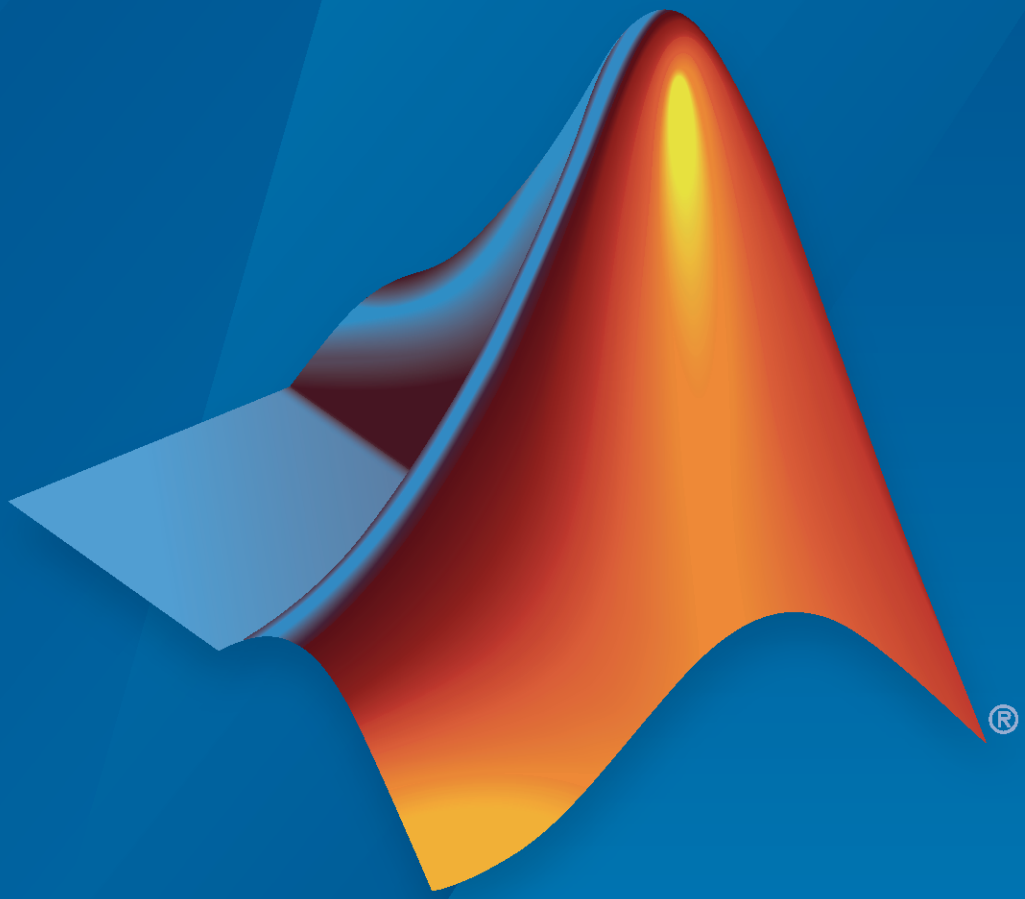
# Stateflow®

API



MATLAB® & SIMULINK®

R2021a

MathWorks®

# How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

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

Phone: 508-647-7000

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

*Stateflow® API*

© COPYRIGHT 2004–2021 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**

| | | |
|---|---|---|
| June 2004 | Online only | Revised for Version 6.0 (Release 14) |
| October 2004 | Online only | Revised for Version 6.1 (Release 14SP1) |
| March 2005 | Online only | Revised for Version 6.2 (Release 14SP2) |
| September 2005 | Online only | Revised for Version 6.3 (Release 14SP3) |
| March 2006 | Online only | Revised for Version 6.4 (Release 2006a) |
| September 2006 | Online only | Revised for Version 6.5 (Release 2006b) |
| 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 | Revised for Version 7.3 (Release 2009a) |
| September 2009 | Online only | Revised for Version 7.4 (Release 2009b) |
| March 2010 | Online only | Revised for Version 7.5 (Release 2010a) |
| September 2010 | Online only | Revised for Version 7.6 (Release 2010b) |
| April 2011 | Online only | Revised for Version 7.7 (Release 2011a) |
| September 2011 | Online only | Revised 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) |
| March 2021 | Online only | Revised for Version 10.4 (Release 2021a) |

# Contents

## 1 Using the Stateflow API

## 2 API Object Reference

## 3 API Object Function Reference

## 4 API Property Reference

# Using the Stateflow API

# Overview of the Stateflow API

| In this section... |
| --- |
|  |
|  |
|  |
|  |

The Stateflow application programming interface (API) allows you to create or change Stateflow charts from the MATLAB Command Window. By placing Stateflow API commands in a MATLAB function or script, you can:

- Automate your chart modification operations by executing several editing steps in a single command.
- Eliminate repetitive chart creation steps by producing a "base" Stateflow chart that you can reuse as a template for your applications.
- Produce a specialized report of your model.

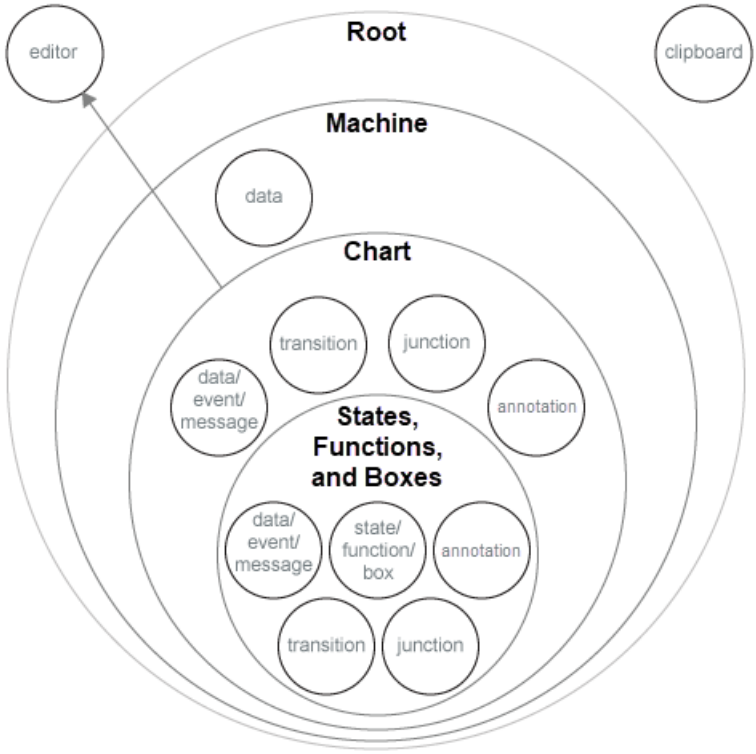
The Stateflow API consists of objects that represent the graphical and nongraphical objects of a Stateflow chart. For example, the API objects `Stateflow.State` and `Stateflow.Transition` represent states and transitions in a Stateflow chart. When you modify the properties of an API object or call one of its object functions, you affect the corresponding object in the Stateflow chart. When you use the Stateflow Editor to perform an operation on an object in the chart, you affect the corresponding API object.

**Note** You cannot undo any operation in the Stateflow Editor that you perform by using the Stateflow API. If you perform an editing operation through the API, the **Undo** and **Redo** buttons in the quick access toolbar are disabled.

## Hierarchy of Stateflow API Objects

Stateflow API objects are organized in a containment hierarchy. For example, if state `A` contains state `B` in a Stateflow chart, then the API object for state `A` contains the API object for state `B`. The Stateflow API hierarchy follows the same rules of containment as the Stateflow object hierarchy. For example, charts can contain states, but states cannot contain charts. For more information, see “Overview of Stateflow Objects”.

This diagram shows the hierarchy of objects in the Stateflow API.




The hierarchy consists of four levels of containment:

- **Root** — The `Simulink.Root` object is the parent of all Stateflow API objects. It is a placeholder at the top of the Stateflow API hierarchy that distinguishes Stateflow objects from other objects in a Simulink® model. You automatically create the `Simulink.Root` object when you add a Stateflow chart, a State Transition Table block, a Truth Table block, or a MATLAB Function block to a Simulink model, or when you load a model that contains one of these blocks.
- **Machine** — From a Stateflow perspective, `Stateflow.Machine` objects are equivalent to Simulink models. A `Stateflow.Machine` object contains objects that represent the Stateflow charts, State Transition Table blocks, Truth Table blocks, and MATLAB Function blocks in a model.
- **Chart** — `Stateflow.Chart`, `Stateflow.StateTransitionTableChart`, `Stateflow.TruthTableChart`, and `Stateflow.EMChart` objects represent Stateflow charts, State Transition Table blocks, Truth Table blocks, and MATLAB Function blocks, respectively. Objects in this level of the hierarchy can contain objects that represent states, functions, boxes, data, events, messages, transitions, junctions, and annotations.
- **States, Functions, and Boxes** — This level of the hierarchy includes `Stateflow.State`, `Stateflow.Function`, and `Stateflow.Box` objects that represent states, functions, and boxes, respectively. These objects can contain other objects that represent states, functions, boxes, data, events, messages, transitions, junctions, and annotations. Levels of nesting can continue indefinitely.

The hierarchy diagram shows two object types that exist outside of the containment hierarchy:

- **Editor** — `Stateflow.Editor` objects provide access to the graphical aspects of charts and state transition tables. For each `Stateflow.Chart` or `Stateflow.StateTransitionTableChart` object, there is a `Stateflow.Editor` object that you can use to control the position, size, and magnification level of the Stateflow Editor. For more information, see “Zoom in on Stateflow

Chart” on page 2-48, “Zoom out on Stateflow Chart” on page 2-49, and “Set Zoom Factor” on page 2-49.

- **Clipboard** — The `Stateflow.Clipboard` object has two functions, `copy` and `pasteTo`, that use the clipboard as a staging area to implement copy-and-paste functionality in the Stateflow API. For more information, see “Copy and Paste by Grouping” on page 2-24 and “Copy and Paste Array of Objects” on page 2-25.

## Access Stateflow API Objects

To use the Stateflow API, you begin by accessing the `Simulink.Root` object, which is the parent of all objects in the Stateflow API. You use the `Simulink.Root` object to access the other API objects in your model. For example:

1. Create a Simulink model with an empty Stateflow chart by calling the function `sfnew`.

   ```
   sfnew
   ```

2. Use the function `sfroot` to access the `Simulink.Root` object.

   ```
   rt = sfroot;
   ```

3. Call the `find` function to access the `Stateflow.Chart` object that corresponds to the chart in your model.

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

4. Call the `Stateflow.State` function to add a state to the chart. This function returns an `Stateflow.State` object that corresponds to the new state.

   ```
   st = Stateflow.State(ch);
   ```

5. Display the new state in the Stateflow Editor.

   ```
   view(st)
   ```

For more information, see “Access Objects in Your Stateflow Chart” on page 1-6 and “Create Charts by Using the Stateflow API” on page 1-19.

## Modify Properties of API Objects

API objects have properties that correspond to values that you normally set for an object through the Stateflow Editor. For example, to change the position of a state, you normally click and drag the state. With the Stateflow API, you change the position of a state by modifying the `Position` property of the corresponding `Stateflow.State` object:

```
st.Position = [10 20 100 80];
```

For more information, see “Modify Properties and Call Functions of Stateflow Objects” on page 1-10.

## Call API Object Functions

API objects have functions that provide services that are normally provided by the Stateflow Editor. For example, to open the Properties dialog box for a transition, you typically right-click the transition and select **Properties**. With the Stateflow API, you open this dialog box by calling the `dialog` function of the corresponding `Stateflow.Transition` object:

```
dialog(tr);
```

For more information, see “Modify Properties and Call Functions of Stateflow Objects” on page 1-10.

## See Also

**Functions**
`dialog` | `find` | `sfnew` | `sfroot` | `view`

**Objects**
`Stateflow.Box` | `Stateflow.Chart` | `Stateflow.Clipboard` | `Stateflow.EMChart` | `Stateflow.Editor` | `Stateflow.Function` | `Stateflow.Machine` | `Stateflow.State` | `Stateflow.StateTransitionTableChart` | `Stateflow.Transition` | `Stateflow.TruthTableChart`

## More About

- “Create Charts by Using the Stateflow API” on page 1-19
- “Create Charts by Using a MATLAB Script” on page 1-24
- “Access Objects in Your Stateflow Chart” on page 1-6
- “Modify Properties and Call Functions of Stateflow Objects” on page 1-10

# Access Objects in Your Stateflow Chart

The Stateflow API consists of objects that represent the graphical and nongraphical objects of a Stateflow chart. For example, the API objects `Stateflow.State` and `Stateflow.Transition` represent states and transitions in a Stateflow chart.

