

Stateflow[®]

Reference



MATLAB[®]&SIMULINK[®]

R2020b



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Stateflow® Reference

© COPYRIGHT 2006–2020 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 for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

March 2006	Online only	New for Version 6.4 (Release 2006a)
September 2006	Online only	Revised for Version 6.5 (Release R2006b)
September 2007	Online only	Rereleased for Version 7.0 (Release 2007b)
March 2008	Online only	Revised for Version 7.1 (Release 2008a)
October 2008	Online only	Revised for Version 7.2 (Release 2008b)
March 2009	Online only	Rereleased for Version 7.3 (Release 2009a)
September 2009	Online only	Revised for Version 7.4 (Release 2009b)
March 2010	Online only	Rereleased for Version 7.5 (Release 2010a)
September 2010	Online only	Rereleased for Version 7.6 (Release 2010b)
April 2011	Online only	Rereleased for Version 7.7 (Release 2011a)
September 2011	Online only	Rereleased for Version 7.8 (Release 2011b)
March 2012	Online only	Revised for Version 7.9 (Release 2012a)
September 2012	Online only	Revised for Version 8.0 (Release 2012b)
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)
September 2019	Online only	Revised for Version 10.1 (Release 2019b)
March 2020	Online only	Revised for Version 10.2 (Release 2020a)
September 2020	Online only	Revised for Version 10.3 (Release 2020b)

1	<hr/>	Blocks
2	<hr/>	Functions
3	<hr/>	Operators
4	<hr/>	Objects
5	<hr/>	Tools

Blocks

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.

Ports

Input

Port_1 – Input port

scalar | vector | matrix

When you create input data in the Symbols pane, 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 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | fixed point | enumerated | bus

Output

Port_1 – Output port

scalar | vector | matrix

When you create output data in the Symbols pane, 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 | int64 | uint8 | uint16 | uint32 | uint64 | 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 Code from 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” parameter setting. You can name the generated function and file using parameters “Function name” and “File name (no extension)”. 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™.

HDL Coder™ provides additional configuration options that affect HDL implementation and synthesized logic.

HDL Architecture

This block has a single, default HDL architecture.

Active State Output

To generate an output port in the HDL code that shows the active state, in the Properties window of the chart, select **Create output for monitoring**. The output is an enumerated data type. See “Simplify Stateflow Charts by Incorporating Active State Output”.

Registered Output

To insert an output register that delays the chart output by a simulation cycle, use the `OutputPipeline` (HDL Coder) block property.

HDL Block Properties

ConstMultiplierOptimization	Canonical signed digit (CSD) or factored CSD optimization. The default is none. See also “ConstMultiplierOptimization” (HDL Coder).
ConstrainedOutputPipeline	Number of registers to place at the outputs by moving existing delays within your design. Distributed pipelining does not redistribute these registers. The default is 0. For more details, see “ConstrainedOutputPipeline” (HDL Coder).
DistributedPipelining	Pipeline register distribution, or register retiming. The default is off. See also “DistributedPipelining” (HDL Coder).
InputPipeline	Number of input pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “InputPipeline” (HDL Coder).
InstantiateFunctions	Generate a VHDL® entity or Verilog® module for each function. The default is off. See also “InstantiateFunctions” (HDL Coder).

LoopOptimization	Unroll, stream, or do not optimize loops. The default is <code>none</code> . See also “LoopOptimization” (HDL Coder).
MapPersistentVarsToRAM	Map persistent arrays to RAM. The default is <code>off</code> . See also “MapPersistentVarsToRAM” (HDL Coder).
OutputPipeline	Number of output pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is <code>0</code> . For more details, see “OutputPipeline” (HDL Coder).
ResetType	Suppress reset logic generation. The default is <code>default</code> , which generates reset logic. See also “ResetType” (HDL Coder).
SharingFactor	Number of functionally equivalent resources to map to a single shared resource. The default is <code>0</code> . See also “Resource Sharing” (HDL Coder).
VariablesToPipeline	<p>Warning <code>VariablesToPipeline</code> is not recommended. Use <code>coder.hdl.pipeline</code> instead.</p> <p>Insert a pipeline register at the output of the specified MATLAB® variable or variables. Specify the list of variables as a character vector, with spaces separating the variables.</p>

Complex Data Support

This block supports code generation for complex signals.

Restrictions

To learn about restrictions of using charts, see “Introduction to Stateflow HDL Code Generation” (HDL Coder).

PLC Code Generation

Generate Structured Text code using Simulink® PLC Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

State Transition Table | Truth Table

Topics

“Construct and Run a Stateflow Chart”

“The Stateflow Chart”

“Specify Properties for Stateflow Charts”

Introduced before R2006a

Sequence Viewer

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

Library: Simulink / Messages & Events
 Simulink Test
 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 — Digits for time increment precision

3 (default) | scalar

Number of digits for time increment precision. When using a variable step solver, change this parameter to adjust the time precision for the sequence viewer. By default the block supports 3 digits of precision.

Suppose the block displays two events that occur at times 0.1215 and 0.1219. Displaying these two events precisely requires 4 digits of precision. If the precision is 3, then the block displays two events at time 0.121.

Programmatic Use

Block Parameter: VariableStepTimePrecision

Type: character vector

Values: '3' | scalar

Default: '3'

History — Maximum number of previous events to display

5000 (default) | scalar

Total number of events before the last event to display.

For example, if **History** is 5 and there are 10 events in your simulation, then the block displays 6 events, including the last event and the five events prior the last event. Earlier events are not displayed. The time ruler is greyed to indicate the time between the beginning of the simulation and the time of the first displayed event.

Each send, receive, drop, or function call event is counted as one event, even if they occur at the same simulation time.

Programmatic Use

Block Parameter: History

Type: character vector

Values: '1000' | scalar

Default: '1000'

Block Characteristics

Data Types	Boolean bus double enumerated fixed point integer single
Direct Feedthrough	no
Multidimensional Signals	yes
Variable-Size Signals	no
Zero-Crossing Detection	no

Extended Capabilities

C/C++ Code Generation

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

This block can be used for visualizing message transitions during simulation, but is not included in the generated code.

HDL Code Generation

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

This block displays messages during simulation when used in subsystems that generate HDL code, but it is not included in the hardware implementation.

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 pane, 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 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | fixed point | enumerated | bus

Output

Port_1 — Output port

scalar | vector | matrix

When you create output data in the Symbols pane, 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` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `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 Code from 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” parameter setting. You can name the generated function and file using parameters “Function name” and “File name (no extension)”. 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™.

HDL Coder provides additional configuration options that affect HDL implementation and synthesized logic.

Tunable Parameters

You can use a tunable parameter in a State Transition Table intended for HDL code generation. For details, see “Generate DUT Ports for Tunable Parameters” (HDL Coder).

HDL Architecture

This block has a single, default HDL architecture.

Active State Output

To generate an output port in the HDL code that shows the active state, in the Properties window of the chart, select **Create output for monitoring**. The output is an enumerated data type. See “Simplify Stateflow Charts by Incorporating Active State Output”.

HDL Block Properties

ConstMultiplierOptimization	Canonical signed digit (CSD) or factored CSD optimization. The default is none. See also “ConstMultiplierOptimization” (HDL Coder).
ConstrainedOutputPipeline	Number of registers to place at the outputs by moving existing delays within your design. Distributed pipelining does not redistribute these registers. The default is 0. For more details, see “ConstrainedOutputPipeline” (HDL Coder).
DistributedPipelining	Pipeline register distribution, or register retiming. The default is off. See also “DistributedPipelining” (HDL Coder).
InputPipeline	Number of input pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “InputPipeline” (HDL Coder).

InstantiateFunctions	Generate a VHDL entity or Verilog module for each function. The default is <code>off</code> . See also “InstantiateFunctions” (HDL Coder).
LoopOptimization	Unroll, stream, or do not optimize loops. The default is <code>none</code> . See also “LoopOptimization” (HDL Coder).
MapPersistentVarsToRAM	Map persistent arrays to RAM. The default is <code>off</code> . See also “MapPersistentVarsToRAM” (HDL Coder).
OutputPipeline	Number of output pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is <code>0</code> . For more details, see “OutputPipeline” (HDL Coder).
ResetType	Suppress reset logic generation. The default is <code>default</code> , which generates reset logic. See also “ResetType” (HDL Coder).
SharingFactor	Number of functionally equivalent resources to map to a single shared resource. The default is <code>0</code> . See also “Resource Sharing” (HDL Coder).
VariablesToPipeline	<p>Warning <code>VariablesToPipeline</code> is not recommended. Use <code>coder.hdl.pipeline</code> instead.</p> <hr/> <p>Insert a pipeline register at the output of the specified MATLAB variable or variables. Specify the list of variables as a character vector, with spaces separating the variables.</p>

PLC Code Generation

Generate Structured Text code using Simulink® PLC Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Chart | Truth Table

Topics

“State Transition Tables in Stateflow”

“State Transition Table Operations”

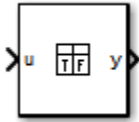
“Specify Properties for Stateflow Charts”

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 “Use Truth Tables to Model Combinatorial Logic”.

Ports

Input

u — Input port

scalar | vector | matrix

When you create input data in the Symbols pane, 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` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point` | `enumerated` | `bus`

Output

y – Output port

`scalar` | `vector` | `matrix`

When you create output data in the Symbols pane, 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 Code from 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” parameter setting. You can name the generated function and file using parameters “Function name” and “File name (no extension)”. 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™.

HDL Coder provides additional configuration options that affect HDL implementation and synthesized logic.

Tunable Parameters

You can use a tunable parameter in a Truth Table intended for HDL code generation. For details, see “Generate DUT Ports for Tunable Parameters” (HDL Coder).

HDL Architecture

This block has a single, default HDL architecture.

