control chart execution with the `after` operator

Syntax

```
after(n,E)
after(n,sec)
```

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,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`.

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

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)
```

See Also

Topics
"Control Chart Execution Using Temporal Logic"

Introduced in R2014b

at

Control chart execution with the `at` operator

Syntax

`at(n,E)`

Description

`at(n,E)` returns true only at the `n`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`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)]
```

See Also

Topics

"Control Chart Execution Using Temporal Logic"

Introduced in R2014b

before

Control chart execution with the `before` operator

Syntax

```
before(n,E)
before(n,sec)
```

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

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)]
```

See Also

Topics

"Control Chart Execution Using Temporal Logic"

Introduced in R2014b

duration

Control chart execution with the `duration` operator

Syntax

```
duration(sec)
```

Description

`duration(sec)` 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

Topics

"Control Chart Execution Using Temporal 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);
```

See Also

Topics

"Control Chart Execution Using Temporal Logic"

Introduced in R2017a

every

Control chart execution with the `every` operator

Syntax

`every(n,E)`

Description

`every(n,E)` returns true at every `n`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 `n` times since activation of the associated state.

The `every` operator resets the counter for `E` to `0` each time the associated state reactivates. Therefore, this operator is useful only in state actions and not in transitions.

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');
```

See Also

Topics

“Control Chart Execution Using Temporal Logic”

Introduced in R2014b

temporalCount

Control chart execution with the `temporalCount` operator

Syntax

```
temporalCount(E)
temporalCount(sec)
```

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.

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

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

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);
```

See Also

Topics

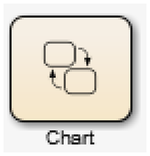
"Control Chart Execution Using Temporal Logic"

Introduced in R2014b

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.

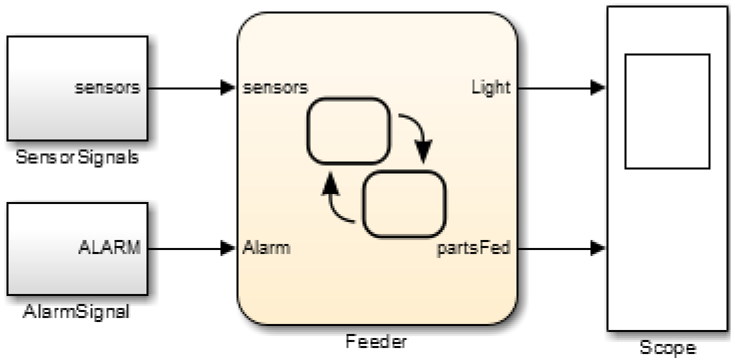
A Stateflow chart is a graphical representation of a finite state machine. *States* and *transitions* form the basic elements of the system. You can also represent stateless flow charts.

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.

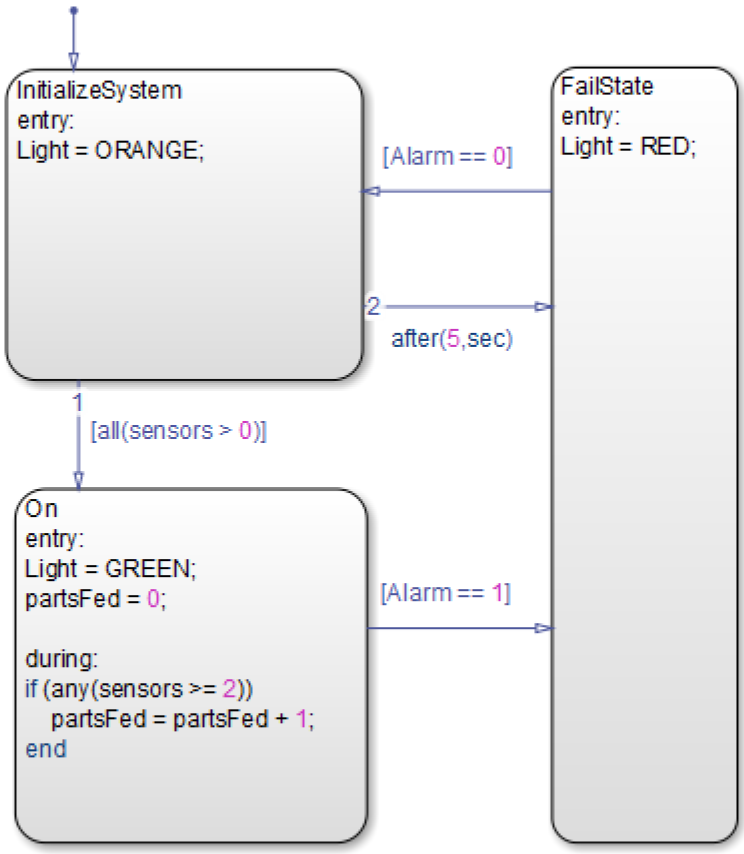
You can also use a state machine to represent the automatic transmission of a car. The transmission has these operating states: park, reverse, neutral, drive, and low. As the driver shifts from one position to another, the system makes a transition from one state to another, for example, from park to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

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: 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)

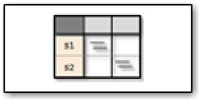
See Also

“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 autogenerated 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

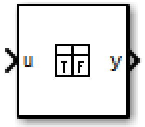
Convert floating-point algorithms to fixed point using Fixed-Point Designer™.

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