## Find Objects in a Chart

With the `find` function, you specify search criteria for the API object that you want to locate. You can combine criteria such as:

- The type of object
- The name of a property or function
- A property name and value

For example, this command searches the `Simulink.Root` object and returns every `Stateflow.State` object with the name `'On'`:

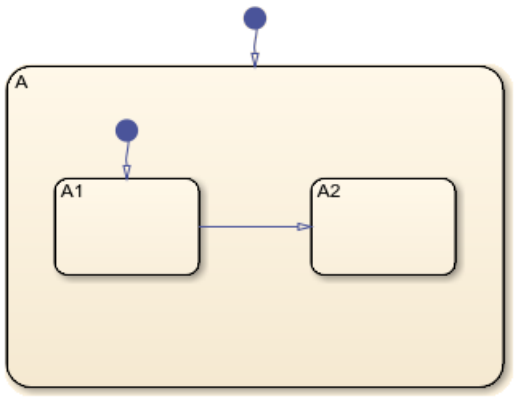
```
onState = find(sfroot,'-isa','Stateflow.State','Name','On')
```

If more than one object meets the search criteria, `find` returns an array of qualifying objects. For example, if more than one chart is open, this command returns an array of `Stateflow.Chart` objects:

```
chartArray = find(sfroot,'-isa','Stateflow.Chart')
```

## Find Objects at Specific Levels of Containment

Once you access an API object, you can navigate through the Stateflow hierarchy and find its children (the objects that it contains) or its parent (the object that contains it). For example, in this chart, state `A` is the parent state of the child states `A1` and `A2`. You can use the `find` and `up` functions to navigate from the chart to state `A` and its children, and back from the child states to the parent state and the chart. For more information on this example, see “Create Charts by Using a MATLAB Script” on page 1-24.




### Find Child Objects

To find the children of an API object, call the `find` function. By default, the `find` function finds objects at all depths of containment within an object. For instance, suppose that `ch` is the

`Stateflow.Chart` object that corresponds to the chart in the previous example. Calling the `find` function to find all the states in `ch` returns an array with three `Stateflow.State` objects:

```
states = find(ch,'-isa','Stateflow.State');
get(states,'Name')

ans =

  3×1 cell array

    {'A'}
    {'A1'}
    {'A2'}
```

To limit the maximum containment depth of a search, use the `'-depth'` argument as part of your search criteria. For example, to find the only `Stateflow.State` object at the first level of containment in `ch`, enter:

```
sA = find(ch,'-isa','Stateflow.State','-depth',1);
sA.Name

ans =

    'A'
```

Similarly, to find all the states in `A`, you can call the `find` function on the `Stateflow.State` object `sA`. In this case, the search includes the zeroth level of containment, which is the searched object itself:

```
states = find(sA,'-isa','Stateflow.State');
get(states,'Name')

ans =

  3×1 cell array

    {'A'}
    {'A1'}
    {'A2'}
```

To exclude the state `A` from the search results, call the MATLAB function `setdiff`:

```
childStates = setdiff(states,sA);
get(childStates,'Name')

ans =

  2×1 cell array

    {'A1'}
    {'A2'}
```

### Find Parent Object

To find the parent of an API object, call the `up` function. For instance, suppose that `sA1` is the `Stateflow.State` object that corresponds to state `A1` in the previous example. Calling the `up` function on `sA1` returns the `Stateflow.State` object that corresponds to the state `A`:

```
parent = up(sA1);
parent.Name
```

```
ans =

    'A'
```

Similarly, calling the `up` function on `parent` returns the `Stateflow.Chart` object that corresponds to the chart:

```
grandparent = up(parent);
grandparent.Name
```

```
ans =

    'Chart'
```

## Retrieve Recently Selected Objects

You can retrieve the most recently selected objects in a chart by calling the `sfgco` function. This function returns a single object or an array of objects, depending on your selection.

For instance, consider the chart in the previous example. Suppose that you select the transition from state `A1` to state `A2`. Calling `sfgco` returns the corresponding `Stateflow.Transition` object:

```
tr = sfgco;
str = ['Transition from ' tr.Source.Name ' to ' tr.Destination.Name]
```

```
str =

    'Transition from A1 to A2'
```

Similarly, if you simultaneously select the three states in the chart, calling `sfgco` returns an array of `Stateflow.State` objects.

```
states = sfgco;
get(states,'Name')
```

```
ans =

  3×1 cell array

    {'A'}
    {'A1'}
    {'A2'}
```

**Note** When you use `sfgco` to access multiple objects, the order of the objects in the array depends on the order in which you select the objects.

## See Also

**Functions**
`find` | `setdiff` | `sfgco` | `up`

**Objects**
`Stateflow.Chart` | `Stateflow.State` | `Stateflow.Transition`

## More About

- “Overview of the Stateflow API” on page 1-2
- “Modify Properties and Call Functions of Stateflow Objects” on page 1-10
- “Create and Delete Stateflow Objects” on page 1-13
- “Create Charts by Using the Stateflow API” on page 1-19

# Modify Properties and Call Functions of Stateflow Objects

| In this section... |
| --- |
|  |
|  |

Stateflow API objects have properties that correspond to values that you normally set through the Stateflow Editor. For example, to change the position of a state, you normally click and drag the state. With the Stateflow API, you change the position of a state by modifying the `Position` property of the corresponding `Stateflow.State` object:

```
st.Position = [10 20 100 80];
```

Additionally, object functions provide services that are normally provided by the Stateflow Editor. For example, to open the Properties dialog box for a transition, you typically right-click the transition and select **Properties**. With the Stateflow API, you open this dialog box by calling the `dialog` function of the corresponding `Stateflow.Transition` object:

```
dialog(tr);
```

## Use Properties and Object Functions

### Function-Call Notation

To call a function of an API object, use standard function-call notation. For example, to open the Chart properties dialog box, call the `dialog` function of the corresponding `Stateflow.Chart` object `ch`:

```
dialog(ch)
```

### Dot Notation

To access a property of an API object, use dot notation. For example, to see the value of the `StateMachineType` property for the `Stateflow.Chart` object `ch`, enter:

```
ch.StateMachineType
```

Similarly, to change the action language of the chart, modify its `ActionLanguage` property:

```
ch.ActionLanguage = 'MATLAB'
```

### Nested Dot Notation

To access the subproperties of an API property, you can nest multiple property names in a single expression that uses dot notation. For example, you can set an entry breakpoint on a chart by changing the subproperty `Debug.Breakpoints.OnEntry` of the corresponding `Stateflow.Chart` object:

```
ch.Debug.Breakpoints.OnEntry = true;
```

When a property or function returns another API object, you can also access the properties and functions for the second object by using nested dot notation. For example, the `Machine` property of a `Stateflow.Chart` returns the `Stateflow.Machine` object that contains the corresponding chart. To access the `Name` property of this `Stateflow.Machine` object, enter the expression:

```
machineName = ch.Machine.Name;
```

Similarly, the `defaultTransitions` function returns an array of `Stateflow.Transition` objects that correspond to the default transitions in the chart. If the chart contains only one default transition, you can retrieve its label by entering:

```
label = defaultTransitions(ch).LabelString;
```

If the chart contains more than one default transition, you must first store the array and then use an array index to retrieve each label:

```
transitions = defaultTransitions(ch);
label1 = transitions(1).LabelString;
label2 = transitions(2).LabelString;
```

#### Get and Set the Values of Multiple Properties

You can access multiple properties of an API object by calling the `get` function. For example, to obtain the name and description for the `Stateflow.Chart` object `ch`, enter:

```
chartInfo = get(ch,{'Name','Description'});
```

Similarly, you can change the value of multiple properties by calling the `set` function. For example, to change the name and description of the `Stateflow.Chart` object `ch`, enter:

```
set(ch,{'Name','Description'},{'Rectifier','Half-wave rectifier.'})
```

You can use the `get` and `set` functions to access or modify properties for every object in an array. For example, this command returns a cell array with the names and descriptions of the `Stateflow.Chart` objects in the array `chartArray`:

```
chartInfo = get(chartArray,{'Name','Description'});
```

## Display Property Information

The `get` function displays the names and values of the properties of an object. For example, to see the values of the properties of the `Stateflow.Chart` object `ch`, enter:

```
get(ch)
```

You can also call `get` to display the values of a subproperty of an object. For example, to see the values of the subproperties of the `StateFont` property of the `Stateflow.Chart` object `ch`, enter:

```
get(ch.StateFont)
```

#### Display Enumerated Values for Properties

Many API properties accept a limited number of possible values. To display a list of acceptable values for a property, call the `set` function. For example, this command displays the enumerated values allowed for the `Decomposition` property of a `Stateflow.Chart` object:

```
set(ch,'Decomposition')
```

### Display Additional Information for Properties

You can display additional information about the properties of an object by using the function `classhandle`. For example, this command displays a list of property names and data types of a `Stateflow.Chart` object:

```
get(classhandle(ch).Properties,{'Name','DataType'})
```

To see the fields that you can use with this syntax, enter:

```
get(classhandle(ch).Properties)
```

## See Also

**Functions**
`classhandle` | `defaultTransitions` | `dialog` | `fitToView` | `get` | `set`

**Objects**
`Stateflow.Chart` | `Stateflow.State` | `Stateflow.Transition`

## More About

- “Overview of the Stateflow API” on page 1-2
- “List of Stateflow API Properties” on page 4-2

# Create and Delete Stateflow Objects

The Stateflow API consists of objects that represent the graphical and nongraphical objects of a Stateflow chart. For example, API objects of type `Stateflow.State` and `Stateflow.Transition` represent states and transitions in a Stateflow chart.

## Create Stateflow Objects

Stateflow API objects are organized in the containment hierarchy described in “Hierarchy of Stateflow API Objects” on page 1-2. To create a Stateflow object as the child of a parent object, you begin by accessing the parent object. Then use the parent object as the input argument to a function that creates th child object. For example, to add a new `Stateflow.State` object in a `Stateflow.Chart` object, follow these steps:

1. Access the parent object `ch` as described in “Access Objects in Your Stateflow Chart” on page 1-6.
2. Call the `Stateflow.State` function using the parent object `ch` as an argument.

   ```
   st = Stateflow.State(ch);
   ```
3. Display the new state in the Stateflow Editor by calling the `view` function. Use the `Stateflow.State` object as the argument to the function.

   ```
   view(st)
   ```
4. Make changes to the state by modifying the properties of the `Stateflow.State` object. For example, you can set the name and position of the state by modifying the `Name` and `Position` properties. To set the `Position` property, specify the new position as a four-element vector in which the first two values are the (*x*,*y*) coordinates of the upper-left corner of the state and the last two values are the width and height of the state.

   ```
   st.Name = 'A';
   st.Position = [30 30 90 60];
   ```

   You can also connect the new state to other states or junctions in your chart by creating a `Stateflow.Transition` object and setting its `Source` or `Destination` properties to `st`.

For an example of how to add states, transitions, and data objects to a chart, see “Create Charts by Using the Stateflow API” on page 1-19.

### Graphical Object Containment

When you create a graphical object such as a state, function, box, junction, or annotation, it appears in the upper-left corner of its parent object. You can move the graphical object to a different location by modifying its `Position` property, as explained in the previous example.

When you create a transition, it appears in the upper-left corner of the chart or subchart where you can view the parent object. You can move the transition to a different location by setting its source and destination or by modifying its `SourceEndPoint`, `MidPoint`, and `DestinationEndPoint` properties.

A graphical object must be located inside the boundary of its parent. Repositioning a graphical object can change its parent or result in an undefined parent error. You can check for this condition by examining the value of the `BadIntersection` property of an object. This property is `true` if the edges of the graphical object overlap with another graphical object. Set the position and size of objects so that they are separate from other objects.

You cannot move an object in a subcharted state, box, or graphical function to a different level of the chart hierarchy by changing its position. Instead, copy and paste the object from one parent object to another. Then delete the original object. For more information, see “Copy and Paste by Grouping” on page 2-24 and “Copy and Paste Array of Objects” on page 2-25.

#### Nongraphical Object Containment

When you create nongraphical objects such as data, events, or messages, they appear in the Model Explorer and in the Symbols Pane at the hierarchical level of their parent object. You can also see the location of the parent object by inspecting the `Path` property of an object.

You cannot change the parent of a nongraphical object programmatically. Instead, use the Model Explorer. For more information, see “Use the Model Explorer with Stateflow Objects”.

## Delete Stateflow Objects

You can delete most objects in a Stateflow chart by calling the function `delete`. For example, to delete a `Stateflow.State` object `st`, enter:

```
delete(st);
```

After you delete the state, the variable `st` still exists in the MATLAB workspace, but it is no longer associated with the state.

---

**Note** You cannot use the `delete` function to delete objects of these types:

- `Simulink.Root`
- `Stateflow.Machine`
- `Stateflow.Chart`
- `Stateflow.EMChart`
- `Stateflow.StateTransitionTableChart`
- `Stateflow.TruthTableChart`
- `Stateflow.Clipboard`
- `Stateflow.Editor`

---

## See Also

#### Functions
`delete` | `view`

#### Objects
`Stateflow.Chart` | `Stateflow.State` | `Stateflow.Transition`

## More About

- “Overview of the Stateflow API” on page 1-2
- “Access Objects in Your Stateflow Chart” on page 1-6

- “Create Charts by Using the Stateflow API” on page 1-19

# Specify Labels in States and Transitions Programmatically

When using the Stateflow API, specify the labels of states and transitions by assigning a character vector to the `LabelString` property.