HDL Block Properties

ConstMultiplierOptimization	Canonical signed digit (CSD) or factored CSD optimization. The default is none. See also “ConstMultiplierOptimization” (HDL Coder).
------------------------------------	---

ConstrainedOutputPipeline	Number of registers to place at the outputs by moving existing delays within your design. Distributed pipelining does not redistribute these registers. The default is 0. For more details, see “ConstrainedOutputPipeline” (HDL Coder).
DistributedPipelining	Pipeline register distribution, or register retiming. The default is off. See also “DistributedPipelining” (HDL Coder).
InputPipeline	Number of input pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “InputPipeline” (HDL Coder).
InstantiateFunctions	Generate a VHDL entity or Verilog module for each function. The default is off. See also “InstantiateFunctions” (HDL Coder).
LoopOptimization	Unroll, stream, or do not optimize loops. The default is none. See also “LoopOptimization” (HDL Coder).
MapPersistentVarsToRAM	Map persistent arrays to RAM. The default is off. See also “MapPersistentVarsToRAM” (HDL Coder).
OutputPipeline	Number of output pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “OutputPipeline” (HDL Coder).
ResetType	Suppress reset logic generation. The default is default, which generates reset logic. See also “ResetType” (HDL Coder).
SharingFactor	Number of functionally equivalent resources to map to a single shared resource. The default is 0. See also “Resource Sharing” (HDL Coder).
VariablesToPipeline	<p>Warning VariablesToPipeline is not recommended. Use coder.hdl.pipeline instead.</p> <p>Insert a pipeline register at the output of the specified MATLAB variable or variables. Specify the list of variables as a character vector, with spaces separating the variables.</p>

PLC Code Generation

Generate Structured Text code using Simulink® PLC Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Chart | State Transition Table

Topics

“Use Truth Tables to Model Combinatorial Logic”

“Program a Truth Table”

“Specify Properties for Stateflow Charts”

Introduced before R2006a

Functions

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 and Paste Stateflow Objects”

“Create Charts by Using the Stateflow API”

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

Selected objects in chart

Syntax

```
object = sfgco
```

Description

`object = sfgco` returns a handle or vector of handles to the most recently selected Stateflow objects. If more than one chart is open, the function searches the last chart with which you interacted that is still open.

Examples

Zoom in on Selected State

In the Stateflow Editor, select a state by clicking on it. Zoom in on the state by entering:

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

Display Names of Selected States

In the Stateflow Editor, simultaneously select several states by clicking each state while pressing the **Shift** key. Display the names of the states by entering:

```
myStates = sfgco;         % Get vector of handles to selected states
myStates.get('Name')      % Display the names of the selected states
```

Output Arguments

object — Selected graphical objects

handle | vector of handles

Selected graphical objects, returned as a handle or vector of handles to Stateflow API objects. This table describes the format and content of the output of the function, depending on your selection.

Value	Description
Empty matrix	You have not opened or edited any 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.

Value	Description
Vector of handles to the most recently selected objects in the most recently selected chart	You selected multiple objects in multiple charts.

See Also

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

Topics

[“Overview of the Stateflow API”](#)

[“Access Objects in Your Stateflow Chart”](#)

[“Create Charts by Using the Stateflow 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 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
-MATLAB	Chart that supports MATLAB expressions in actions
-C	Chart that supports C expressions in actions
-Mealy	Chart that supports Mealy machine semantics
-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”

“Use Truth Tables to Model Combinatorial Logic”

“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:

'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 an enhanced 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 'sf_car/shift_logic' in a JPEG format to a file in the current folder named 'sf_car_shift_logic.jpg'.

Example: `sfprint('sf_bounce/BouncingBall','meta','myImage')`

Prints the complete chart 'sf_bounce/BouncingBall' as an enhanced metafile in the current folder named 'myImage.emf'.

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:

```
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”

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 chart
ch = rt.find('-isa','Stateflow.Chart');

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

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



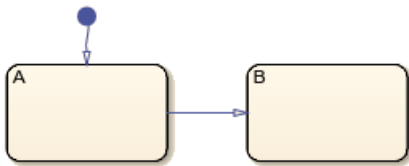
```
% Add a default transition to state A
dtA = Stateflow.Transition(ch);
dtA.Destination = sA;
dtA.DestinationOClock = 0;
x = sA.Position(1)+sA.Position(3)/2;
y = sA.Position(2)-30;
dtA.SourceEndPoint = [x y];

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

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

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

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”.

Compatibility Considerations

Opening Stateflow

Behavior change in future release

The behavior of the `stateflow` function will change in a future release. Use `sfnew` and `sflib` instead.

See Also

`edit` | `sflib` | `sfnew`

Introduced before R2006a

Stateflow.exportAsClass

Export MATLAB class for standalone chart

Syntax

```
Stateflow.exportAsClass(source)  
Stateflow.exportAsClass(source,destination)
```

Description

`Stateflow.exportAsClass(source)` saves a standalone Stateflow chart as a MATLAB class file in the current folder. The saved file has the same name as the chart. For example, if `source` is `chart.sfx`, the function saves the MATLAB class in the file `chart.m`.

`Stateflow.exportAsClass(source,destination)` saves the chart as a MATLAB class file in the folder `destination`.

Note The MATLAB class produced by `Stateflow.exportAsClass` is intended for debugging purposes only, and not for production use or manual modification. For more information, see “Tips” on page 2-22.

Examples

Export Chart in Current Folder

Save Stateflow chart `chart.sfx` as the MATLAB class file `chart.m` in the current folder.

```
Stateflow.exportAsClass('chart.sfx');
```

Export Chart in Folder Specified by Path

Save Stateflow chart `chart.sfx`, which is located in folder `dir1`, as the MATLAB class file `chart.m` in the current folder.

```
Stateflow.exportAsClass(fullfile('dir1','chart.sfx'));
```

Export Chart to MATLAB Class in Another Folder

Save Stateflow chart `chart.sfx`, which is located in the current folder, as the MATLAB class file `chart.m` in the folder `dir2`.

```
Stateflow.exportAsClass('chart.sfx','dir2');
```

Input Arguments

source — Path and file name of standalone Stateflow chart

character vector | string scalar

Path and file name of a standalone chart, specified as a string scalar or character vector. You can use the absolute path from the root folder or the relative path from the current folder. Standalone charts have the extension `.sfx`.

Data Types: `char` | `string`

destination — Path of destination folder for MATLAB class file

character vector | string scalar

Path of the destination folder for the MATLAB class file, specified as a string scalar or character vector. You can use the absolute path from the root folder or the relative path from the current folder. If not specified, the function saves the MATLAB script file in the current folder.

Data Types: `char` | `string`

Tips

- Use the code produced by `Stateflow.exportAsClass` to debug run-time errors that are otherwise difficult to diagnose. For example, suppose that you encounter an error while executing a Stateflow chart that controls a MATLAB application. If you export the chart as a MATLAB class file, you can replace the chart with the class in your application and diagnose the error by using the MATLAB debugger.

Note Error messages produced by the MATLAB class point to different line numbers than the corresponding error messages produced by the Stateflow chart.

- When you execute the MATLAB class produced by `Stateflow.exportAsClass`, the Stateflow Editor does not animate the original chart.

See Also

`fullfile`

Topics

“Create Stateflow Charts for Execution as MATLAB Objects”

Introduced in R2019b

Stateflow.exportToVersion

Export standalone chart for use in previous version of Stateflow

Syntax

```
exported_file = Stateflow.exportToVersion(source,file_name,version)
```

Description

`exported_file = Stateflow.exportToVersion(source,file_name,version)` exports the chart `source` to a file named `file_name` in a format that the specified previous Stateflow version can load. You can only export to R2019a and later releases.

Examples

Export Chart to an Earlier Version of MATLAB

To complete the export process, you need access to the versions of Stateflow from which and to which you are exporting.

Using the later version of Stateflow, convert the standalone chart `chart.sfx`.

```
edit chart.sfx
Stateflow.exportToVersion('chart','chart_19a.sfx','R2019a')
```

Using the earlier version of Stateflow, open and resave the exported chart.

```
edit chart_19a.sfx
sfsave chart_19a
```

Input Arguments

source — Chart to export

character vector | string scalar

Chart to export, specified as a string scalar or character vector, without any file extension. The chart must be open in the Stateflow Editor and have no unsaved changes.

Example: 'chart'

Data Types: char | string

file_name — Exported file name

character vector | string scalar

Exported file name, specified as a string scalar or character vector. The exported file must not have the same name as the source chart.

Example: 'chart_19a.sfx'

Data Types: char | string

version — MATLAB release name`'R2019a' | 'R2019b' | ...`

MATLAB release name, specified as a string scalar or character vector. Release names are case sensitive. You can only export to R2019a and later releases.

Data Types: `char` | `string`

Output Arguments**exported_file — Absolute path to exported file**

character vector

Absolute path to exported file, returned as a character vector.

Tips

Attempting to execute an exported chart before resaving it will result in an error.

See Also

`edit` | `sfsave`

Topics

“Create Stateflow Charts for Execution as MATLAB Objects”

Introduced in R2020a

Operators

after

Execute chart after event broadcast or specified time

Syntax

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

Description

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

`after(n,tick)` returns `true` if the chart has woken up at least `n` times since the associated state became active. Otherwise, the operator returns `false`.

The implicit event `tick` is not supported when a Stateflow chart in a Simulink model has input events.

`after(n,time_unit)` returns `true` if at least `n` units of time have elapsed since the associated state became active. Otherwise, the operator returns `false`.

In charts in a Simulink model, specify `time_unit` as seconds (`sec`), milliseconds (`msec`), or microseconds (`usec`). If you specify `n` as an expression, the chart adjusts the temporal delay as the expression changes value during the simulation.

In standalone charts in MATLAB, specify `n` with a value greater than or equal to `0.001` and `time_unit` as seconds (`sec`). The operator creates a MATLAB `timer` object that generates an implicit event to wake up the chart. MATLAB `timer` objects are limited to 1 millisecond precision. For more information, see “Events in Standalone Charts”.

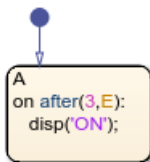
- The `timer` object is created when the chart finishes executing the `entry` actions of the associated state and its substates. If you specify `n` as an expression whose value changes during chart execution, the chart does not adjust the temporal delay of the `timer` object.
- The `timer` object starts running at the end of the chart step when the associated state becomes active. This step can include the execution of other parallel states in the chart.
- If the chart is processing another operation when it receives the implicit event from the `timer` object, the chart queues the event. When the current step is completed, the chart processes the event.
- If the state associated with the temporal logic operator becomes inactive before the chart processes the implicit event, the event does not wake up the chart.

Examples

Execute State Action on Event Broadcast

Display a status message when the chart processes a broadcast of the event E, starting on the third broadcast of E after the state became active.

```
on after(3,E):
    disp('ON');
```



Trigger Transition on Event Broadcast

Transition out of the associated state when the chart processes a broadcast of the event E, starting on the fifth broadcast of E after the state became active.

```
after(5,E)
```

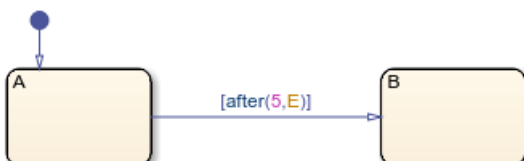


Guard Transition with Temporal Condition

Transition out of the associated state if the state has been active for at least five broadcasts of the event E.

In charts in a Simulink model, enter:

```
[after(5,E)]
```



Conditional notation for temporal logic operators is not supported in standalone charts in MATLAB.

Trigger Transition on Chart Execution

Transition out of the associated state when the chart wakes up for at least the seventh time since the state became active, but only if the variable `temp` is greater than 98.6.

```
after(7,tick)[temp > 98.6]
```



Execute State Action After Specified Time

Set the `temp` variable to LOW every time that the chart wakes up, starting when the associated state is active for at least 12.3 seconds.

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



Tips

- You can use quotation marks to enclose the keywords 'tick', 'sec', 'msec', and 'usec'. For example, `after(5, 'tick')` is equivalent to `after(5, tick)`.
- The Stateflow chart resets the counter used by the `after` operator each time the associated state reactivates.
- The timing for absolute-time temporal logic operators depends on the type of Stateflow chart:
 - Charts in a Simulink model define absolute-time temporal logic in terms of simulation time.
 - Standalone charts in MATLAB define absolute-time temporal logic in terms of wall-clock time, which is limited to 1 millisecond precision.

The difference in timing can affect the behavior of a chart. For example, suppose that this chart is executing the `during` action of state A.



- In a Simulink model, the function call to `f` executes in a single time step and does not contribute to the simulation time. The transition from state A to state B occurs the first time the chart wakes up and state A has been active for at least 2 seconds. The value displayed by the entry action in state B depends only on the step size used by the Simulink solver.
- In a standalone chart, the function call to `f` can take several seconds of wall-clock time to complete. If the call lasts more than two seconds, the chart queues the implicit event associated with the `after` operator. The transition from state A to state B occurs when the function `f` finishes executing. The value displayed by the entry action in state B depends on the time the function call to `f` takes to complete.

See Also

`at` | `before` | `every` | `timer`

Topics

“Control Chart Execution by Using Temporal Logic”

“Use Events to Execute Charts”

“Control Chart Behavior by Using Implicit Events”

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.

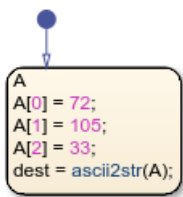
Note The operator `ascii2str` is supported only in Stateflow charts that use C as the action language.

Examples

Array of Type `uint8` to String

Return string "Hi!".

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



See Also

`str2ascii` | `strcpy`

Topics

“Manage Textual Information by Using Strings”
“Share String Data with Custom C Code”

Introduced in R2018b

at

Execute chart at event broadcast or specified time

Syntax

```
at(n,E)
at(n,tick)
at(n,sec)
```

Description

`at(n,E)` returns `true` if the event `E` has occurred exactly `n` times since the associated state became active. Otherwise, the operator returns `false`.

`at(n,tick)` returns `true` if the chart has woken up exactly `n` times since the associated state became active. Otherwise, the operator returns `false`.

The implicit event `tick` is not supported when a Stateflow chart in a Simulink model has input events.

`at(n,sec)` returns `true` if exactly `n` seconds have elapsed since the associated state became active. Otherwise, the operator returns `false`.

In standalone charts in MATLAB, specify `n` with a value greater than or equal to `0.001`. The operator creates a MATLAB `timer` object that generates an implicit event to wake up the chart. MATLAB `timer` objects are limited to 1 millisecond precision. For more information, see “Events in Standalone Charts”.

- The `timer` object is created when the chart finishes executing the entry actions of the associated state and its substates. If you specify `n` as an expression whose value changes during chart execution, the chart does not adjust the temporal delay of the `timer` object.
- The `timer` object starts running at the end of the chart step when the associated state becomes active. This step can include the execution of other parallel states in the chart.
- If the chart is processing another operation when it receives the implicit event from the `timer` object, the chart queues the event. When the current step is completed, the chart processes the event.
- If the state associated with the temporal logic operator becomes inactive before the chart processes the implicit event, the event does not wake up the chart.

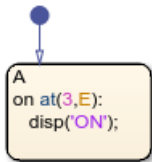
Note Using `at` as an absolute-time temporal logic operator is supported only in standalone charts in MATLAB. For charts in Simulink models, use the `after` operator instead. For more information, see “Do Not Use `at` for Absolute-Time Temporal Logic in Charts in Simulink Models”.

Examples

Execute State Action on Event Broadcast

Display a status message when the chart processes the third broadcast of the event E after the state became active.

```
on at(3,E):
    disp('ON');
```



Trigger Transition on Event Broadcast

Transition out of the associated state when the chart processes the fifth broadcast of the event E after the state became active.

```
at(5,E)
```



Guard Transition with Temporal Condition

Transition out of the associated state if the state has been active for exactly five broadcasts of the event E.

In charts in a Simulink model, enter:

```
[at(5,E)]
```



Conditional notation for temporal logic operators is not supported in standalone charts in MATLAB.

Trigger Transition on Chart Execution

Transition out of the associated state when the chart wakes up for the seventh time since the state became active, but only if the variable `temp` is greater than 98.6.

```
at(7,tick)[temp > 98.6]
```

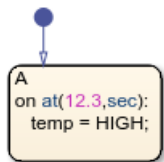


Execute State Action at Specified Time

Set the `temp` variable to HIGH if the state has been active for exactly 12.3 seconds.

In standalone charts in MATLAB, enter:

```
on at(12.3,sec):
    temp = HIGH;
```



Using `every` as an absolute-time temporal logic operator is not supported in charts in Simulink models.

Tips

- You can use quotation marks to enclose the keywords `'tick'` and `'sec'`. For example, `at(5, 'tick')` is equivalent to `at(5,tick)`.
- The Stateflow chart resets the counter used by the `at` operator each time the associated state reactivates.
- Standalone charts in MATLAB define absolute-time temporal logic in terms of wall-clock time, which is limited to 1 millisecond precision.

See Also

`after` | `before` | `every` | `timer`

Topics

“Control Chart Execution by Using Temporal Logic”

“Use Events to Execute Charts”

“Control Chart Behavior by Using Implicit Events”

Introduced in R2014b

before

Execute chart before event broadcast or specified time

Syntax

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

Description

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

`before(n,tick)` returns `true` if the chart has woken up fewer than `n` times since the associated state became active. Otherwise, the operator returns `false`.

The implicit event `tick` is not supported when a Stateflow chart in a Simulink model has input events.

`before(n,time_unit)` returns `true` if fewer than `n` units of time have elapsed since the associated state became active. Otherwise, the operator returns `false`.

Specify `time_unit` as seconds (`sec`), milliseconds (`msec`), or microseconds (`usec`). If you specify `n` as an expression, the chart adjusts the temporal delay as the expression changes value during the simulation.

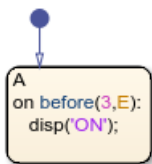
Note The temporal logic operator `before` is supported only in Stateflow charts in Simulink models.

Examples

Execute State Action on Event Broadcast

Display a status message when the chart processes the first and second broadcasts of the event `E` after the state became active.

```
on before(3,E):
    disp('ON');
```



Trigger Transition on Event Broadcast

Transition out of the associated state when the chart processes a broadcast of the event E, but only if the state has been active for fewer than five broadcasts of E.

```
before(5,E)
```



Guard Transition with Temporal Condition

Transition out of the associated state if the state has been active for fewer than five broadcasts of the event E.

```
[before(5,E)]
```



Trigger Transition on Chart Execution

Transition out of the associated state when the chart wakes up, but only if the variable `temp` is greater than 98.6 and the chart has woken up fewer than seven times since the state became active.

```
before(7,tick)[temp > 98.6]
```



Execute State Action Before Specified Time

Set the `temp` variable to `MED` every time that the chart wakes up, but only if the associated state has been active for fewer 12.3 seconds.

```
on before(12.3,sec):  
    temp = MED;
```



Tips

- You can use quotation marks to enclose the keywords 'tick', 'sec', 'msec', and 'usec'. For example, `before(5, 'tick')` is equivalent to `before(5, tick)`.
- The Stateflow chart resets the counter used by the `before` operator each time the associated state reactivates.

See Also

`after` | `at` | `every`

Topics

“Control Chart Execution by Using Temporal Logic”

Introduced in R2014b

change, chg

Generate implicit event when data changes value

Syntax

```
change(data_name)
chg(data_name)
```

Description

`change(data_name)` generates an implicit local event when the chart sets the value of the variable `data_name`. If more than one data object has the same name, use dot notation to specify the name of the data object. For more information, see “Identify Data by Using Dot Notation”.

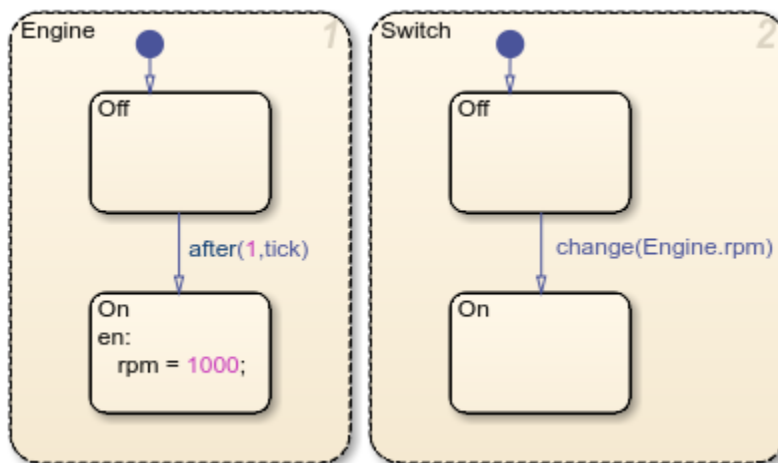
`chg(data_name)` is an alternative way to execute `change(data_name)`.

Examples

Implicit Event When Data Changes Value

Define an implicit local event when a state or transition action writes a value to the variable `Engine.rpm`.

```
change(Engine.rpm)
```



Tips

- The change operator is supported only in Stateflow charts in Simulink models.
- The change operator only works with data at the chart level or lower in the chart hierarchy. To determine when the value of machine-parented data changes, use change detection operators. For more information, see “Detect Changes in Data Values”.

See Also

hasChanged | hasChangedFrom | hasChangedTo

Topics

“Control Chart Behavior by Using Implicit Events”

“Use Events to Execute Charts”

“Detect Changes in Data Values”

Introduced before R2006a

count

Chart executions during which condition is valid

Syntax

```
count(C)
```

Description

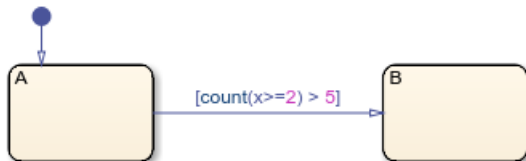
`count(C)` returns the number of times that the chart has woken up since the conditional expression `C` became `true` and the associated state became active.

Examples

Guard Transition with Temporal Condition

Transition out of the associated state when the variable `x` has been greater than or equal to 2 for longer than five chart executions.

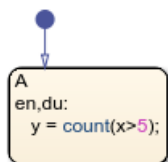
```
[count(x>=2) > 5]
```



Determine Number of Chart Executions

Store the number of chart executions since the variable `x` became greater than 5.

```
en,du:  
y = count(x>5);
```



Tips

- The Stateflow chart resets the value of the `count` operator if the conditional expression becomes `false` or if the associated state becomes inactive.