To extract parts of the state or transition label, use the properties of the `Stateflow.State` and `Stateflow.Transition` objects listed in this table.

| API Object | Property | Description |
|---|---|---|
| `Stateflow.State` | `DuringAction` | Text in the `during` action in this state. This property is not supported in Moore charts. |
| | `EntryAction` | Text in the `entry` action in this state. This property is not supported in Moore charts. |
| | `ExitAction` | Text in the `exit` action in this state. This property is not supported in Moore charts. |
| | `MooreAction` | Text in the action in this state. This property is supported only in Moore charts. For more information, see “Design Rules for Moore Charts”. |
| | `Name` | Name of this state. |
| | `OnAction` | Text in the `on` actions in this state, parsed as a cell array of this form:<br>`{'trigger1','action1',...,'triggerN','actionN'}`<br>This property is not supported in Moore charts. |
| `Stateflow.Transition` | `Condition` | Text in the condition on this transition. |
| | `ConditionAction` | Text in the condition action on this transition. |
| | `TransitionAction` | Text in the transition action on this transition. |
| | `Trigger` | Text in the trigger on this transition. |

With the exception of `Name`, all of these properties are read-only. For more information on the syntax for state and transition labels, see “State Labels” and “Transition Labels”.

## Enter Labels on Transitions

Suppose that `tr` is the `Stateflow.Transition` object that corresponds to a transition. You can assign a label that specifies a trigger, condition, and condition action on this transition by entering:

```
tr.LabelString = 'trigger[guard]{action();}';
```


To extract the trigger, condition, and condition action specified by the transition label, enter:

```
trigger = tr.Trigger
```

```
trigger =

    'trigger'
```

```
cond = tr.Condition
```

```
cond =

    'guard'
```

```
action = tr.ConditionAction
```

```
action =

    'action();'
```

## Enter Multiline Labels in States

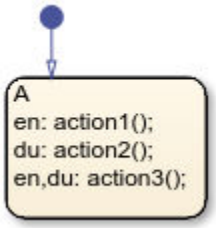
There are two equivalent ways to enter multiline labels for states and transitions. For example, Suppose that `sA` is a `Stateflow.State` object that corresponds to a state. To enter a multiline label with `entry` and `during` actions, you can:

- Call the MATLAB function `sprintf` and use the escape sequence `\n` to insert newline characters:

  ```
  str = sprintf('A\nen: action1();\ndu: action2();\nen,du: action3();');
  sA.LabelString = str;
  ```

- Enter a concatenated text expression that uses the integer 10 as the ASCII equivalent of a newline character:

  ```
  str = ['A',10, ...
      'en: action1();',10, ...
      'du: action2();',10, ...
      'en,du: action3();'];
  sA.LabelString = str;
  ```




To extract the state name, `entry` action, and `during` action specified by the state label, enter:

```
name = sA.Name
```

```
name =

    'A'
```

```
entry = sA.EntryAction
```

```
entry =
```

```
    ' action1();
      action3();'

during = sA.DuringAction

during =

    ' action2();
      action3();'
```

## See Also

**Functions**
`sprintf`

**Objects**
`Stateflow.State` | `Stateflow.Transition`

## More About

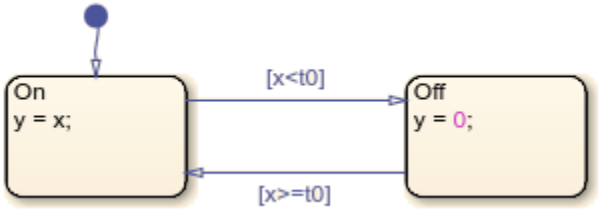
- “Overview of the Stateflow API” on page 1-2
- “Create Charts by Using the Stateflow API” on page 1-19
- “State Labels”
- “Transition Labels”

# Create Charts by Using the Stateflow API

This example shows how to create a Stateflow® chart by using the Stateflow application programming interface (API). The Stateflow API is a tool to create or change Stateflow charts through MATLAB® commands. For more information, see “Overview of the Stateflow API” on page 1-2.

### Create a Stateflow Chart

This Stateflow chart presents the logic underlying a half-wave rectifier. The chart contains two states labeled `On` and `Off`. In the `On` state, the chart output signal `y` is equal to the input `x`. In the `Off` state, the output signal is set to zero. When the input signal crosses some threshold `t0`, the chart transitions between these states. The actions in each state update the value of `y` at each time step of the simulation.




For more information on simulating this chart, see “Construct and Run a Stateflow Chart”.

1. Close all models.

```
bdclose all
```

2. Create a Simulink® model called `rectify` that contains an empty Stateflow Chart block.

```
sfnew rectify
```

### Access the `Chart` Object

To use the Stateflow API, you begin by accessing the `Simulink.Root` object, which is the parent of all objects in the Stateflow API. You use the `Simulink.Root` object to access the other API objects in your model.

1. Use the function `sfroot` to access the `Simulink.Root` object.

```
rt = sfroot;
```

2. Call the `find` function to access the `Stateflow.Chart` object that corresponds to the chart in your model.

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

3. To open the chart in the Stateflow Editor, call the `view` function.

```
view(ch);
```

4. To change the action language, modify the `ActionLanguage` property of the chart.

```
ch.ActionLanguage = 'C';
```

### Add States

To create a Stateflow API object as the child of a parent object, use the parent object as the input argument to a function that creates the child object. For more information, see “Create and Delete Stateflow Objects” on page 1-13.

1. Call the `Stateflow.State` function to add a state to the chart.

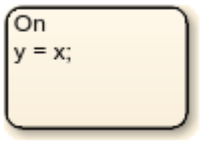
```
s1 = Stateflow.State(ch);
```

2. Adjust the position of the state by changing the `Position` property of the corresponding `State` object. Specify the new position as a four-element vector in which the first two values are the *(x,y)* coordinates of the upper-left corner of the state and the last two values are the width and height of the state.

```
s1.Position = [30 30 90 60];
```

3. Specify the name and label for the state by changing the `LabelString` property, as described in “Specify Labels in States and Transitions Programmatically” on page 1-16.

```
s1.LabelString = ['On',10,'y = x;'];
```




4. Create a second state. Adjust its position and specify its name and label.

```
s2 = Stateflow.State(ch);
s2.Position = [230 30 90 60];
s2.LabelString = ['Off',10,'y = 0;'];
```


### Add Transitions

When you add a transition, you specify its source and destination by modifying its `Source` and `Destination` properties. For a default transition, you specify a destination but no source.

1. Call the `Stateflow.Transition` function to add a transition to the chart.

```
t1 = Stateflow.Transition(ch);
```

2. Set the transition source and destination.

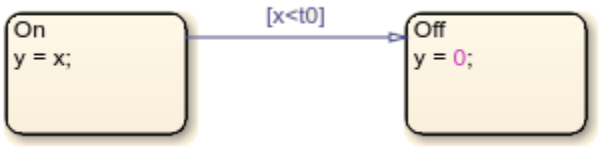
```
t1.Source = s1;
t1.Destination = s2;
```

3. Adjust the position of the transition by modifying its `SourceOClock` property.

```
t1.SourceOClock = 2.1;
```

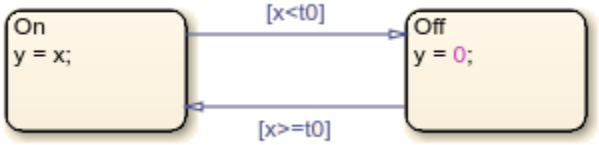
4. Specify the transition label and its position by changing the `LabelString` and `LabelPosition` properties.

```
t1.LabelString = '[x<t0]';
t1.LabelPosition= [159 23 31 16];
```



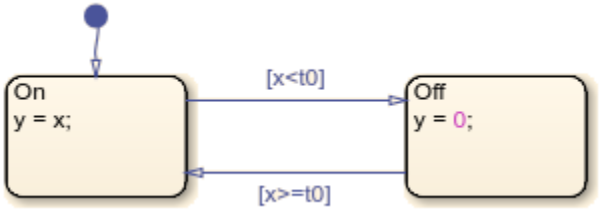

5. Create a second transition. Specify its source, destination, and label.

```
t2 = Stateflow.Transition(ch);
t2.Source = s2;
t2.Destination = s1;
t2.SourceOClock = 8.1;
t2.LabelString = '[x>=t0]';
t2.LabelPosition= [155 81 38 16];
```




6. Add a default transition to the state `On`. To make a vertical transition, modify the values of the `SourceEndpoint` and `Midpoint` properties. For more information, see “Add a Default Transition” on page 2-123.

```
t0 = Stateflow.Transition(ch);
t0.Destination = s1;
t0.DestinationOClock = 0;
t0.SourceEndpoint = t0.DestinationEndpoint-[0 30];
t0.Midpoint = t0.DestinationEndpoint-[0 15];
```




### Add Data

Before you can simulate your chart, you must define each data symbol that you use in the chart and specify its scope and type.

1. Call the `Stateflow.Data` function to add a data object that represents the input to the chart.

```
x = Stateflow.Data(ch);
```

2. Specify the name of the data object as `'x'` and its scope as `'Input'`.

```
x.Name = 'x';
x.Scope = 'Input';
```

3. To specify that the input `x` has type `double`, set its `Props.Type.Method` property to `'Built-in'`. The default built-in data type is `'double'`.

```
x.Props.Type.Method = 'Built-in';
x.DataType
```

```
ans =
'double'
```

4. Add a data object that represents the output for the chart. Specify its name as `'y'` and its scope as `'Output'`.

```
y = Stateflow.Data(ch);
y.Name = 'y';
y.Scope = 'Output';
```

5. To specify that the output y has type `single`, set its `Props.Type.Method` property to `'Built-in'` and its `DataType` property to `'single'`.

```
y.Props.Type.Method = 'Built-in';
y.DataType = 'single';
y.DataType
```

```
ans =
'single'
```

6. Add a data object that represents the transition threshold in the chart. Specify its name as `'t0'` and its scope as `'Constant'`. Set its initial value to 0.

```
t0 = Stateflow.Data(ch);
t0.Name = 't0';
t0.Scope = 'Constant';
t0.Props.InitialValue = '0';
```

7. To specify that the threshold `t0` has a fixed-point data type, set its `Props.Type.Method` property to `'Fixed-point'`. Then specify the values of the `Props.Type` properties that apply to fixed-point data.

```
t0.Props.Type.Method = 'Fixed point';
t0.Props.Type.Signed = true;
t0.Props.Type.WordLength = '5';
t0.Props.Type.Fixpt.ScalingMode = 'Binary point';
t0.Props.Type.Fixpt.FractionLength = '2';
t0.DataType
```

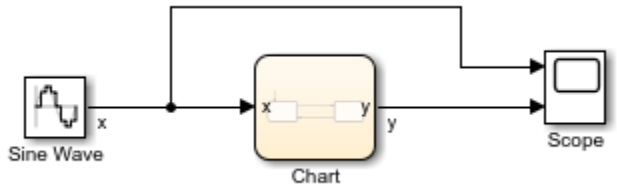
```
ans =
'fixdt(1,5,2)'
```

### Save and Simulate Your Chart

To save the model that contains your completed chart, call the `sfsave` function.

```
sfsave
```

To simulate the chart, connect it to other blocks in the Simulink model through input and output ports.




For more information, see “Simulate the Chart as a Simulink Block”.

## See Also

**Blocks**
Chart

**Functions**
`bdclose` | `find` | `sfnew` | `sfroot` | `sfsave` | `view`

**Objects**
`Stateflow.Data` | `Stateflow.State` | `Stateflow.Transition`

## More About

- “Overview of the Stateflow API” on page 1-2
- “Create and Delete Stateflow Objects” on page 1-13
- “Modify Properties and Call Functions of Stateflow Objects” on page 1-10
- “Specify Labels in States and Transitions Programmatically” on page 1-16

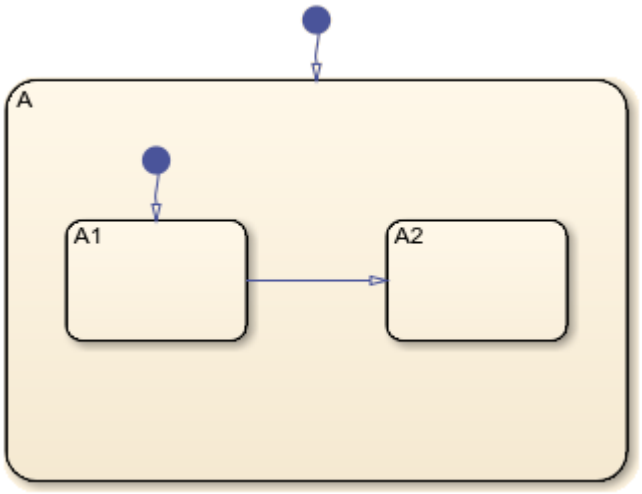
# Create Charts by Using a MATLAB Script

This example shows how to include Stateflow® API commands in a MATLAB® function or script. Creating a script of API commands allows you to avoid repetitive chart creation steps and recreate the same model with a single command. For more information, see “Overview of the Stateflow API” on page 1-2.