- When a chart in a Simulink model does not have input events, the value of `count` depends on the step size. Changing the solver or step size for the model affects the results produced by the `count` operator.
- To ensure that your Stateflow chart simulates without error, do not use `count` in these objects:
 - Continuous time charts
 - Graphical, MATLAB, or Simulink functions
 - Simulink based states
 - Transitions that can be reached from multiple states
 - Default transitions

See Also

[duration](#) | [elapsed](#) | [temporalCount](#)

Topics

[“Control Chart Execution by Using Temporal Logic”](#)

Introduced in R2019a

discard

Discard message

Syntax

```
discard(message_name)
```

Description

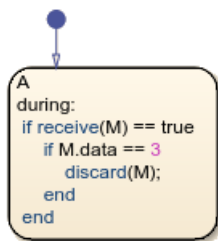
`discard(message_name)` discards a valid input or local message. After a chart discards a message, it can remove another message from the queue in the same time step. A chart cannot access the data of a discarded message.

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  
end
```



See Also

receive

Topics

“Control Message Activity in Stateflow Charts”

Introduced in R2018b

duration

Time during which condition is valid

Syntax

```
duration(C)
```

Description

`duration(C)` returns the number of seconds since the conditional expression `C` became `true` and the associated state became active.

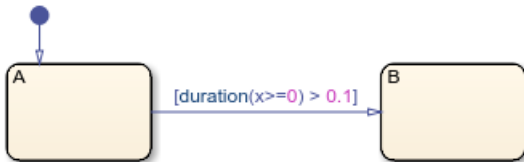
Note The temporal logic operator `duration` is not supported in standalone charts in MATLAB.

Examples

Guard Transition with Temporal Condition

Transition out of the state when the variable `x` has been greater than or equal to 0 for longer than 0.1 seconds.

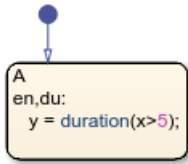
```
[duration(x>=0) > 0.1]
```



Determine Elapsed Time

Store the number of seconds since the variable `x` became greater than 5.

```
en,du:  
y = duration(x>5);
```



Tips

- The Stateflow chart resets the value of the `duration` operator if the conditional expression `C` becomes `false` or if the associated state becomes inactive.
- The `duration` operator does not support conditions that depend on local or output structures. For more information, see “Access Bus Signals Through Stateflow Structures”.

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”

“Implement an Automatic Transmission Gear System by Using the `duration` Operator”

Introduced in R2017a

elapsed

Time since state became active

Syntax

```
elapsed(sec)
et
```

Description

`elapsed(sec)` returns the length of time that has elapsed since the associated state became active. `et` is an alternative way to execute `elapsed(sec)`.

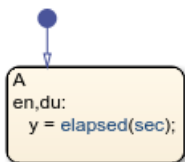
Note The expressions `elapsed(sec)` and `et` are equivalent to `temporalCount(sec)`.

Examples

Determine Time of State Activity

Store the number of seconds since the state became active.

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



Display Elapsed Time

When the chart processes a broadcast of the event E, transition out of the associated state and display the elapsed time since the state became active.

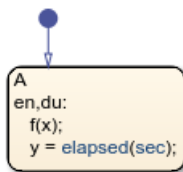
```
E{disp(et);}
```



Tips

- In state and transition actions, you can use quotation marks to enclose the keyword 'sec'. For example, `elapsed('sec')` is equivalent to `elapsed(sec)`.
- The Stateflow chart resets the counter used by the `elapsed` operator each time the associated state reactivates.
- The timing for absolute-time temporal logic operators depends on the type of Stateflow chart:
 - Charts in a Simulink model define temporal logic in terms of simulation time.
 - Standalone charts in MATLAB define temporal logic in terms of wall-clock time.

The difference in timing can affect the behavior of a chart. For example, suppose that this chart is executing the entry action of state A.



- In a Simulink model, the function call to `f` executes in a single time step and does not contribute to the simulation time. After calling the function `f`, the chart assigns a value of zero to `y`.
- In a standalone chart, the function call to `f` can take several seconds of wall-clock time to complete. After calling the function `f`, the chart assigns the nonzero time that has elapsed since state A became active to `y`.

See Also

[count](#) | [duration](#) | [temporalCount](#)

Topics

“Control Chart Execution by Using Temporal Logic”

Introduced in R2017a

enter, en

Generate implicit event when state becomes active

Syntax

```
enter(state_name)
en(state_name)
```

Description

`enter(state_name)` generates an implicit local event when the chart execution enters the state `state_name`. If more than one state has the same name, use dot notation to specify the name of the state. For more information, see “Identify Data by Using Dot Notation”.

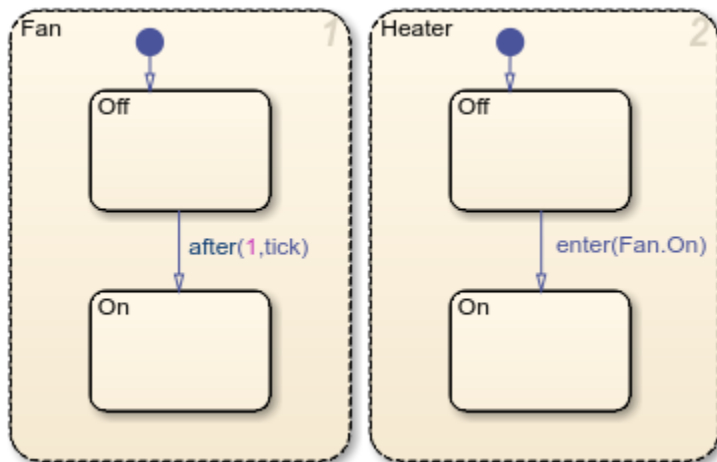
`en(state_name)` is an alternative way to execute `enter(state_name)`.

Examples

Implicit Event When State Becomes Active

Define an implicit local event when the chart execution enters the state `Fan.On`.

```
enter(Fan.On)
```



Tips

The `enter` operator is supported only in Stateflow charts in Simulink models.

See Also

`exit` | `in`

Topics

“Control Chart Behavior by Using Implicit Events”

“Use Events to Execute Charts”

“Check State Activity by Using the in Operator”

Introduced before R2006a

every

Execute chart at regular intervals

Syntax

```
every(n,E)
every(n,tick)
every(n,sec)
```

Description

`every(n,E)` returns `true` at every n^{th} occurrence of the event `E` since the associated state became active. Otherwise, the operator returns `false`.

`every(n,tick)` returns `true` at every n^{th} time that the chart wakes up since the associated state became active. Otherwise, the operator returns `false`.

The implicit event `tick` is not supported when a Stateflow chart in a Simulink model has input events.

`every(n,sec)` returns `true` every n seconds since the associated state became active. Otherwise, the operator returns `false`.

In standalone charts in MATLAB, specify n with a value greater than or equal to `0.001`. The operator creates a MATLAB timer object that generates an implicit event to wake up the chart. MATLAB timer objects are limited to 1 millisecond precision. For more information, see “Events in Standalone Charts”.

- The timer object is created when the chart finishes executing the entry actions of the associated state and its substates. For subsequent iterations, the timer object is reset when the chart finishes executing the during actions of the associated state and its substates. If you specify n as an expression whose value changes during chart execution, the chart adjusts the temporal delay only when the timer object is reset.
- The timer object starts running at the end of the chart step when the associated state becomes active. This step can include the execution of other parallel states in the chart.
- If the chart is processing another operation when it receives the implicit event from the timer object, the chart queues the event. When the current step is completed, the chart processes the event and resets the timer object for the next iteration.
- If the state associated with the temporal logic operator becomes inactive before the chart processes the implicit event, the event does not wake up the chart.

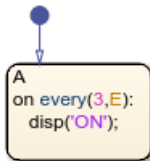
Note Using `every` as an absolute-time temporal logic operator is supported only in standalone charts in MATLAB. In charts in Simulink models, use an outer self-loop transition with the `after` operator instead. For more information, see “Do Not Use `every` for Absolute-Time Temporal Logic in Charts in Simulink Models”.

Examples

Execute State Action on Event Broadcast

Display a status message when the chart processes every third broadcast of the event E after the state became active.

```
on every(3,E):
    disp('ON');
```



Trigger Transition on Event Broadcast

Transition out of the associated state when the chart processes every fifth broadcast of the event E after the state became active.

```
every(5,E)
```



Trigger Transition on Chart Execution

Transition out of the associated state every seventh tick event since the state became active, but only if the variable temp is greater than 98.6.

```
every(7,tick)[temp > 98.6]
```

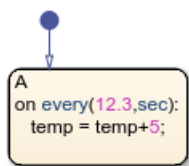


Execute State Action at Specified Time

Increment the temp variable by 5 every 12.3 seconds that the state is active.

In standalone charts in MATLAB, enter:

```
on every(12.3,sec):  
    temp = temp+5;
```



Using `every` as an absolute-time temporal logic operator is not supported in charts in Simulink models.

Tips

- You can use quotation marks to enclose the keywords 'tick' and 'sec'. For example, `every(5, 'tick')` is equivalent to `every(5,tick)`.
- The Stateflow chart resets the counter used by the `every` operator each time the associated state reactivates.
- Standalone charts in MATLAB define absolute-time temporal logic in terms of wall-clock time, which is limited to 1 millisecond precision.

See Also

`after` | `at` | `before` | `timer`

Topics

“Control Chart Execution by Using Temporal Logic”

“Use Events to Execute Charts”

“Control Chart Behavior by Using Implicit Events”

Introduced in R2014b

exit, ex

Generate implicit event when state becomes inactive

Syntax

```
exit(state_name)
ex(state_name)
```

Description

`exit(state_name)` generates an implicit local event when the chart execution exits the state `state_name`. If more than one state has the same name, use dot notation to specify the name of the state. For more information, see “Identify Data by Using Dot Notation”.

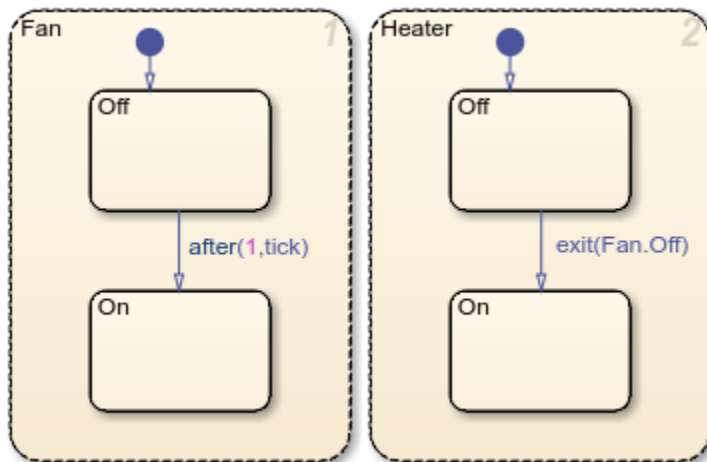
`ex(state_name)` is an alternative way to execute `exit(state_name)`.

Examples

Implicit Event When State Becomes Inactive

Define an implicit local event when the chart execution exits the state `Fan.Off`.

```
exit(Fan.Off)
```



Tips

The `exit` operator is supported only in Stateflow charts in Simulink models.

See Also

`enter` | `in`

Topics

“Control Chart Behavior by Using Implicit Events”

“Use Events to Execute Charts”

“Check State Activity by Using the in Operator”

Introduced before R2006a

forward

Forward message

Syntax

```
forward(message_in_name,message_out_name)
```

Description

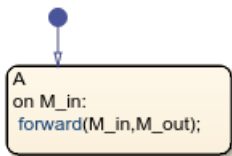
`forward(message_in_name,message_out_name)` forwards a valid input or local message to a local queue or an output port. After a chart forwards a message, it can remove another message from the queue in the same time step.

Examples

Forward an Input Message

Check the input queue for message `M_in`. If a message is present, remove the message from the queue and forward it to the output port `M_out`.

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



Forward a Local Message

Check the local queue for message `M_local`. If a message is present, transition from state A to state B. Remove the message from the `M_local` message queue and forward it to the output port `M_out`.

```
M_local{forward(M_local,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(data_name)
```

Description

`tf = hasChanged(data_name)` returns 1 (`true`) if the value of `data_name` at the beginning of the current time step is different from the value of `data_name` at the beginning of the previous time step. Otherwise, the operator returns 0 (`false`).

The argument `data_name` can be:

- A scalar variable.
- A matrix or an element of a matrix.
 - If `data_name` is a matrix, the operator returns `true` when it detects a change in any element of `data_name`.
 - Index elements of a matrix by using numbers or expressions that evaluate to a constant integer. See “Supported Operations for Vectors and Matrices”.
- A structure or a field in a structure.
 - If `data_name` is a structure, the change detection operator returns `true` when it detects a change in any field of `data_name`.
 - Index fields in a structure by using dot notation. See “Index and Assign Values to Stateflow Structures”.
- Any valid combination of structure fields or matrix elements.

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

Note Standalone charts in MATLAB do not support change detection on an element of a matrix or a field in a structure.

Examples

Detect Change in Matrix

Transition out of state if any element of the matrix `M` has changed value since the last time step or input event.

```
[hasChanged(M)]
```

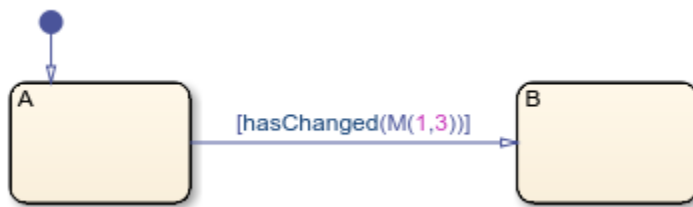


Detect Change in Matrix Element

Transition out of state 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, use:

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



In charts that use C as the action language, use:

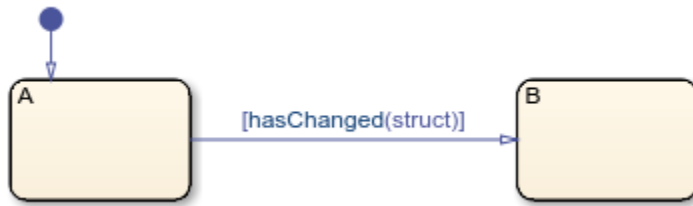
```
[hasChanged(M[0][2])]
```



Detect Change in Structure

Transition out of state 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

Transition out of state if the field `struct.field` has changed value since the last time step or input event.

`[hasChanged(struct.field)]`



Tips

- If multiple input events occur in the same time step, the `hasChanged` operator can detect changes in data value between input events.
- If the chart writes to the data object but does not change the data value, the `hasChanged` operator returns `false`.
- The type of Stateflow chart determines the scope of the data supported by the change detection operators:
 - Standalone Stateflow charts in MATLAB: `Local` only
 - In Simulink models, charts that use MATLAB as the action language: `Input` only
 - In Simulink models, charts that use C as the action language: `Input`, `Output`, `Local`, or `Data Store Memory`
- In a standalone chart in MATLAB, a change detection operator can detect changes in data specified in a call to the `step` function because these changes occur before the start of the current time step. For example, if `x` is equal to zero, the expression `hasChanged(x)` returns `true` when you execute the chart `ch` with the command:

```
step(ch, 'x', 1);
```

In contrast, a change detection operator cannot detect changes in data caused by assignments in state or transition actions in the same time step. Instead, the operator detects the change in value at the start of the next time step.

- In a chart in a Simulink model, if you enable the chart option **Initialize Outputs Every Time Chart Wakes Up**, using an output as the argument of the `hasChanged` operator always returns `false`. For more information, see “Initialize outputs every time chart wakes up”.

See Also

hasChangedFrom | hasChangedTo

Topics

“Detect Changes in Data Values”

“Supported Operations for Vectors and Matrices”

“Index and Assign Values to Stateflow Structures”

“Assign Values to All Elements of a Matrix”

Introduced in R2007a

hasChangedFrom

Detect change in data from specified value

Syntax

```
tf = hasChangedFrom(data_name,value)
```

Description

`tf = hasChangedFrom(data_name,value)` returns 1 (`true`) if the value of `data_name` was equal to the specified `value` at the beginning of the previous time step and is a different value at the beginning of the current time step. Otherwise, the operator returns 0 (`false`).

The argument `data_name` can be:

- A scalar variable.
- A matrix or an element of a matrix.
 - If `data_name` is a matrix, the operator returns `true` when it detects a change in any element of `data_name` and the previous value of `data_name` was equal to `value`.
 - Index elements of a matrix by using numbers or expressions that evaluate to a constant integer. See “Supported Operations for Vectors and Matrices”.
- A structure or a field in a structure.
 - If `data_name` is a structure, the change detection operator returns `true` when it detects a change in any element of `data_name` and the previous value of `data_name` was equal to `value`.
 - Index fields in a structure by using dot notation. See “Index and Assign Values to Stateflow Structures”.
- Any valid combination of structure fields or matrix elements.

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

Note Standalone charts in MATLAB do not support change detection on an element of a matrix or a field in a structure.

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

- If `data_name` is a scalar, then `value` must resolve to a scalar value.
- If `data_name` is a matrix, then `value` must resolve to a matrix value with the same dimensions as `data_name`.

Alternatively, in a chart that uses `C` as the action language, `value` can resolve to a scalar value. The chart uses scalar expansion to compare `data_name` to a matrix whose elements are all equal to the value specified by `value`. See “Assign Values to All Elements of a Matrix”.

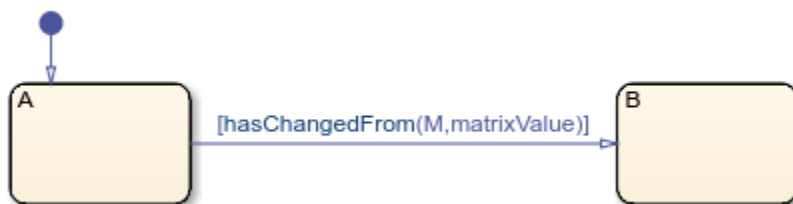
- If `data_name` is a structure, then `value` must resolve to a structure value whose field specification matches `data_name` exactly.

Examples

Detect Change in Matrix

Transition out of state 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

Transition out of state 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, use:

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



In charts that use C as the action language, use:

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



Detect Change in Structure

Transition out of state 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

Transition out of state if the field `struct.field` has changed from the value 5 since the last time step or input event.

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



Tips

- If multiple input events occur in the same time step, the `hasChangedFrom` operator can detect changes in data value between input events.
- If the chart writes to the data object but does not change the data value, the `hasChangedFrom` operator returns `false`.
- The type of Stateflow chart determines the scope of the data supported by the change detection operators:
 - Standalone Stateflow charts in MATLAB: `Local` only
 - In Simulink models, charts that use MATLAB as the action language: `Input` only
 - In Simulink models, charts that use C as the action language: `Input`, `Output`, `Local`, or `Data Store Memory`
- In a standalone chart in MATLAB, a change detection operator can detect changes in data specified in a call to the `step` function because these changes occur before the start of the current time step. For example, if `x` is equal to zero, the expression `hasChangedFrom(x,0)` returns `true` when you execute the chart `ch` with the command:

```
step(ch, 'x', 1);
```

In contrast, a change detection operator cannot detect changes in data caused by assignments in state or transition actions in the same time step. Instead, the operator detects the change in value at the start of the next time step.

- In a chart in a Simulink model, if you enable the chart option **Initialize Outputs Every Time Chart Wakes Up**, using an output as the argument of the `hasChanged` operator always returns `false`. For more information, see “Initialize outputs every time chart wakes up”.

See Also

`hasChanged` | `hasChangedTo`

Topics

“Detect Changes in Data Values”

“Supported Operations for Vectors and Matrices”

“Index and Assign Values to Stateflow Structures”

“Assign Values to All Elements of a Matrix”

Introduced in R2007a

hasChangedTo

Detect change in data to specified value

Syntax

```
tf = hasChangedTo(data_name,value)
```

Description

`tf = hasChangedTo(data_name,value)` returns 1 (`true`) if the value of `data_name` was not equal to the specified `value` at the beginning of the previous time step and is equal to `value` at the beginning of the current time step. Otherwise, the operator returns 0 (`false`).

The argument `data_name` can be:

- A scalar variable.
- A matrix or an element of a matrix.
 - If `data_name` is a matrix, the operator returns `true` when it detects a change in any element of `data_name` and the new value of `data_name` is equal to `value`.
 - Index elements of a matrix by using numbers or expressions that evaluate to a constant integer. See “Supported Operations for Vectors and Matrices”.
- A structure or a field in a structure.
 - If `data_name` is a structure, the change detection operator returns `true` when it detects a change in any element of `data_name` and the new value of `data_name` is equal to `value`.
 - Index fields in a structure by using dot notation. See “Index and Assign Values to Stateflow Structures”.
- Any valid combination of structure fields or matrix elements.

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

Note Standalone charts in MATLAB do not support change detection on an element of a matrix or a field in a structure.

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

- If `data_name` is a scalar, then `value` must resolve to a scalar value.
- If `data_name` is a matrix, then `value` must resolve to a matrix value with the same dimensions as `data_name`.

Alternatively, in a chart that uses C as the action language, `value` can resolve to a scalar value. The chart uses scalar expansion to compare `data_name` to a matrix whose elements are all equal to the value specified by `value`. See “Assign Values to All Elements of a Matrix”.

- If `data_name` is a structure, then `value` must resolve to a structure value whose field specification matches `data_name` exactly.

Examples

Detect Change in Matrix

Transition out of state 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

Transition out of state 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, use:

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



In charts that use C as the action language, use:

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



Detect Change in Structure

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