### Run the MATLAB Function

The function `makeMyModel`, which is defined at the bottom of this page on page 1-0 , produces a "base" Stateflow chart that you can reuse as a template for your applications.

```
ch = makeMyModel;
view(ch)
```




### Create Base Chart Function

This function creates a Stateflow chart and returns the corresponding `Stateflow.Chart` object.

```
function ch = makeMyModel
```

Create model and access new `Stateflow.Chart` object.

```
    rt = sfroot;
    prev_machines = find(rt,'-isa','Stateflow.Machine');
    sfnew;
    curr_machines = find(rt,'-isa','Stateflow.Machine');
    m = setdiff(curr_machines,prev_machines);
    ch = find(m,'-isa','Stateflow.Chart');
```

Create state `A` in chart.

```
    sA = Stateflow.State(ch);
    sA.Name = 'A';
    sA.Position = [50 50 310 200];
```

Create state `A1` inside of state `A`.

```
    sA1 = Stateflow.State(ch);
    sA1.Name = 'A1';
    sA1.Position = [80 120 90 60];
```

Create state A2 inside of state A.

```
    sA2 = Stateflow.State(ch);
    sA2.Name = 'A2';
    sA2.Position = [240 120 90 60];
```

Create transition from A1 to A2.

```
    tA1A2 = Stateflow.Transition(ch);
    tA1A2.Source = sA1;
    tA1A2.Destination = sA2;
    tA1A2.SourceOClock = 3;
    tA1A2.DestinationOClock = 9;
```

Add default transition to state A.

```
    dtA = Stateflow.Transition(ch);
    dtA.Destination = sA;
    dtA.DestinationOClock = 0;
    dtA.SourceEndPoint = dtA.DestinationEndpoint-[0 30];
    dtA.MidPoint = dtA.DestinationEndpoint-[0 15];
```

Add default transition to state A1.

```
    dtA1 = Stateflow.Transition(ch);
    dtA1.Destination = sA1;
    dtA1.DestinationOClock = 0;
    dtA1.SourceEndPoint = dtA1.DestinationEndpoint-[0 30];
    dtA1.MidPoint = dtA1.DestinationEndpoint-[0 15];
end
```

## See Also

**Functions**
`find` | `setdiff` | `sfnew` | `sfroot` | `view`

**Objects**
`Stateflow.State` | `Stateflow.Transition`

## More About

- “Overview of the Stateflow API” on page 1-2
- “Create Charts by Using the Stateflow API” on page 1-19

# API Object Reference

# Stateflow.Annotation

Annotation in chart, state, box, or function

# Description

Use `Stateflow.Annotation` objects to include descriptive comments in your chart. Annotations can contain any combination of:

- Text
- Images
- Equations using TeX commands
- Hyperlinks that open a website or perform MATLAB functions

For more information, see “Add Descriptive Comments in a Chart”.

# Creation

## Syntax

```
annotation = Stateflow.Annotation(parent)
```

### Description

`annotation = Stateflow.Annotation(parent)` creates a `Stateflow.Annotation` object in a parent chart, state, box, or graphical function.

### Input Arguments

#### `parent` — Parent for new annotation

`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

Parent for the new annotation, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.Function`
- `Stateflow.State`

## Properties

### Content

#### `Text` — Text for annotation

`'?'` (default) | character vector

Text for the annotation, specified as a character vector.

**`Alignment` — Alignment of text**
`'LEFT'` (default) | `'CENTER'` | `'RIGHT'`

Alignment of the annotation text, specified as `'LEFT'`, `'CENTER'`, or `'RIGHT'`.

**`Interpretation` — Format of text**
`'OFF'` (default) | `'RICH'` | `'TEX'`

Format of the annotation text, specified as `'OFF'`, `'RICH'`, or `'TEX'`.

**`PlainText` — Text without formatting**
character vector

This property is read-only.

Annotation text without formatting, specified as a character vector.

**`IsImage` — Whether annotation contains image**
`false` or 0 (default) | `true` or 1

This property is read-only.

Whether the annotation contains an image, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**Graphical Appearance**

**`Position` — Position and size of annotation box**
`[0 0 8 16]` (default) | `[left top width height]`

Position and size of annotation box, specified as a four-element numeric vector of the form `[left top width height]`.

**`InternalMargins` — Space between text and border of annotation box**
`[0 0 0 0]` (default) | `[left top right bottom]`

Space between the text and the border of the annotation box, specified as a four-element numeric vector of the form `[left top right bottom]`.

**`DropShadow` — Whether to display a drop shadow around annotation box**
`false` or 0 (default) | `true` or 1

Whether to display a drop shadow around the annotation box, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`FixedHeight` — Whether to fix height of annotation box**
`false` or 0 (default) | `true` or 1

Whether to fix height of the annotation box, specified as a numeric or logical 1 (`true`) or 0 (`false`).

- `true` — Fixes the height of the annotation box and hides content that is longer than the box.
- `false` — Resizes the annotation box vertically as you add content.

**`FixedWidth` — Whether to fix width of annotation box**
`false` or 0 (default) | `true` or 1

Whether to fix height of the annotation box, specified as a numeric or logical 1 (`true`) or 0 (`false`).

- `true` — Fixes the width of the annotation box and wraps text that is longer than the box.
- `false` — Resizes the annotation box horizontally as you add content.

**`BackgroundColor` — Background color**
`[1 1 1]` (default) | `[red green blue]`

Background color for the annotation, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1. This property applies only when the `AutoBackgroundColor` property is `false`.

**`ForegroundColor` — Foreground color**
`[0 0 0]` (default) | `[red green blue]`

Foreground color for the annotation, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1. This property applies only when the `AutoForegroundColor` property is `false`.

**`AutoBackgroundColor` — Whether to use default background color**
`true` or 1 (default) | `false` or 0

Whether to use the default background color, specified as a numeric or logical 1 (`true`) or 0 (`false`).

- `true` — Use the default color specified by the `ChartColor` property of the chart that contains the annotation.
- `false` — Use the color specified by the `BackgroundColor` property of the annotation.

**`AutoForegroundColor` — Whether to use default foreground color**
`true` or 1 (default) | `false` or 0

Whether to use the default foreground color, specified as a numeric or logical 1 (`true`) or 0 (`false`).

- `true` — Use the default color specified by the `StateLabelColor` property of the chart that contains the annotation.
- `false` — Use the color specified by the `ForegroundColor` property of the annotation.

**`Font.Angle` — Font angle**
`'NORMAL'` (default) | `'ITALIC'`

Font angle for the annotation text, specified as `'NORMAL'` or `'ITALIC'`.

Example: `annotation.Font.Angle = 'ITALIC';`

**`Font.Weight` — Font weight**
`'NORMAL'` (default) | `'BOLD'`

Font weight for the annotation text, specified as `'NORMAL'` or `'BOLD'`.

Example: `annotation.Font.Weight = 'BOLD';`

**`Font.Size` — Font size**
scalar

Font size for the annotation text, specified as a scalar. The `StateFont.Size` property of the chart that contains the annotation sets the initial value of this property.

Example: `annotation.Font.Size = 10;`

**`Font.Name` — Font name**
`'Helvetica'` (default) | character vector

This property is read-only.

Font name for the annotation text, specified as a character vector. The `StateFont.Name` property of the chart that contains the annotation sets the value of this property.

### Callbacks

**`ClickFcn` — Callback on click**
`''` (default) | character vector

Callback on click, specified as a character vector. This callback contains MATLAB code to execute when to execute when you click the annotation.

**`LoadFcn` — Callback at model load**
`''` (default) | character vector

Callback at model load, specified as a character vector. This callback contains MATLAB code to execute when you load the model that contains the annotation.

**`DeleteFcn` — Callback at delete**
`''` (default) | character vector

Callback at delete, specified as a character vector. This callback contains MATLAB code to execute before you delete the annotation.

**`UseDisplayTextAsClickCallback` — Whether to use annotation text as callback**
`false` or 0 (default) | `true` or 1

Whether to use the annotation text as a callback, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is enabled, the contents of the `Text` property is used as the callback when you click the annotation.

### Hierarchy

**`Chart` — Chart that contains annotation**
`Stateflow.Chart` object

This property is read-only.

Chart that contains the annotation, specified as a `Stateflow.Chart` object.

**`Subviewer` — Subviewer for annotation**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

This property is read-only.

Subviewer for the annotation, specified as a `Stateflow.Chart`, `Stateflow.State`, `Stateflow.Box`, or `Stateflow.Function` object. The subviewer is the chart or subchart where you can graphically view the annotation.

### `Machine` — Machine that contains annotation

`Stateflow.Machine` object

This property is read-only.

Machine that contains the annotation, specified as a `Stateflow.Machine` object.

### `Path` — Location of parent in model hierarchy

character vector

This property is read-only.

Location of the parent of the annotation in the model hierarchy, specified as a character vector.

### Identification

### `Description` — Description

`''` (default) | character vector

Description for the annotation, specified as a character vector.

### `Document` — Document link

`''` (default) | character vector

Document link for the annotation, specified as a character vector.

### `Tag` — User-defined tag

`[]` (default) | any data type | ...

User-defined tag for the annotation, specified as data of any type.

### `Id` — Unique identifier

scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Use this property to distinguish the annotation from other objects in the model. The value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| fitToView | Zoom in on graphical object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| set | Set properties with specified values |
| setImage | Insert image into annotation |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add Text Annotation to Chart

Add an annotation in the chart `ch`. Set its content to `'This is an annotation.'`

```
annotation = Stateflow.Annotation(ch);
annotation.Text = 'This is an annotation';
```

### Add Image Annotation to Chart

Add an annotation in the chart `ch`. Use the file `myImageFile.png`, which is located in the folder `myfolder/annotation_images`, as the image for the annotation.

```
annotation = Stateflow.Annotation(ch);
setImage(annotation, ...
    fullfile('myfolder','annotation_images','myImageFile.png');
```

## See Also

`Stateflow.Box` | `Stateflow.Chart` | `Stateflow.Function` | `Stateflow.State`

### Topics

“Overview of the Stateflow API” on page 1-2
“Add Descriptive Comments in a Chart”
“List of Stateflow API Properties” on page 4-2

### Introduced in R2017b

# Stateflow.AtomicBox

Atomic box in chart, state, box, or function

## Description

Use `Stateflow.AtomicBox` objects to encapsulate graphical, truth table, MATLAB, and Simulink functions in a separate namespace. Atomic boxes allow for:

- Faster simulation after making small changes to a function in a chart with many states or levels of hierarchy
- Reuse of the same functions across multiple charts and models
- Ease of team development for people working on different parts of the same chart
- Manual inspection of generated code for a specific function in a chart

For more information, see “Reuse Functions by Using Atomic Boxes”.

## Creation

### Syntax

```
atomicBox = Stateflow.AtomicBox(parent)
```

#### Description

`atomicBox = Stateflow.AtomicBox(parent)` creates a `Stateflow.AtomicBox` object in a parent chart, state, box, or graphical function.

#### Input Arguments

**`parent` — Parent for new atomic box**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

Parent for the new atomic box, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.Function`
- `Stateflow.State`

### Properties

#### Content

**`Name` — Name of atomic box**
`''` (default) | character vector

Name of the atomic box, specified as a character vector.

**`LabelString` — Full label for atomic box**
`'?'` (default) | character vector

Full label for the atomic box, specified as a character vector.

**`IsLink` — Whether atomic box is a library link**
`true` or 1 | `false` or 0

This property is read-only.

Whether the atomic box is a library link, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`IsExplicitlyCommented` — Whether to comment out atomic box**
`false` or 0 (default) | `true` or 1

Whether to comment out the atomic box, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the atomic box and selecting **Comment Out**. For more information, see "Commenting Stateflow Objects in a Chart".

**`IsImplicitlyCommented` — Whether atomic box is implicitly commented out**
`true` or 1 | `false` or 0

This property is read-only.

Whether the atomic box is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The atomic box is implicitly commented out when you comment out a superstate in its hierarchy.

**`CommentText` — Comment text**
`''` (default) | character vector

Comment text for the atomic box, specified as a character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge  on the atomic box, the text appears as a tooltip. When you set the `IsExplicitlyCommented` property to `false`, the value of `CommentText` reverts to `''`.

**Graphical Appearance**

**`Position` — Position and size of atomic box**
`[0 0 90 60]` (default) | `[left top width height]`

Position and size of the atomic box, specified as a four-element numeric vector of the form `[left top width height]`.

**`BadIntersection` — Whether atomic box intersects a box, state, or function**
`true` or 1 | `false` or 0

This property is read-only.

Whether the atomic box graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`ContentPreviewEnabled` — Whether to display preview of atomic box contents**
`false` or 0 (default) | `true` or 1

Whether to display a preview of the atomic box contents, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`FontSize` — Font size for atomic box label**
scalar

Font size for the atomic box label, specified as a scalar. The `StateFont.Size` property of the chart that contains the atomic box sets the initial value of this property.

**Hierarchy**

**`Chart` — Chart that contains atomic box**
`Stateflow.Chart` object

This property is read-only.

Chart that contains the atomic box, specified as a `Stateflow.Chart` object.

**`Subchart` — Contents of atomic box**
`Stateflow.Chart` object

This property is read-only.

Contents of the atomic box, specified as a `Stateflow.Chart` object. Use this object to add children, such as states and transitions, to the atomic box.