```
[hasChangedTo(struct,structValue)]
```



Detect Change in Structure Field

Transition out of state if the field `struct.field` has changed to the value 5 since the last time step or input event.

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



Tips

- If multiple input events occur in the same time step, the `hasChangedTo` operator can detect changes in data value between input events.
- If the chart writes to the data object but does not change the data value, the `hasChangedTo` operator returns `false`.
- The type of Stateflow chart determines the scope of the data supported by the change detection operators:
 - Standalone Stateflow charts in MATLAB: `Local` only
 - In Simulink models, charts that use MATLAB as the action language: `Input` only
 - In Simulink models, charts that use C as the action language: `Input`, `Output`, `Local`, or `Data Store Memory`
- In a standalone chart in MATLAB, a change detection operator can detect changes in data specified in a call to the `step` function because these changes occur before the start of the current time step. For example, if `x` is equal to zero, the expression `hasChangedTo(x,1)` returns `true` when you execute the chart `ch` with the command:

```
step(ch, 'x',1);
```


In contrast, a change detection operator cannot detect changes in data caused by assignments in state or transition actions in the same time step. Instead, the operator detects the change in value at the start of the next time step.

- In a chart in a Simulink model, if you enable the chart option **Initialize Outputs Every Time Chart Wakes Up**, using an output as the argument of the `hasChanged` operator always returns `false`. For more information, see “Initialize outputs every time chart wakes up”.

See Also

`hasChanged` | `hasChangedFrom`

Topics

“Detect Changes in Data Values”

“Supported Operations for Vectors and Matrices”

“Index and Assign Values to Stateflow Structures”

“Assign Values to All Elements of a Matrix”

Introduced in R2007a

in

Check state activity in Stateflow

Syntax

```
in(state_name)
```

Description

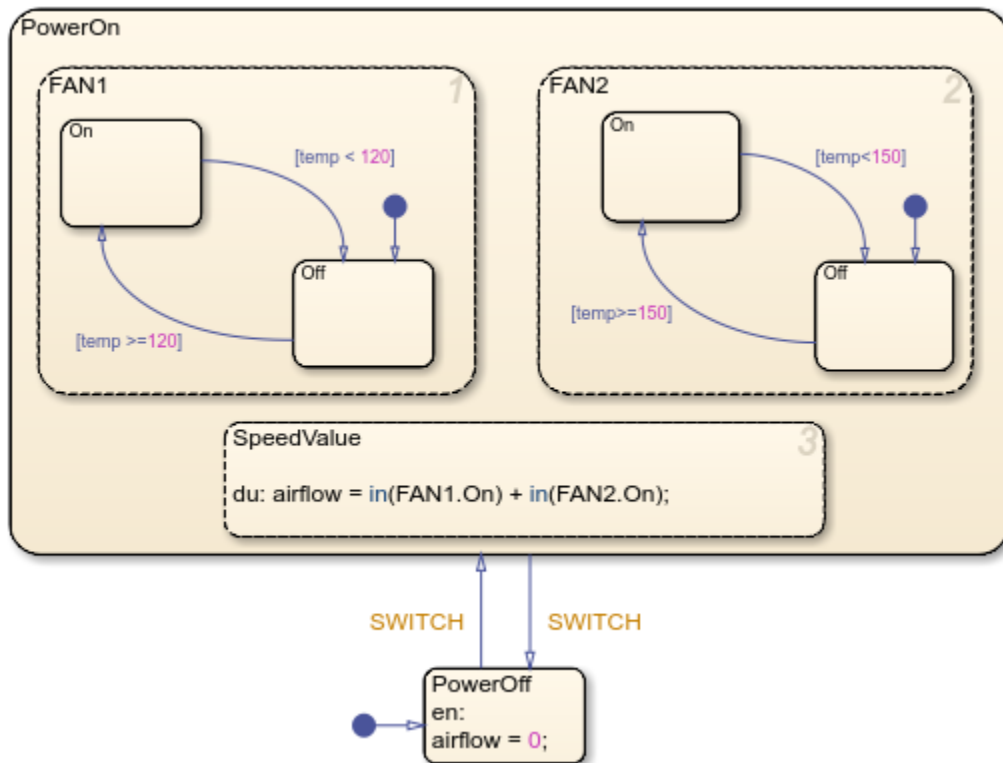
`in(state_name)` returns 1 (true) if the state `state_name` is active. Otherwise, the operator returns 0 (false).

Examples

Number of Active Subcomponents

Set the value of `airflow` to the number of fans that are turned on.

```
airflow = in(FAN1.0n) + in(FAN2.0n);
```



Tips

To determine the state activity, a Stateflow chart performs a localized search of the state hierarchy. The chart does not perform an exhaustive search for all states and does not stop after finding the first match. To improve the chances of finding a unique search result:

- Use dot notation to qualify the name of the state.
- Give states unique names.
- Use states and boxes as enclosures to limit the scope of the path resolution search.

For more information, see “Resolution of State Activity”.

See Also

enter | exit

Topics

“Check State Activity by Using the in Operator”

Introduced before R2006a

isvalid

Determine if message is valid

Syntax

```
isvalid(message_name)
```

Description

`isvalid(message_name)` checks if an input or local 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

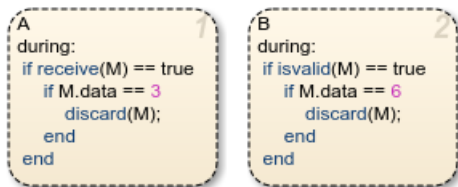
When state A is active, receive message M. If the message has a data value equal to 3, discard the message. Then, when state B is active, check that the message M is still valid. If the message is valid and has a data value equal to 6, discard the message.

In state A:

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

In state B:

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



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

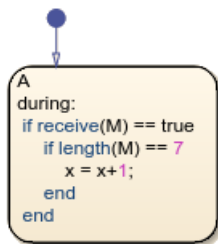
`length(message_name)` checks the number of messages in the internal receiving queue of an input or local message.

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  
end
```



Tips

- The length operator is not supported for input messages that use external receiving queues. To use the length operator, enable the **Use Internal Queue** property for this message.

See Also

receive

Topics

“Control Message Activity in Stateflow Charts”

Introduced in R2018b

receive

Extract message from queue

Syntax

```
receive(message_name)
```

Description

`receive(message_name)` extracts an input or local message from its receiving 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

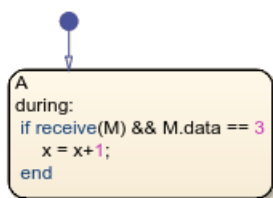
Extract Message in State Action

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, remove it from the queue regardless of the data value.

```
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(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(event_name)` sends a local or output event.

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

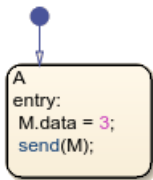
`send(state_name.local_event_name)` broadcasts a local event to its parent state `state_name` and any offspring of that state in the hierarchy.

Examples

Broadcast Message

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

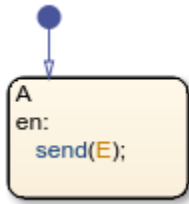
```
M.data = 3;
send(M);
```



Broadcast Output Event

Send an output event E.

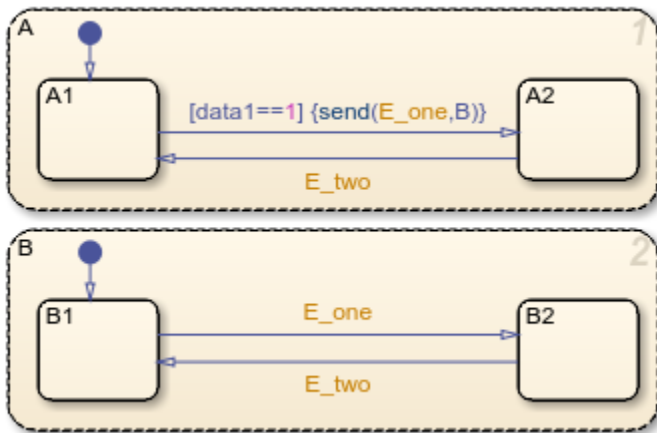
```
send(E);
```



Broadcast Directed Local Event

Send a local event E_one to state B and any of its substates.

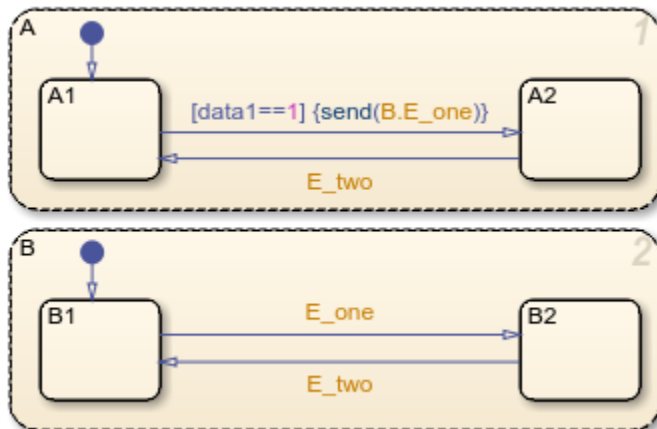
```
send(E_one,B);
```



Broadcast by Using Qualified Event Name

Send a local event E_one to its parent state B and any of its substates.

```
send(B.E_one);
```



Tips

- If a chart sends a message that exceeds the capacity of the receiving queue, a queue overflow occurs. The result of the queue overflow depends on the type of receiving queue.
 - When an overflow occurs in an internal queue, the Stateflow chart drops the new message. You can control the level of diagnostic action by setting the **Queue Overflow Diagnostic** property for the message. See “Queue Overflow Diagnostic”.
 - When an overflow occurs in an external queue, the Queue block either drops the new message or overwrites the oldest message in the queue, depending on the configuration of the block. See “Overwrite the oldest element if queue is full”. An overflow in an external queue always results in a warning.
- Avoid using undirected local event broadcasts. Undirected local event broadcasts can cause unwanted recursive behavior in your chart. Instead, send local events by using directed broadcasts. For more information, see “Broadcast Local Events to Synchronize Parallel States”.
- Use the send operator to send events to the Schedule Editor. The Schedule Editor enables you to schedule the execution of aperiodic partitions. For more information on using the send operator with the Schedule Editor, see “Events in Schedule Editor”.

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.

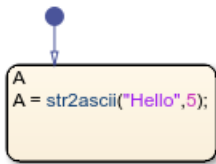
Note The operator `str2ascii` is supported only in Stateflow charts that use C as the action language.

Examples

String to ASCII Values

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

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



Tips

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.

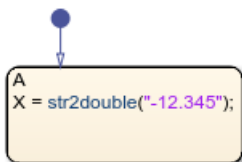
Note The operator `str2double` is supported only in Stateflow charts that use C as the action language.

Examples

String Containing Decimal Notation

Return a value of -12.345.

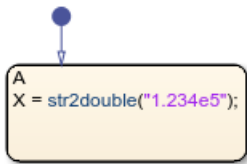
```
X = str2double("-12.345");
```



String Containing Exponential Notation

Return a value of 123400.

```
X = str2double("1.234e5");
```



```
A  
X = str2double("1.234e5");
```

Tips

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`.

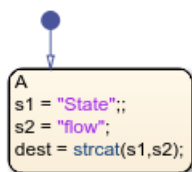
Note The operator `strcat` is supported only in Stateflow charts that use C as the action language.

Examples

Concatenation of Strings

Concatenate strings to form "Stateflow".

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



Tips

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.

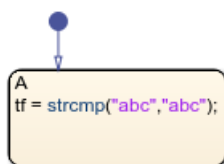
Note The operator `strcmp` is supported only in Stateflow charts that use C as the action language.

Examples

Comparison by Using strcmp

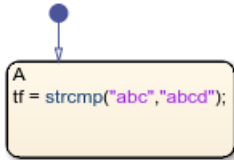
Return a value of 0 (strings are equal).

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



Return a nonzero value (strings are not equal).

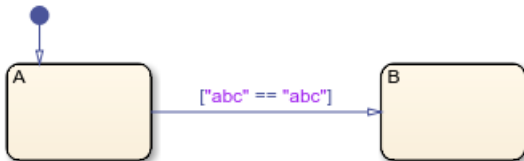
```
tf = strcmp("abc", "abcd");
```



Comparison by Using ==

Return a value of true.

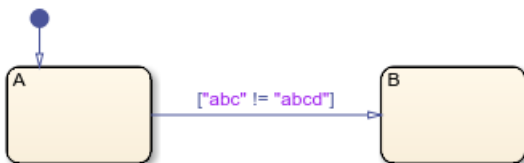
```
["abc" == "abc"]
```



Comparison by Using !=

Return a value of true.

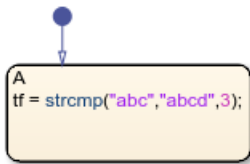
```
["abc" != "abcd"]
```



Comparison of Substrings

Return a value of 0 (substrings are equal).

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



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

Tips

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

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

Description

strcpy(dest,src) assigns string src to dest.

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

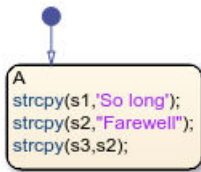
Note The operator strcpy is supported only in Stateflow charts that use C as the action language.

Examples

Assignment by Using strcpy

Assign string data to s1, s2, and s3.

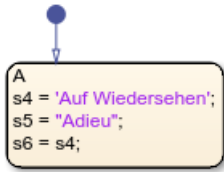
```
strcpy(s1,'So long');  
strcpy(s2,"Farewell");  
strcpy(s3,s2);
```



Assignment by Using =

Assign string data to s4, s5, and s6.

```
s4 = 'Auf Wiedersehen';  
s5 = "Adieu";  
s6 = s4;
```



```
A  
s4 = 'Auf Wiedersehen';  
s5 = "Adieu";  
s6 = s4;
```

Tips

- Source and destination arguments 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.

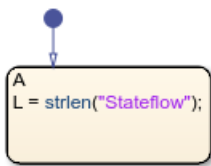
Note The operator strlen is supported only in Stateflow charts that use C as the action language.

Examples

Number of Characters in String

Return a value of 9.

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



Tips

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.

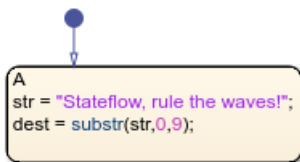
Note The operator `substr` is supported only in Stateflow charts that use C as the action language.

Examples

Extract Substring

Extract substring "Stateflow" from a longer string.

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



Tips

- 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

Number of events, chart executions, or time since state became active

Syntax

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

Description

`temporalCount(E)` returns the number of occurrences of the event `E` since the associated state became active.

`temporalCount(tick)` returns the number of times that the chart has woken up since the associated state became active.

The implicit event `tick` is not supported when a Stateflow chart in a Simulink model has input events.

`temporalCount(time_unit)` returns the length of time that has elapsed since the associated state became active. Specify `time_unit` as seconds (`sec`), milliseconds (`msec`), or microseconds (`usec`).

Note Standalone Stateflow charts in MATLAB support using `temporalCount` only as an absolute-time temporal logic operator.

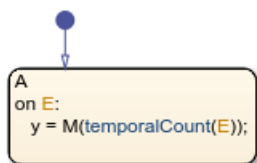
Examples

Perform Action on Event Broadcast

Access successive elements of the array `M` each time that the chart processes a broadcast of the event `E`.

In charts in a Simulink model, enter:

```
on E:
    y = M(temporalCount(E));
```



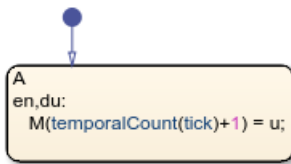
Using `temporalCount` as an event-based temporal logic operator is not supported in standalone charts in MATLAB.

Perform Action on Chart Execution

Store the value of the input data `u` in successive elements of the array `M`.

In charts in a Simulink model, enter:

```
en,du:  
M(temporalCount(tick)+1) = u;
```

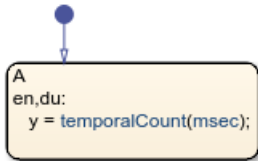


Using `temporalCount` as an event-based temporal logic operator is not supported in standalone charts in MATLAB.

Determine Time of State Activity

Store the number of milliseconds since the state became active.

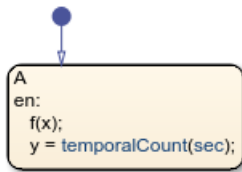
```
en,du:  
y = temporalCount(msec);
```



Tips

- You can use quotation marks to enclose the keywords 'tick', 'sec', 'msec', and 'usec'. For example, `temporalCount('tick')` is equivalent to `temporalCount(tick)`.
- The Stateflow chart resets the counter used by the `temporalCount` operator each time the associated state reactivates.
- The timing for absolute-time temporal logic operators depends on the type of Stateflow chart:
 - Charts in a Simulink model define temporal logic in terms of simulation time.
 - Standalone charts in MATLAB define temporal logic in terms of wall-clock time.

The difference in timing can affect the behavior of a chart. For example, suppose that this chart is executing the entry action of state `A`.



- In a Simulink model, the function call to `f` executes in a single time step and does not contribute to the simulation time. After calling the function `f`, the chart assigns a value of zero to `y`.
- In a standalone chart, the function call to `f` can take several seconds of wall-clock time to complete. After calling the function `f`, the chart assigns the nonzero time that has elapsed since state `A` became active to `y`.

See Also

count | duration | elapsed

Topics

“Control Chart Execution by Using Temporal Logic”

“Count Events by Using the temporalCount Operator”

Introduced in R2008a

this

Access chart data during simulation

Syntax

this

Description

this provides external MATLAB code, such as functions and apps, access to chart data during simulation.

- For charts in Simulink models, external MATLAB code can access inputs, outputs, and local data.
- For standalone charts in MATLAB, external MATLAB code can access local data and call `step`, input event functions, and graphical and MATLAB functions in the chart. For more information, see “Execute a Standalone Chart”.

Note In charts in Simulink models, the keyword `this` is supported only as an argument to external MATLAB code. Any other use of the keyword in the chart results in a compile-time error.

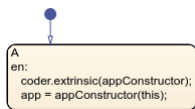
Examples

Connect Chart to MATLAB App

Create a bidirectional connection between a Stateflow chart and a MATLAB app created in App Designer. Call the app as an extrinsic function using `this` as an argument to the constructor. In the app, create a custom property to interface with the chart during simulation. In the chart, store the value returned by the function call to the app as a local data object.

In a chart that uses MATLAB as the action language, enter:

```
coder.extrinsic(appConstructor);
app = appConstructor(this);
```



In a chart that uses C as the action language, enter:

```
app = ml.appConstructor(this);
```



For additional examples that illustrate this workflow, see “Model a Power Window Controller” and “Simulate a Media Player”.

Change Data Value While Debugging Standalone Chart

Modify the value of the local data `x` while debugging a standalone Stateflow chart in MATLAB.

At the debugging prompt, enter:

```
this.x = 7
```

For more information, see “Examine and Change Values of Chart Data”.

Note When debugging a chart in a Simulink model, you can access all Stateflow data directly at the debugging prompt. For more information, see “View and Modify Data in the MATLAB Command Window”.

Tips

- Do not use the keyword `this` to access chart data after simulation has stopped.
- Calling an external function named `this` from a chart disables the keyword `this` throughout the chart. To use the keyword, rename the extrinsic function.

See Also

`coder.extrinsic`

Topics

“Model a Power Window Controller”

“Simulate a Media Player”

“Model a Fitness Tracker”

“Call Extrinsic MATLAB Functions in Stateflow Charts”

“Access MATLAB Functions and Workspace Data in C Charts”

“Debug a Standalone Stateflow Chart”

Introduced in R2020b

tostring

Convert numeric value to string

Syntax

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

Description

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

Note The operator `tostring` is supported only in Stateflow charts that use C as the action language.

Examples

Numeric Value to String

Convert numeric value to string "1.2345".

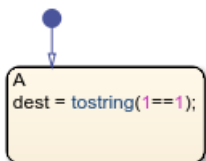
```
dest = tostring(1.2345);
```



Boolean Value to String

Convert Boolean value to string "true".

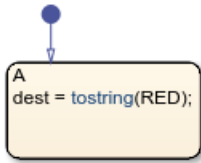
```
dest = tostring(1==1);
```



Enumerated Value to String

Convert enumerated value to string "RED".

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



See Also

[str2double](#) | [strcpy](#)

Topics

“Manage Textual Information by Using Strings”

Introduced in R2018b

Objects

Stateflow.SimulationData.Data

Data values during simulation

Description

Use `Stateflow.SimulationData.Data` to log the values of local and output data during simulation.

Creation

- 1 In the Symbols pane, select a local or output data object.
- 2 In the Property Inspector, under **Logging**, select the **Log signal data** check box.

Properties

Name — Logging name of data object

character array

Logging name of the data object, specified as a character array. By default, the logging name for a data object is the name of the data object. To assign another name to the data object, in the Property Inspector, under **Logging Name**, select Custom and enter a custom logging name.

Data Types: char

BlockPath — Block path for source block

`Simulink.SimulationData.BlockPath`

Block path for the source block, specified as a `Simulink.SimulationData.BlockPath` object.

Data Types: `Simulink.SimulationData.BlockPath`

Values — Logged data and time

timeseries

Logged data and time, specified as a timeseries object.

Data Types: timeseries

Object Functions

plot Plot simulation output data in the Simulation Data Inspector

Examples

Access Logged Data

- 1 Open the `sf_semantics_hotel_checkin` model.

- 2 Open the Hotel chart.
- 3 Open the Symbols pane. In the **Simulation** tab, in **Prepare**, click **Symbols Pane**.
- 4 Open the Property Inspector. In the **Simulation** tab, in **Prepare**, click **Property Inspector**.
- 5 Configure the `service` local data for logging.
 - In the Symbols pane, select `service`.
 - In the Property Inspector, on the **Logging** tab, select the **Log signal data** check box.
- 6 Return to the Simulink model.
- 7 Simulate the model. After starting the simulation, check into the hotel by toggling the first switch and order room service multiple times by toggling the second switch. During simulation, Stateflow saves logged data in a `Simulink.SimulationData.Dataset` signal logging object. The default name of the signal logging object is `logsout`. For more information, see “Export Signal Data Using Signal Logging”.
- 8 Stop the simulation.
- 9 To access the signal logging object, at the MATLAB command prompt, enter:

```
logsout = out.logsout
```

```
logsout =
```

```
Simulink.SimulationData.Dataset 'logsout' with 1 element
```

		Name	BlockPath
1	[1x1 Data]	service	sf_semantics_hotel_checkin/Hotel

- 10 To access logged element, use the `get` method.