**`Subviewer` — Subviewer for atomic box**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

This property is read-only.

Subviewer for the atomic box, specified as a `Stateflow.Chart`, `Stateflow.State`, `Stateflow.Box`, or `Stateflow.Function` object. The subviewer is the chart or subchart where you can graphically view the atomic box.

**`Machine` — Machine that contains atomic box**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the atomic box, specified as a `Stateflow.Machine` object.

**`Path` — Location of parent in model hierarchy**
character vector

This property is read-only.

Location of the parent of the atomic box in the model hierarchy, specified as a character vector.

**Identification**

**`Description` — Description**
`''` (default) | character vector

Description for the atomic box, specified as a character vector.

### `Document` — Document link

`''` (default) | character vector

Document link for the atomic box, specified as a character vector.

### `Tag` — User-defined tag

`[]` (default) | any data type

User-defined tag for the atomic box, specified as data of any type.

### `SSIdNumber` — Session-independent identifier

scalar

This property is read-only.

Session-independent identifier, specified as an integer scalar. Use this property to distinguish the atomic box from other objects in the model.

### `Id` — Unique identifier

scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Unlike `SSIdNumber`, the value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| fitToView | Zoom in on graphical object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| highlight | Highlight graphical object |
| isCommented | Determine if object is commented out |
| set | Set properties with specified values |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add Atomic Box to Chart

Add an atomic box in the chart `ch`. Set its name to `'A'`.

```
atomicBox = Stateflow.AtomicBox(ch);
atomicBox.Name = 'A';
```

## See Also

`Stateflow.Box` | `Stateflow.Chart` | `Stateflow.Function` | `Stateflow.State`

### Topics

“Overview of the Stateflow API” on page 1-2

“Reuse Functions by Using Atomic Boxes”
“List of Stateflow API Properties” on page 4-2

**Introduced in R2012b**

# Stateflow.AtomicSubchart

Atomic subchart in chart, state, or box

## Description

Use `Stateflow.AtomicSubchart` objects to create independent subcomponents in a Stateflow chart. Atomic subcharts allow for:

- Reuse of the same state or subchart across multiple charts and models
- Faster simulation after making small changes to a chart with many states or levels of hierarchy
- Ease of team development when multiple people are working on different parts of the same chart
- Manual inspection of generated code for a specific state or subchart in a chart

For more information, see “Create Reusable Subcomponents by Using Atomic Subcharts”.

## Creation

### Syntax

```
atomicSubchart = Stateflow.AtomicSubchart(parent)
```

#### Description

`atomicSubchart = Stateflow.AtomicSubchart(parent)` creates a `Stateflow.AtomicSubchart` object in a parent chart, state, or box.

#### Input Arguments

**`parent` — Parent for new atomic subchart**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object

Parent for the new atomic subchart, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.State`

### Properties

#### Content

**`Name` — Name of atomic subchart**
`''` (default) | character vector

Name of the atomic subchart, specified as a character vector.

**`LabelString` — Full label for atomic subchart**
`'?'` (default) | character vector

Full label for the atomic subchart, specified as a character vector.

**`IsLink` — Whether atomic subchart is a library link**
`true` or 1 | `false` or 0

This property is read-only.

Whether the atomic subchart is a library link, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`IsExplicitlyCommented` — Whether to comment out atomic subchart**
`false` or 0 (default) | `true` or 1

Whether to comment out the atomic subchart, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the atomic subchart and selecting **Comment Out**. For more information, see "Commenting Stateflow Objects in a Chart".

**`IsImplicitlyCommented` — Whether atomic subchart is implicitly commented out**
`true` or 1 | `false` or 0

This property is read-only.

Whether the atomic subchart is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The atomic subchart is implicitly commented out when you comment out a superstate in its hierarchy.

**`CommentText` — Comment text**
`''` (default) | character vector

Comment text for the atomic subchart, specified as a character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge on the atomic subchart, the text appears as a tooltip. When you set the `IsExplicitlyCommented` property to `false`, the value of `CommentText` reverts to `''`.

**Graphical Appearance**

**`Position` — Position and size of atomic subchart**
`[0 0 90 60]` (default) | `[left top width height]`

Position and size of the atomic subchart, specified as a four-element numeric vector of the form `[left top width height]`.

**`BadIntersection` — Whether atomic subchart intersects a box, state, or function**
`true` or 1 | `false` or 0

This property is read-only.

Whether the atomic subchart graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`ContentPreviewEnabled` — Whether to display preview of atomic subchart contents**
`false` or 0 (default) | `true` or 1

Whether to display a preview of the atomic subchart contents, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`ArrowSize` — Size of incoming transition arrows**
8 (default) | scalar

Size of incoming transition arrows, specified as a scalar.

**`FontSize` — Font size for atomic subchart label**
scalar

Font size for the atomic subchart label, specified as a scalar. The `StateFont.Size` property of the chart that contains the atomic subchart sets the initial value of this property.

**State Decomposition**

**`Type` — Decomposition of sibling states**
`'AND'` | `'OR'`

This property is read-only.

Decomposition of sibling states, specified as `'OR'` or `'AND'`. The atomic subchart inherits this property from the `Decomposition` property of its parent state or chart.

**`ExecutionOrder` — Execution order in parallel (AND) decomposition**
scalar

Execution order for the atomic subchart in parallel (AND) decomposition, specified as an integer scalar. This property applies only when both of these conditions are satisfied:

- The `Type` property of the atomic subchart is `'AND'`.
- The `UserSpecifiedStateTransitionExecutionOrder` property of the chart that contains the atomic subchart is `true`.

**Active State Output**

**`HasOutputData` — Whether to create active state data output**
`false` or 0 (default) | `true` or 1

Whether to create an active state data output port for the atomic subchart, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Monitor State Activity Through Active State Data”.

**`OutputData` — Active state data object**
`Stateflow.Data` object

This property is read-only.

Active state data object for the atomic subchart, specified as a `Stateflow.Data` object. This property applies only when the `HasOutputData` property for the atomic subchart is `true`.

**`OutputPortName` — Name of active state data object**
character vector

Name of the active state data object for the atomic subchart, specified as a character vector. This property applies only when the `HasOutputData` property for the atomic subchart is `true`.

**`OutputMonitoringMode` — Monitoring mode for active state output**
`'SelfActivity'`

Monitoring mode for the active state output data, specified as a character vector. For atomic subcharts, the only option is `'SelfActivity'`.

**Signal Logging**

**`LoggingInfo.DataLogging` — Whether to enable signal logging for state**
`false` or 0 (default) | `true` or 1

Whether to enable signal logging for the atomic subchart, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Log Simulation Output for States and Data”.

Example: `atomicSubchart.LoggingInfo.DataLogging = true;`

**`LoggingInfo.DecimateData` — Whether to limit logged data**
`false` or 0 (default) | `true` or 1

Whether to limit the amount of logged data, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is `true`, signal logging skips samples by using the interval size specified by the `LoggingInfo.Decimation` property.

Example: `atomicSubchart.LoggingInfo.DeimateData = true;`

**`LoggingInfo.Decimation` — Decimation interval**
2 (default) | scalar

Decimation interval, specified as an integer scalar. The default value of 2 means that the chart logs every other sample.

Example: `atomicSubchart.LoggingInfo.Decimation = 5;`

**`LoggingInfo.LimitDataPoints` — Whether to limit number of data points to log**
`false` or 0 (default) | `true` or 1

Whether to limit the number of data points to log, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is `true`, signal logging limits the number of data points by using the value specified by the `LoggingInfo.MaxPoints` property.

Example: `atomicSubchart.LoggingInfo.LimitDataPoints = true;`

**`LoggingInfo.MaxPoints` — Maximum number of data points to log**
`5000` (default) | scalar

Maximum number of data points to log, specified as an integer scalar. The default value of `5000` means the chart logs the last 5000 data points generated by the simulation.

Example: `atomicSubchart.LoggingInfo.MaxPoints = 100;`

**`LoggingInfo.NameMode` — Source of signal name**
`'SignalName'` (default) | `'Custom'`

Source of the signal name used to log the atomic subchart, specified as one of these values:

- `'SignalName'` — Use the name of the atomic subchart.

- `'Custom'` — Use the custom signal name specified by the `LoggingInfo.LoggingName` property.

Example: `atomicSubchart.LoggingInfo.NameMode = 'Custom';`

**`LoggingInfo.LoggingName` — Custom signal name**
character vector

Custom signal name used for logging the atomic subchart, specified as a character vector. This property applies only when the `LoggingInfo.NameMode` property is `'Custom'`.

Example: `atomicSubchart.LoggingInfo.LoggingName = 'Atomic Subchart';`

**Debugging**

**`Debug.Breakpoints.OnDuring` — Whether to set `During State` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `During State` breakpoint for the atomic subchart, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `atomicSubchart.Debug.Breakpoints.OnDuring = true;`

**`Debug.Breakpoints.OnEntry` — Whether to set `On State Entry` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `On State Entry` breakpoint for the atomic subchart, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `atomicSubchart.Debug.Breakpoints.OnEntry = true;`

**`Debug.Breakpoints.OnExit` — Whether to set `On State Exit` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `On State Exit` breakpoint for the atomic subchart, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `atomicSubchart.Debug.Breakpoints.OnExit = true;`

**`TestPoint` — Whether to set atomic subchart as test point**
`false` or 0 (default) | `true` or 1

Whether to set the atomic subchart as a test point, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**Hierarchy**

**`Chart` — Chart that contains atomic subchart**
`Stateflow.Chart` object

This property is read-only.

Chart that contains the atomic subchart, specified as a `Stateflow.Chart` object.

**`Subchart` — Contents of atomic subchart**
`Stateflow.Chart` object

This property is read-only.

Contents of the atomic subchart, specified as a `Stateflow.Chart` object. Use this object to add children, such as states and transitions, to the atomic subchart.

**`Subviewer` — Subviewer for atomic subchart**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object

This property is read-only.

Subviewer for the atomic subchart, specified as a `Stateflow.Chart`, `Stateflow.State`, or `Stateflow.Box` object. The subviewer is the chart or subchart where you can graphically view the atomic subchart.

**`Machine` — Machine that contains atomic subchart**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the atomic subchart, specified as a `Stateflow.Machine` object.

**`Path` — Location of parent in model hierarchy**
character vector

This property is read-only.

Location of the parent of the atomic subchart in the model hierarchy, specified as a character vector.

**Identification**

**`Description` — Description**
`''` (default) | character vector

Description for the atomic subchart, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the atomic subchart, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the atomic subchart, specified as data of any type.

**`SSIdNumber` — Session-independent identifier**
scalar

This property is read-only.

Session-independent identifier, specified as an integer scalar. Use this property to distinguish the atomic subchart from other objects in the model.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Unlike `SSIdNumber`, the value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| fitToView | Zoom in on graphical object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| highlight | Highlight graphical object |
| isCommented | Determine if object is commented out |
| set | Set properties with specified values |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add Atomic Subchart to Chart

Add an atomic subchart in the chart `ch`. Set its name to `'A'`.

```
atomicSubchart = Stateflow.AtomicSubchart(ch);
atomicSubchart.Name = 'A';
```

## See Also

`Stateflow.Box` | `Stateflow.Chart` | `Stateflow.State`

### Topics

“Overview of the Stateflow API” on page 1-2
“Create Reusable Subcomponents by Using Atomic Subcharts”
“List of Stateflow API Properties” on page 4-2

### Introduced in R2010b

# Stateflow.Box

Box in chart, state, box, or function

# Description

Use `Stateflow.Box` objects to organize objects such as functions and states in your chart. You can also use a box to encapsulate states and functions in a separate namespace. For more information, see “Group Chart Objects by Using Boxes”.

# Creation

## Syntax

```
box = Stateflow.Box(parent)
```

### Description

`box = Stateflow.Box(parent)` creates a `Stateflow.Box` object in a parent chart, state, box, or graphical function.

### Input Arguments

**`parent` — Parent for new box**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

Parent for the new box, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.Function`
- `Stateflow.State`

## Properties

### Content

**`Name` — Name of box**
`''` (default) | character vector

Name of the box, specified as a character vector.

**`LabelString` — Full label for box**
`'?'` (default) | character vector

Full label for the box, specified as a character vector.

**`IsExplicitlyCommented` — Whether to comment out box**
`false` or 0 (default) | `true` or 1

Whether to comment out the box, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the box and selecting **Comment Out**. For more information, see “Commenting Stateflow Objects in a Chart”.

**`IsImplicitlyCommented` — Whether box is implicitly commented out**
`true` or 1 | `false` or 0

This property is read-only.

Whether the box is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The box is implicitly commented out when you comment out a superstate in its hierarchy.

**`CommentText` — Comment text**
`''` (default) | character vector

Comment text for the box, specified as a character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge on the box, the text appears as a tooltip. When you set the `IsExplicitlyCommented` property to `false`, the value of `CommentText` reverts to `''`.

**Graphical Appearance**

**`Position` — Position and size of box**
`[0 0 90 60]` (default) | `[left top width height]`

Position and size of the box, specified as a four-element numeric vector of the form `[left top width height]`.

**`BadIntersection` — Whether box intersects a box, state, or function**
`true` or 1 | `false` or 0

This property is read-only.

Whether the box graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`IsGrouped` — Whether box is a grouped box**
`false` or 0 (default) | `true` or 1

Whether the box is a grouped box, specified as a numeric or logical 1 (`true`) or 0 (`false`). When you copy and paste a grouped box, you copy not only the box but all of its contents. For more information, see “Copy and Paste by Grouping” on page 2-24.

**`IsSubchart` — Whether box is a subchart**
`false` or 0 (default) | `true` or 1

Whether the box is a subchart, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`ContentPreviewEnabled` — Whether to display preview of box contents**
`false` or 0 (default) | `true` or 1

Whether to display a preview of the box contents, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `IsSubchart` property is `true`.

**`FontSize` — Font size for box label**
scalar

Font size for the box label, specified as a scalar. The `StateFont.Size` property of the chart that contains the box sets the initial value of this property.

**Hierarchy**

**`Chart` — Chart that contains box**
`Stateflow.Chart` object

This property is read-only.

Chart that contains the box, specified as a `Stateflow.Chart` object.

**`Subviewer` — Subviewer for box**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

This property is read-only.

Subviewer for the box, specified as a `Stateflow.Chart`, `Stateflow.State`, `Stateflow.Box`, or `Stateflow.Function` object. The subviewer is the chart or subchart where you can graphically view the box.

**`Machine` — Machine that contains box**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the box, specified as a `Stateflow.Machine` object.

**`Path` — Location of parent in model hierarchy**
character vector

This property is read-only.

Location of the parent of the box in the model hierarchy, specified as a character vector.

**Identification**

**`Description` — Description**
`''` (default) | character vector

Description for the box, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the box, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the box, specified as data of any type.

### `Id` — Unique identifier

scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Use this property to distinguish the box from other objects in the model. The value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| defaultTransitions | Return default transitions in object at top level of containment |
| dialog | Open properties dialog box |
| find | Specified objects in hierarchy |
| fitToView | Zoom in on graphical object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| highlight | Highlight graphical object |
| innerTransitions | Return inner transitions that originate with chart or state and terminate on contained object |
| isCommented | Determine if object is commented out |
| outerTransitions | Return array of outer transitions for object |
| set | Set properties with specified values |
| sinkedTransitions | Return transitions that have object as destination |
| sourcedTransitions | Return transitions that have object as source |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add Box to Chart

Add a box in the chart `ch`. Set its name to `'A'`.

```
box = Stateflow.Box(ch);
box.Name = 'A';
```

## See Also

`Stateflow.Chart` | `Stateflow.Function` | `Stateflow.State`

### Topics

“Overview of the Stateflow API” on page 1-2
“Group Chart Objects by Using Boxes”
“List of Stateflow API Properties” on page 4-2

### Introduced before R2006a

# Stateflow.Clipboard

Clipboard to copy and paste Stateflow objects

## Description

Use the `Stateflow.Clipboard` object to copy and paste graphical and nongraphical objects within the same chart, between charts in the same Simulink model, or between charts in different models.

## Creation

There is only one `Stateflow.Clipboard` object, which is created automatically when you start Stateflow. To access this object, call the `sfclipboard` function:

```
clipboard = sfclipboard;
```

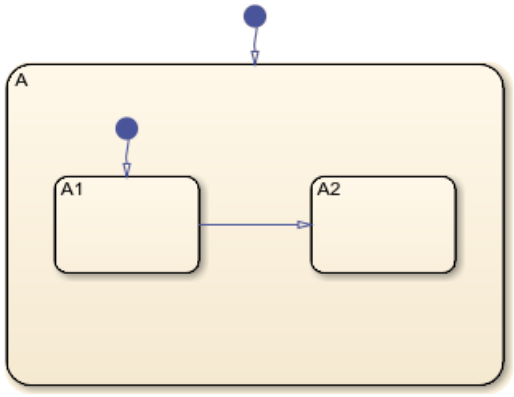
### Object Functions

| | |
|---|---|
| copy | Copy array of objects to clipboard |
| pasteTo | Paste objects in clipboard to specified container object |

### Examples

#### Copy and Paste by Grouping

Group state `A` and copy its contents to chart `ch`. When you group a state, box, or graphical function, you can copy and paste all the objects contained in the grouped object, as well as all the relationships among these objects. This method is the simplest way of copying and pasting objects programmatically. If a state is not grouped, copying the state does not copy any of its contents.




1 Find the `Stateflow.State` object named `A` in chart `ch`.

```
sA = find(ch,'-isa','Stateflow.State','Name','A');
```

2. Group state `A` and its contents by setting the `IsGrouped` property for `sA` to `true`. Save the previous setting of this property so you can revert to it later.

```
prevGrouping = sA.IsGrouped;
sA.IsGrouped = true;
```

3. Change the name of the state to `'Copy_of_A'`. Save the previous name so you can revert to it later.

```
prevName = sA.Name;
newName = ['Copy_of_' prevName];
sA.Name = newName;
```

4. Access the clipboard object.

```
cb = sfclipboard;
```

5. Copy the grouped state to the clipboard.

```
copy(cb,sA);
```

6. Restore the state properties to their original settings.

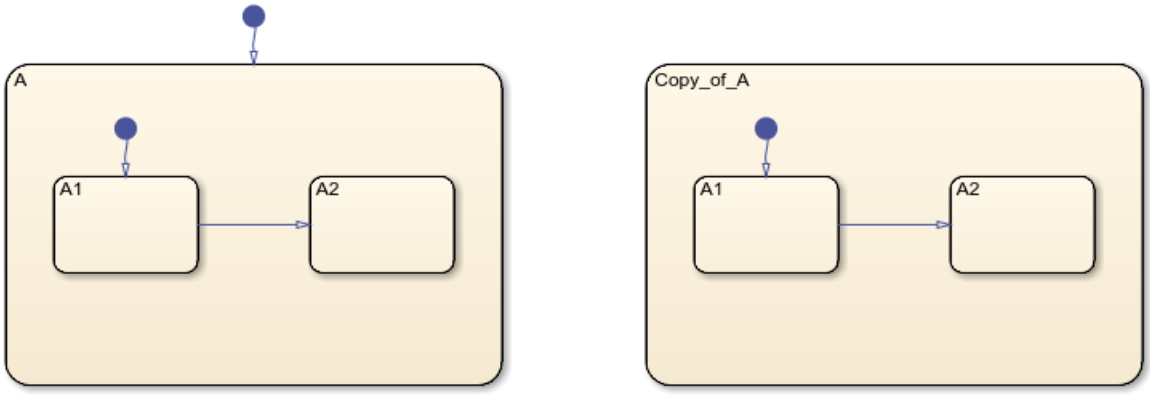
```
sA.IsGrouped = prevGrouping;
sA.Name = prevName;
```

7. Paste a copy of the objects from the clipboard to the chart.

```
pasteTo(cb,ch);
```

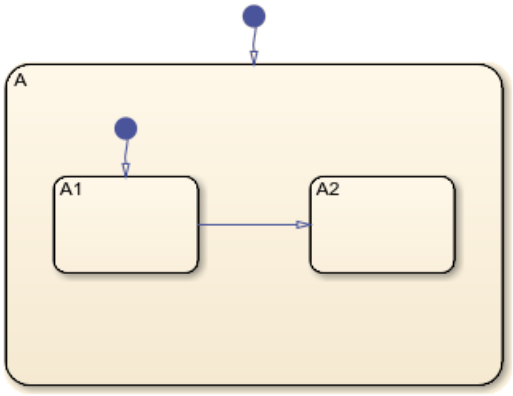
8. Adjust the state properties of the new state.

```
sNew = find(ch,'-isa','Stateflow.State','Name',newName);
sNew.Position = sA.Position + [400 0 0 0];
sNew.IsGrouped = prevGrouping;
```




### Copy and Paste Array of Objects

Copy states `A1` and `A2`, along with the transition between them, to a new state in chart `ch`. To preserve transition connections and containment relationships between objects, copy all the connected objects at once.




1. Find the `Stateflow.State` object named `A` in chart `ch`.

```
sA = find(ch,'-isa','Stateflow.State','Name','A');
```

2. Add a new state called `B`. To enable pasting of other objects inside `B`, convert the new state to a subchart.

```
sB = Stateflow.State(ch);
sB.Name = 'B';
sB.Position = sA.Position + [400 0 0 0];
sB.IsSubchart = true;
```

3. Create an array called `objArray` that contains the states and transitions in state `A`. Use the function `setdiff` to remove state `A` from the array of objects to copy.

```
objArrayS = find(sA,'-isa','Stateflow.State');
objArrayS = setdiff(objArrayS,sA);
objArrayT = find(sA,'-isa','Stateflow.Transition');
objArray = [objArrayS objArrayT];
```

4. Access the clipboard object.

```
cb = sfclipboard;
```

5. Copy the objects in `objArray` and paste them in subchart `B`.

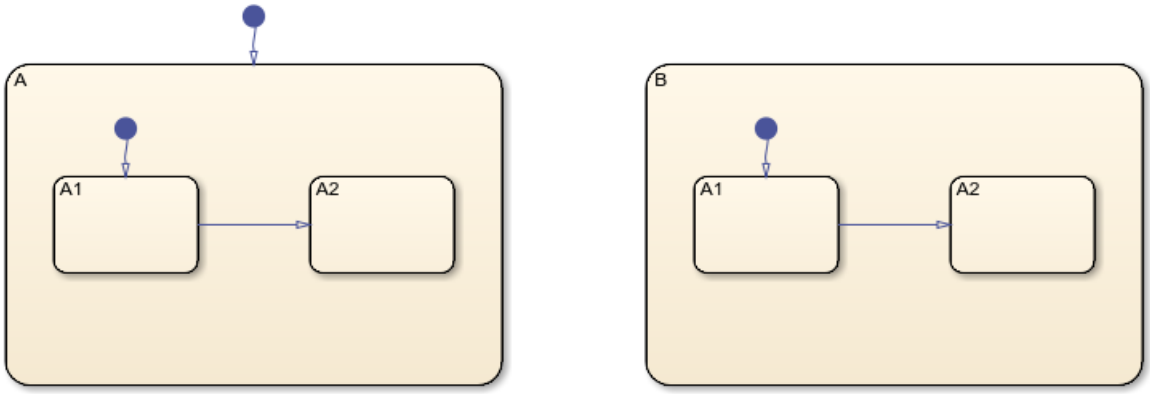
```
copy(cb,objArray);
pasteTo(cb,sB);
```

6. Revert `B` to a state.

```
sB.IsSubchart = false;
sB.IsGrouped = false;
```

7. Reposition the states and transitions in `B`.

```
newStates = find(sB,'-isa','Stateflow.State');
newStates = setdiff(newStates,sB);
newTransitions = find(sB,'-isa','Stateflow.Transition');
newOClocks = get(newTransitions,{'SourceOClock','DestinationOClock'});
for i = 1:numel(newStates)
newStates(i).Position = newStates(i).Position + [25 35 0 0];
end
set(newTransitions,{'SourceOClock','DestinationOClock'},newOClocks);
```




## See Also

**Functions**
`find` | `get` | `set` | `setdiff` | `sfclipboard`

**Objects**
`Stateflow.State`

**Topics**
“Overview of the Stateflow API” on page 1-2
“List of Stateflow API Properties” on page 4-2

**Introduced before R2006a**

# Stateflow.Chart

Graphical representation of a finite state machine

## Description

Use a `Stateflow.Chart` object to create a graphical representation of a finite state machine by combining states, transitions, and data. For more information, see "Finite State Machine Concepts" and "Create Charts by Using the Stateflow API" on page 1-19.

## Creation

To create a `Stateflow.Chart` object, call the function `sfnew`. For example, to create an empty chart in a new Simulink model called `myModel`, enter:

```
sfnew myModel
```

Alternatively, you can add a new chart to an existing model by using the function `add_block`:

```
add_block('sflib/Chart','myModel/Chart')
```

Then, to access the `Stateflow.Chart` object, call the `find` function for the `Simulink.Root` object:

```
rt = sfroot;
chart = find(rt,'-isa','Stateflow.Chart', ...
    'Path','myModel/Chart');
```

### Properties

#### Content

**`Name` — Name of chart**
`'Chart'` (default) | character vector

Name of the chart, specified as a character vector.

**`ActionLanguage` — Action language**
`'MATLAB'` (default) | `'C'`

Action language used to program the chart, specified as `'MATLAB'` or `'C'`. For more information, see "Differences Between MATLAB and C as Action Language Syntax".

**`StateMachineType` — State machine semantics**
`'Classic'` (default) | `'Mealy'` | `'Moore'`

State machine semantics implemented by the chart, specified as `'Classic'`, `'Mealy'`, or `'Moore'`. For more information, see "Overview of Mealy and Moore Machines".

**`SupportVariableSizing` — Whether chart supports variable-size data**
`true` or 1 (default) | `false` or 0

Whether the chart supports variable-size data, specified as a numeric or logical 1 (`true`) or 0 (`false`). Only variable-size data can change dimension during simulation. For more information, see "Declare Variable-Size Data in Stateflow Charts".