```
serviceLog = logsout.get('service')
```

```
serviceLog =
```

```
Stateflow.SimulationData.Data  
Package: Stateflow.SimulationData
```

```
Properties:  
  Name: 'service'  
  BlockPath: [1x1 Simulink.SimulationData.BlockPath]  
  Values: [1x1 timeseries]
```

- 11 To access the logged data and time of each logged element, use the `Values.Data` and `Values.Time` properties. For example, arrange logged data in tabular form by using the `table` function.

```
Tbl = table(serviceLog.Values.Time,serviceLog.Values.Data);  
Tbl.Properties.VariableNames = {'Time','Data'}
```

```
Tbl =
```

```
6x2 table
```

Time	Data
1.7076e+06	0
1.8607e+06	1
1.9653e+06	2
1.9653e+06	3
1.9653e+06	4
2.2912e+06	5

See Also

Simulink.SimulationData.BlockPath | Stateflow.SimulationData.State | plot | timeseries

Topics

“Log Simulation Output for States and Data”
“Export Signal Data Using Signal Logging”

Introduced in R2017b

Stateflow.SimulationData.State

State activity during simulation

Description

Use `Stateflow.SimulationData.State` to log the activity of a state during simulation.

Creation

- 1 In the Stateflow Editor, select a state.
- 2 In the **Simulation** tab, in **Prepare**, select **Log Self Activity**. Alternatively, in the Property Inspector, under **Logging**, select the **Log self activity** check box.

Properties

Name — Logging name of state

character array

Logging name of the state, specified as a character array. By default, the logging name for a state is the hierarchical name using a period (.) to separate each level in the hierarchy of states. To assign a shorter name to the state, in the Property Inspector, set **Logging Name** to Custom and enter a custom logging name.

Data Types: char

BlockPath — Block path for source block

`Simulink.SimulationData.BlockPath`

Block path for the source block, specified as a `Simulink.SimulationData.BlockPath` object.

Data Types: `Simulink.SimulationData.BlockPath`

Values — State activity

timeseries

State activity, specified as a `timeseries` object. Data values represent whether the state is active (1) or not active (0). Time values correspond to simulation time.

Data Types: `timeseries`

Object Functions

`plot` Plot simulation output data in the Simulation Data Inspector

Examples

Access Logged State Activity

- 1 Open the `sf_semantics_hotel_checkin` model.
- 2 Open the `Hotel` chart.
- 3 Open the Symbols pane. In the **Simulation** tab, in **Prepare**, click **Symbols Pane**.
- 4 Configure the `Dining_area` state for logging.

- In the Stateflow Editor, select the `Dining_area` state.
- In the **Simulation** tab, under **Prepare**, select **Log Self Activity**.

In the Property Inspector, under **Logging**, select the **Log self activity** check box.

- By default, the logging name for this state is the hierarchical signal name `Check_in.Checked_in.Executive_suite.Dining_area`. To assign a shorter name to the state, set **Logging Name** to Custom and enter `Dining Room`.
- 5 Return to the Simulink model.
 - 6 Simulate the model. After starting the simulation, check into the hotel by toggling the first switch and order room service multiple times by toggling the second switch. During simulation, Stateflow saves logged data in a `Simulink.SimulationData.Dataset` signal logging object. The default name of the signal logging object is `logsout`. For more information, see “Export Signal Data Using Signal Logging”.
 - 7 Stop the simulation.
 - 8 To access the signal logging object, at the MATLAB command prompt, enter:

```
logsout = out.logsout
```

```
logsout =
```

```
Simulink.SimulationData.Dataset 'logsout' with 1 element
```

	Name	BlockPath
1 [1x1 State]	Dining Room	sf_semantics_hotel_checkin/Hotel

- 9 To access logged elements, use the `get` method.

```
diningLog = logsout.get('Dining Room')
```

```
diningLog =
```

```
Stateflow.SimulationData.State  
Package: Stateflow.SimulationData
```

```
Properties:
```

```
    Name: 'Dining Room'
```

```
    BlockPath: [1x1 Simulink.SimulationData.BlockPath]
```

```
    Values: [1x1 timeseries]
```

- 10 To access the logged data and time of each logged element, use the `Values.Data` and `Values.Time` properties. For example, arrange logged data in tabular form by using the `table` function.

```
Tbl = table(diningLog.Values.Time,diningLog.Values.Data);  
Tbl.Properties.VariableNames = {'Time','Data'}
```

```
Tbl =
```

```
6x2 table
```

Time	Data
0	0
1.8607e+06	1
1.9653e+06	0
1.9653e+06	1
1.9653e+06	0
2.2912e+06	1

See Also

`Simulink.SimulationData.BlockPath` | `Stateflow.SimulationData.Data` | `plot` | `timeseries`

Topics

“Log Simulation Output for States and Data”

“Export Signal Data Using Signal Logging”

Introduced in R2017b

Tools

Sequence Viewer

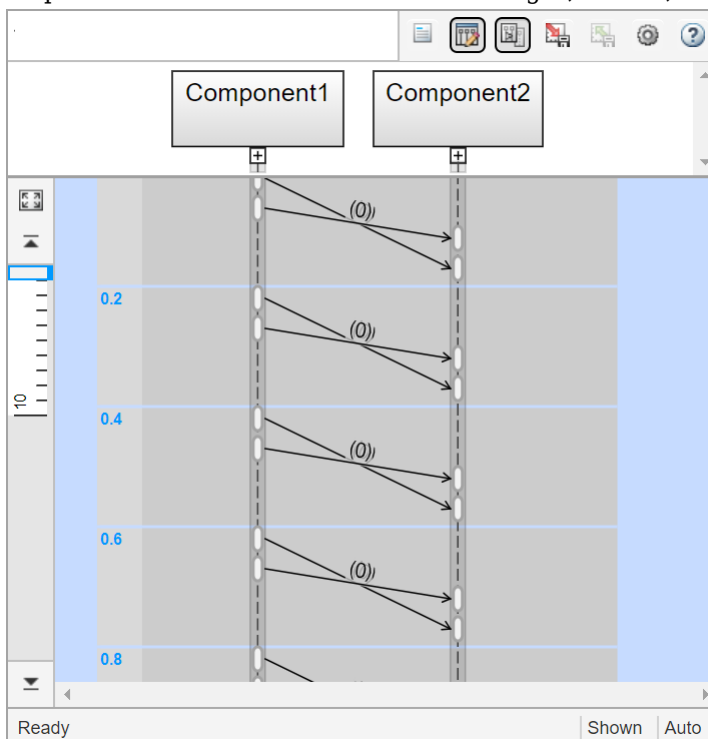
Visualize messages, events, states, transitions, and functions

Description

The Sequence Viewer visualizes message flow, function calls, and state transitions.

Use the Sequence Viewer to see the interchange of messages, events, function calls in Simulink behavior models in System Composer™ and between Stateflow charts in Simulink models.

In the Sequence Viewer window, you can view event data related to Stateflow chart execution and the exchange of messages between Stateflow charts. The Sequence Viewer window shows messages as they are created, sent, forwarded, received, and destroyed at different times during model execution. The Sequence Viewer window also displays state activity, transitions, and function calls to Stateflow graphical functions, Simulink functions, and MATLAB functions. For more information, see “Use the Sequence Viewer Block to Visualize Messages, Events, and Entities”.



Open the Sequence Viewer

- Simulink Toolstrip: On the **Simulation** tab, in the **Review Results** section, click **Sequence Viewer**.

Examples

Using the Sequence Viewer Tool

- 1 To activate logging events, in the Simulink Toolstrip, under the **Simulation** tab, in the **Prepare** section, click **Log Events**.
 - 2 Simulate your model.
 - 3 To open the tool, in the Simulink Toolstrip, under the **Simulation** tab, in the **Review Results** section, click **Sequence Viewer**.
- “Use the Sequence Viewer Block to Visualize Messages, Events, and Entities”
 - “Simulink Messages Overview”

Parameters

Sequence Viewer Time Precision — Digits for time increment precision

3 (default) | scalar

Number of digits for time increment precision. When using a variable step solver, change this parameter to adjust the time precision for the sequence viewer. By default the block supports 3 digits of precision. Minimum and maximum precision are 1 and 16, respectively.

Suppose the block displays two events that occur at times 0.1215 and 0.1219. Displaying these two events precisely requires 4 digits of precision. If the precision is 3, then the block displays two events at time 0.121.

Programmatic Use

Block Parameter: SequenceViewerTimePrecision

Type: character vector

Values: '3' | scalar

Default: '3'

Sequence Viewer History — Maximum number of previous events to display

1000 (default) | scalar

Total number of events before the last event to display. Minimum and maximum number of events are 0 and 25000, respectively.

For example, if **History** is 5 and there are 10 events in your simulation, then the block displays 6 events, including the last event and the five events prior the last event. Earlier events are not displayed. The time ruler is greyed to indicate the time between the beginning of the simulation and the time of the first displayed event.

Each send, receive, drop, or function call event is counted as one event, even if they occur at the same simulation time.

Programmatic Use

Block Parameter: SequenceViewerHistory

Type: character vector

Values: '1000' | scalar

Default: '1000'

See Also

Topics

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

“Simulink Messages Overview”

Introduced in R2020b