### State Decomposition

#### `Decomposition` — Decomposition of substates

`'EXCLUSIVE_OR'` (default) | `'PARALLEL_AND'`

Decomposition of substates at the top level of containment in the chart, specified as `'EXCLUSIVE_OR'` or `'PARALLEL_AND'`. For more information, see "Specify Substate Decomposition".

### Chart Initialization

#### `ExecuteAtInitialization` — Whether to initialize state configuration

`false` or 0 (default) | `true` or 1

Whether to initialize the state configuration of the chart at time zero instead of at the first input event, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see "Execution of a Chart at Initialization".

#### `StatesWhenEnabling` — Behavior of states when event reenables chart

`''` (default) | `'held'` | `'reset'`

Behavior of the states when a function-call input event reenables the chart, specified as one of these values:

- `''` — The chart does not contain function-call input events.
- `'held'` — The chart maintains the most recent values of the states.
- `'reset'` — The chart reverts to the initial conditions of the states.

For more information, see "Control States in Charts Enabled by Function-Call Input Events".

#### `InitializeOutput` — Whether to initialize output data

`false` or 0 (default) | `true` or 1

Whether to initialize the output data every time the chart wakes up, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see "Initialize outputs every time chart wakes up".

### Active State Output

#### `HasOutputData` — Whether to create active state data output

`false` or 0 (default) | `true` or 1

Whether to create an active state data output port for the chart, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see "Monitor State Activity Through Active State Data".

#### `OutputData` — Active state data object

`Stateflow.Data` object

This property is read-only.

Active state data object for the chart, specified as a `Stateflow.Data` object. This property applies only when the `HasOutputData` property for the chart is `true`.

**`OutputPortName` — Name of active state data object**
character vector

Name of the active state data object for the chart, specified as a character vector. This property applies only when the `HasOutputData` property for the chart is `true`.

**`OutputMonitoringMode` — Monitoring mode for active state output**
`'ChildActivity'` (default) | `'LeafStateActivity'`

Monitoring mode for the active state output data, specified as `'ChildActivity'` or `'LeafStateActivity'`.

**`EnumTypeName` — Name of enumerated data type for active state data object**
character vector

Name of the enumerated data type for the active state data object for the chart, specified as a character vector. For more information, see “Enum Name”.

**`DoNotAutogenerateEnum` — Whether to define enumerated data type manually**
`false` or 0 (default) | `true` or 1

Whether to define the enumerated data type for the active state data output manually, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Define State Activity Enumeration Type”.

**Discrete and Continuous-Time Semantics**

**`ChartUpdate` — Activation method for chart**
`'INHERITED'` (default) | `'CONTINUOUS'` | `'DISCRETE'`

Activation method for the chart, specified as `'CONTINUOUS'`, `'DISCRETE'`, or `'INHERITED'`. For more information, see “Update Method”.

**`SampleTime` — Sample time for activating chart**
`'-1'` (default) | character vector

Sample time for activating the chart, specified as a character vector. This property applies only when the `ChartUpdate` property for the chart is `'DISCRETE'`.

**`EnableZeroCrossings` — Whether to enable zero-crossing detection**
`true` or 1 (default) | `false` or 0

Whether to enable zero-crossing detection on state transitions in the chart, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `ChartUpdate` property for the chart is set to `'CONTINUOUS'`. For more information, see “Disable Zero-Crossing Detection”.

**Super Step Semantics**

**`EnableNonTerminalStates` — Whether to enable super step semantics**
`false` or 0 (default) | `true` or 1

Whether to enable super step semantics for the chart, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Super Step Semantics”.

**`NonTerminalMaxCounts` — Maximum number of transitions in one super step**
`1000` (default) | scalar

Maximum number of transitions the chart can take in one super step, specified as an integer scalar. This property applies only when the `EnableNonTerminalStates` property for the chart is `true`.

**`NonTerminalUnstableBehavior` — Behavior if super step exceeds maximum number of transitions**
`'Proceed'` (default) | `'Throw Error'`

Behavior if a super step for the chart exceeds the maximum number of transitions specified in the `NonTerminalMaxCounts` property before reaching a stable state, specified as one of these values:

- `'Proceed'` — The chart goes to sleep with the last active state configuration.
- `'Throw Error'` — The chart generates an error.

This property applies only when the `EnableNonTerminalStates` property for the chart is `true`.

**Exported Functions**

**`ExportChartFunctions` — Whether to export chart-level functions**
`false` or 0 (default) | `true` or 1

Whether to export chart-level functions to other blocks in the Simulink model, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Export Stateflow Functions for Reuse”.

**`AllowGlobalAccessToExportedFunctions` — Whether exported functions are globally visible**
`false` or 0 (default) | `true` or 1

Whether exported functions from the chart are globally visible in the Simulink model, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is enabled, blocks throughout the model can call functions exported from the chart without using qualified notation. This property applies only when the `ExportChartFunctions` property for the chart is `true`.

**Integer and Fixed-Point Data**

**`SaturateOnIntegerOverflow` — Whether data saturates on integer overflow**
`true` or 1 (default) | `false` or 0

Whether the data in the chart saturates on integer overflow, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is disabled, the data in the chart wraps on integer overflow. For more information, see “Handle Integer Overflow for Chart Data”.

**`TreatAsFi` — Inherited Simulink signals to treat as `fi` objects**
`'Fixed-point'` (default) | `'Fixed-point & Integer'`

Inherited Simulink signals to treat as Fixed-Point Designer™ `fi` objects, specified as one of these values:

- `'Fixed-point'` — The chart treats all fixed-point inputs as `fi` objects.
- `'Fixed-point & Integer'` — The chart treats all fixed-point and integer inputs as `fi` objects.

This property applies only to charts that use MATLAB as the action language.

**`EmlDefaultFimath` — Default `fimath` properties**
`'Same as MATLAB Default'` (default) | `'Other:UserSpecified'`

Default `fimath` properties for the chart, specified as one of these values:

- `'Same as MATLAB Default'` — Use the same `fimath` properties as the current default `fimath` object.
- `'Other:UserSpecified'` — Use the `InputFimath` property to specify the default `fimath` object.

This property applies only when the `ActionLanguage` property of the chart is `'MATLAB'`.

**`InputFimath` — Default `fimath` object**
character vector

Default `fimath` object, specified as a character vector. When the `EmlDefaultFimath` property for the chart is `'Other:UserSpecified'`, you can use this property to:

- Enter an expression that constructs a `fimath` object.
- Enter the variable name for a `fimath` object in the MATLAB or model workspace.

This property applies only to charts that use MATLAB as the action language.

**Code Generation**

**`GeneratePreprocessorConditionals` — Whether generated code includes a preprocessor conditional**
`false` or 0 (default) | `true` or 1

Whether the generated code includes a preprocessor conditional statement for the variant conditions in the chart, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when generating code with Embedded Coder®. For more information, see “Code Generation Using Variant Transitions”.

**C Action Language**

**`StrongDataTypingWithSimulink` — Whether to use strong data typing**
`true` or 1 (default) | `false` or 0

Whether to use strong data typing when the chart interfaces with Simulink input and output signals, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only to charts that use C as the action language. For more information, see “Use strong data typing with Simulink I/O”.

**`EnableBitOps` — Whether to use bit operations**
`false` or 0 (default) | `true` or 1

Whether to use bit operations in state and transition actions in the chart, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only to charts that use C as the action language. For more information, see “Enable C-bit operations”.

**`UserSpecifiedStateTransitionExecutionOrder` — Whether to use explicit ordering of parallel states and transitions**
`true` or 1 (default) | `false` or 0

Whether to use explicit ordering of parallel states and transitions, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only to charts that use C as the action language. For more information, see “User-specified state/transition execution order”.

### Debugging

**`Debug.Breakpoints.OnEntry` — Whether to set `On Chart Entry` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `On Chart Entry` breakpoint for the chart, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `chart.Debug.Breakpoints.OnEntry = true;`

### Graphical Appearance

**`Editor` — Editor**
`Stateflow.Editor` object

This property is read-only.

Editor for the chart, specified as a `Stateflow.Editor` object. You can use this object to control the position, size, and magnification level of the Stateflow Editor window.

**`Visible` — Whether editor is displaying chart**
`true` or 1 | `false` or 0

Whether the Stateflow Editor window is displaying the chart, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`ChartColor` — Background color**
`[1 0.9608 0.8824]` (default) | `[red green blue]`

Background color for the chart, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.

**`StateColor` — Color for states**
`[0 0 0]` (default) | `[red green blue]`

Color for the boxes, functions, and states in the chart, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.

**`TransitionColor` — Color for transitions**
`[0.2902 0.3294 0.6039]` (default) | `[red green blue]`

Color for transitions in the chart, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.

**`JunctionColor` — Color for junctions**
`[0.6824 0.3294 0]` (default) | `[red green blue]`

Color for junctions in the chart, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.

**`StateLabelColor` — Color for state labels**
`[0 0 0]` (default) | `[red green blue]`

Color for the box, function, and state labels in the chart, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.

**`StateFont.Angle` — Font angle for state labels**
`'NORMAL'` (default) | `'ITALIC'`

Font angle for the box, function, and state labels in the chart, specified as `'NORMAL'` or `'ITALIC'`.

Example: `chart.StateFont.Angle = 'ITALIC';`

**`StateFont.Weight` — Font weight for state labels**
`'NORMAL'` (default) | `'BOLD'`

Font weight for the box, function, and state labels in the chart, specified as `'NORMAL'` or `'BOLD'`.

Example: `chart.StateFont.Weight = 'BOLD;`

**`StateFont.Size` — Initial font size for state labels**
`12` (default) | scalar

Initial font size for the annotation, box, function, and state labels in the chart, specified as a scalar.

Example: `chart.StateFont.Size = 8;`

**`StateFont.Name` — Font name for state labels**
`'Helvetica'` (default) | character vector

Font name for the annotation, box, function, and state labels in the chart, specified as a character vector.

Example: `chart.StateFont.Name = 'Arial';`

**`TransitionLabelColor` — Color for transition labels**
`[0.2902 0.3294 0.6039]` (default) | `[red green blue]`

Color for the transition labels in the chart, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.

**`TransitionFont.Angle` — Font angle for transition labels**
`'NORMAL'` (default) | `'ITALIC'`

Font angle for the transition labels in the chart, specified as `'NORMAL'` or `'ITALIC'`.

Example: `chart.TransitionFont.Angle = 'ITALIC';`

**`TransitionFont.Weight` — Font weight for the transition labels in this chart**
`'NORMAL'` (default) | `'BOLD'`

Font weight for the transition labels in the chart, specified as `'NORMAL'` or `'BOLD'`.

Example: `chart.TransitionFont.Weight = 'BOLD';`

**`TransitionFont.Size` — Initial font size for transition labels**
`12` (default) | scalar

Initial font size for the transition labels in the chart, specified as a scalar.

Example: `chart.TransitionFont.Size = 8;`

**`TransitionFont.Name` — Font name for transition labels**
`'Helvetica'` (default) | character vector

Font name for the transition labels in the chart, specified as a character vector.

Example: `chart.TransitionFont.Name = 'Arial';`

**Hierarchy**

**`Machine` — Machine that contains chart**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the chart, specified as a `Stateflow.Machine` object.

**`Path` — Location of chart in model hierarchy**
character vector

This property is read-only.

Location of the chart in the model hierarchy, specified as a character vector.

**`Dirty` — Whether chart has changed**
`true` or 1 | `false` or 0

Whether the chart has changed after being opened or saved, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`Locked` — Whether chart is locked**
`false` or 0 (default) | `true` or 1

Whether the chart is locked, specified as a numeric or logical 1 (`true`) or 0 (`false`). Enable this property to prevent changes in the chart.

**`Iced` — Whether chart is locked**
`false` or 0 (default) | `true` or 1

This property is read-only.

Whether the chart is locked, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property is equivalent to the property `Locked`, but is used internally to prevent changes in the chart during simulation.

**Identification**

**`Description` — Description**
`''` (default) | character vector

Description for the chart, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the chart, specified as a character vector.

**Tag — User-defined tag**
[ ] (default) | any data type

User-defined tag for the chart, specified as data of any type.

**Id — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Use this property to distinguish the chart from other objects in the model. The value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| defaultTransitions | Return default transitions in object at top level of containment |
| dialog | Open properties dialog box |
| find | Specified objects in hierarchy |
| fitToView | Zoom in on graphical object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| parse | Parse single chart or all charts in model |
| set | Set properties with specified values |
| view | Display object in editing environment |

## Examples

### Create Empty Stateflow Chart

Call the function `sfnew` to open a new Simulink model that contains an empty Stateflow chart.

```
sfnew
```

Access the `Simulink.Root` object by calling the `sfroot` function.

```
rt = sfroot;
```

Access the `Stateflow.Chart` object by calling the `find` function for the `Simulink.Root` object.

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

## See Also

**Blocks**
Chart

**Functions**
`add_block` | `find` | `sfnew` | `sfroot`

**Topics**
“Overview of the Stateflow API” on page 1-2

“Finite State Machine Concepts”
“Specify Properties for Stateflow Charts”
“Create Charts by Using the Stateflow API” on page 1-19
“List of Stateflow API Properties” on page 4-2

**Introduced before R2006a**

# Stateflow.Data

Data in chart, state, box, or function

## Description

Use `Stateflow.Data` objects to store values that are visible at a specific level of the Stateflow hierarchy. For more information, see “Add Stateflow Data” and “Set Data Properties”.

## Creation

### Syntax

```
data = Stateflow.Data(parent)
```

#### Description

`data = Stateflow.Data(parent)` creates a `Stateflow.Data` object in a parent chart, state, box, or function.

#### Input Arguments

**`parent` — Parent for new data object**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object | ...

Parent for the new data object, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.EMFunction`
- `Stateflow.Function`
- `Stateflow.SimulinkBasedState`
- `Stateflow.SLFunction`
- `Stateflow.State`
- `Stateflow.TruthTable`

### Properties

#### Interface

**`Name` — Name of data object**
`'data'` (default) | character vector

Name of the data object, specified as a character vector.

**Scope — Scope of data object**
`'Local'` (default) | `'Input'` | `'Output'` | `'Constant'` | `'Parameter'` | `'Data Store Memory'` | `'Temporary'` | `'Imported'` | `'Exported'`

Scope of the data object, specified as one of these values:

- `'Local'`
- `'Input'`
- `'Output'`
- `'Constant'`
- `'Parameter'`
- `'Data Store Memory'`
- `'Temporary'`
- `'Imported'`
- `'Exported'`

For more information, see “Scope”.

**`Port` — Port index for data object**
scalar

Port index for the data object, specified as an integer scalar. This property applies only to input and output data. For more information, see “Port”.

**`UpdateMethod` — Method for updating data object**
`'Discrete'` (default) | `'Continuous'`

Method for updating data object, specified as `'Discrete'` or `'Continuous'`. This property applies only when the `ChartUpdate` property of the chart that contains the data is `'CONTINUOUS'`. For more information, see “Continuous-Time Modeling in Stateflow”.

**`InitializeMethod` — Method for initializing data object**
`'Expression'` (default) | `'Parameter'` | `'Not Needed'`

Method for initializing the value of the data object, specified as a character vector that depends on the scope of the data:

- For local and output data, use `'Expression'` or `'Parameter'`.
- For constant data, use `'Expression'`.
- For input data, parameters, and data store memory, use `'Not Needed'`.

This property is equivalent to the **Initial Value** drop-down list in the Model Explorer and the Data properties dialog box. For more information, see “Initial Value”.

**`Props.InitialValue` — Initial value of data object**
`''` (default) | character vector

Initial value of the data object, specified as a character vector. For more information, see “Initial Value”.

Example: `data.Props.InitialValue = '1.5';`

**`Props.Range.Maximum` — Maximum value for data object**
`''` (default) | character vector

Maximum value for the data object, specified as a character vector. For more information, see “Limit Range”.

Example: `data.Props.Range.Maximum = '1024';`

**`Props.Range.Minimum` — Minimum value for data object**
`''` (default) | character vector

Minimum value for the data object, specified as a character vector. For more information, see “Limit Range”.

Example: `data.Props.Range.Minimum = '0';`

**`SaveToWorkspace` — Whether to save data object to workspace variable**
`false` or 0 (default) | `true` or 1

Whether to save the value of the data object to a variable of the same name in the MATLAB base workspace at the end of the simulation, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only to data in charts that use C as the action language. For more information, see “Save Final Value to Base Workspace”.

**`Tunable` — Whether data object is tunable parameter**
`true` or 1 (default) | `false` or 0

Whether the data object is a tunable parameter, specified as a numeric or logical 1 (`true`) or 0 (`false`). Only tunable parameters can be modified during simulation. This property applies only to parameter data.

**`Props.Complexity` — Whether data object accepts complex values**
`'Off'` (default) | `'On'`

Whether the data object accepts complex values, specified as `'On'` or `'Off'`. For more information, see “Complex Data in Stateflow Charts”.

Example: `data.Props.Complexity = 'On';`

**`Props.ResolveToSignalObject` — Whether data object resolves to `Simulink.Signal` object**
`false` or 0 (default) | `true` or 1

Whether the data object resolves to a `Simulink.Signal` object that you define in the model or base workspace, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Resolve Data Properties from Simulink Signal Objects”.

Example: `data.Props.ResolveToSignalObject = true;`

**`Props.Unit.Name` — Unit of measurement for data object**
`'inherit'` (default) | character vector

Unit of measurement for the data object, specified as a character vector. This property applies only to data in charts that use C as the action language. For more information, see “Specify Units for Stateflow Data”.

Example: `data.Props.Unit.Name = 'm';`

### Data Type

#### `DataType` — Type of data object

`'Inherit: From definition in chart'` (default) | `'double'` | `'single'` | `'int32'` | `'uint32'` | `'boolean'` | ...

Type of the data object, specified as a character vector that depends on the `Props.Type.Method` property of the data object:

- If the `Props.Type.Method` property of the data object is `'Inherit'`, the value of this property is `'Inherit: From definition in chart'` for local data and `'Inherit: Same as Simulink'` for input, output, and parameter data.
- If the `Props.Type.Method` property of the data object is `'Built-in'`, you can specify this property with one of these options:
  - `'double'`
  - `'single'`
  - `'int8'`
  - `'int16'`
  - `'int32'`
  - `'int64'`
  - `'uint8'`
  - `'uint16'`
  - `'uint32'`
  - `'uint64'`
  - `'boolean'`
  - `'ml'` (Supported only in charts that use C as the action language)
  - `'string'` (Supported only in charts that use C as the action language)
- Otherwise, the `Props.Type` properties of the data object determine the value of this property.

For more information, see the section Add Data on page 1-0 in "Create Charts by Using the Stateflow API" on page 1-19.

#### `Props.Type.Method` — Method for setting data type

`'Inherited'` (default) | `'Built-in'` | `'Bus Object'` | `'Enumerated'` | `'Expression'` | `'Fixed point'`

Method for setting the data type, specified as a character vector that depends on the scope of the data:

- For local, input, output, or parameter data, use `'Inherited'`, `'Built-in'`, `'Bus Object'`, `'Enumerated'`, `'Expression'`, or `'Fixed point'`.
- For constant data, use `'Built-in'`, `'Expression'`, or `'Fixed point'`.
- For data store memory data, use `'Inherited'`.

This property is equivalent to the **Mode** field of the Data Type Assistant in the Model Explorer and the Data properties dialog box. For more information, see "Specify Type of Stateflow Data".

Example: `data.Props.Method = 'Built-in';`

**`Props.Type.BusObject` — Name of `Simulink.Bus` object**
`''` (default) | character vector

Name of the `Simulink.Bus` object that defines the data object, specified as a character vector. This property applies only when the `Props.Type.Method` property of the data object is `'Bus Object'`. For more information, see “Access Bus Signals Through Stateflow Structures”.

Example: `data.Props.Type.BusObject = 'COUNTERBUS';`

**`Props.Type.EnumType` — Name of enumerated type**
`''` (default) | character vector

Name of the enumerated type that defines the data object, specified as a character vector. This property applies only when the `Props.Type.Method` property of the data object is `'Enumerated'`. For more information, see “Reference Values by Name by Using Enumerated Data”.

Example: `data.Props.Type.EnumType = 'BasicColors';`

**`Props.Type.Expression` — Expression that evaluates to data type**
`''` (default) | character vector

Expression that evaluates to the data type of the data object, specified as a character vector. This property applies only when the `Props.Type.Method` property of the data object is `'Expression'`. For more information, see “Specify Data Properties by Using MATLAB Expressions”.

Example: `data.Props.Type.Expression = 'type(y)';`

**`Props.Type.Signed` — Signedness of fixed-point data**
`true` or 1 (default) | `false` or 0

Signedness of the fixed-point data object, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `Props.Type.Method` property of the data object is `'Fixed point'`. For more information, see “Fixed-Point Data in Stateflow Charts”.

Example: `data.Props.Type.Signed = false;`

**`Props.Type.WordLength` — Word length of fixed-point data**
`'16'` (default) | character vector

Word length, in bits, of the fixed-point data object, specified as a character vector. This property applies only when the `Props.Type.Method` property of the data object is `'Fixed point'`. For more information, see “Fixed-Point Data in Stateflow Charts”.

Example: `data.Props.Type.WordLength = '32';`

**`Props.Type.Fixpt.ScalingMode` — Method for scaling fixed-point data**
`'None'` (default) | `'Binary point'` | `'Slope and bias'`

Method for scaling the fixed-point data object, specified as `'Binary point'`, `'Slope and bias'`, or `'None'`. This property applies only when the `Props.Type.Method` property of the data object is `'Fixed point'`. For more information, see “Fixed-Point Data in Stateflow Charts”.

Example: `data.Props.Type.Fixpt.ScalingMode = 'Binary point';`

**`Props.Type.Fixpt.FractionLength` — Fraction length of fixed-point data**
`''` (default) | character vector

Fraction length, in bits, of the fixed-point data object, specified as a character vector. This property applies only to fixed-point data when the `Props.Type.Fixpt.ScalingMode` property is `'Binary point'`. For more information, see “Fixed-Point Data in Stateflow Charts”.

Example: `data.Props.Type.Fixpt.FractionLength = '2';`

#### `Props.Type.Fixpt.Slope` — Slope of fixed-point data

`''` (default) | character vector

Slope of the fixed-point data object, specified as a character vector. This property applies only to fixed-point data when the `Props.Type.Fixpt.ScalingMode` property is `'Slope and bias'`. For more information, see “Fixed-Point Data in Stateflow Charts”.

Example: `data.Props.Type.Fixpt.Slope = '2^-2';`

#### `Props.Type.Fixpt.Bias` — Bias of fixed-point data

`''` (default) | character vector

Bias of the fixed-point data object, specified as a character vector. This property applies only to fixed-point data when the `Props.Type.Fixpt.ScalingMode` property is `'Slope and bias'`. For more information, see “Fixed-Point Data in Stateflow Charts”.

Example: `data.Props.Type.Fixpt.Bias = '0';`

#### `Props.Type.Fixpt.Lock` — Whether to prevent replacement of fixed-point type

`false` or 0 (default) | `true` or 1

Whether to prevent replacement of the fixed-point type of the data object with an autoscaled type chosen by the Fixed-Point Tool (Fixed-Point Designer), specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Autoscaling Using the Fixed-Point Tool” (Fixed-Point Designer).

Example: `data.Props.Type.Fixpt.Lock = true;`

#### `CompiledType` — Data type as determined by compiler

`'unknown'` (default) | character vector

This property is read-only.

Data type as determined by the compiler, specified as a character vector.

### Data Size

#### `Props.Array.Size` — Size of data object

`'-1'` (default) | character vector

Size of the data object, specified as a character vector. For more information, see “Specify Size of Stateflow Data”.

#### `Props.Array.IsDynamic` — Whether data object has variable size

`false` or 0 (default) | `true` or 1

Whether the data object has variable size, specified as a numeric or logical 1 (`true`) or 0 (`false`). Only variable-size data can change size during simulation. This property applies only to input and output data and is equivalent to the **Variable Size** check box in the Data properties dialog box. For more information, see “Declare Variable-Size Data in Stateflow Charts”.

**`Props.Array.FirstIndex` — Index for first element of array**
character vector

Index for the first element of the array data object, specified as a character vector. This property applies only to array data in charts that use C as the action language. For more information, see "Save Final Value to Base Workspace".

**`CompiledSize` — Data size as determined by compiler**
`''` (default) | character vector

This property is read-only.

Data size as determined by the compiler, specified as a character vector.

**Active State Output**

**`OutputState` — State or chart monitored by data object**
`[]` (default) | `Stateflow.AtomicSubchart` object | `Stateflow.Chart` | `Stateflow.SimulinkBasedState` object | `Stateflow.State` object | `Stateflow.StateTransitionTableChart` object

This property is read-only.

State or chart monitored by the data object, specified as an empty array or a `Stateflow.AtomicSubchart`, `Stateflow.Chart`, `Stateflow.SimulinkBasedState`, `Stateflow.State`, or `Stateflow.StateTransitionTableChart` object. For more information, see "Monitor State Activity Through Active State Data".

**Signal Logging**

**`LoggingInfo.DataLogging` — Whether to enable signal logging for data object**
`false` or 0 (default) | `true` or 1

Whether to enable signal logging for the data object, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see "Log Simulation Output for States and Data".

Example: `data.LoggingInfo.DataLogging = true;`

**`LoggingInfo.DecimateData` — Whether to limit logged data**
`false` or 0 (default) | `true` or 1

Whether to limit the amount of logged data, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is `true`, signal logging skips samples by using the interval size specified by the `LoggingInfo.Decimation` property. For more information, see "Decimation".

Example: `data.LoggingInfo.DeimateData = true;`

**`LoggingInfo.Decimation` — Decimation interval**
2 (default) | scalar

Decimation interval, specified as an integer scalar. The default value of 2 means that the chart logs every other sample. For more information, see "Decimation".

Example: `data.LoggingInfo.Decimation = 5;`

**`LoggingInfo.LimitDataPoints` — Whether to limit number of data points to log**
`false` or 0 (default) | `true` or 1

Whether to limit the number of data points to log, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is `true`, signal logging limits the number of data points by using the value specified by the `LoggingInfo.MaxPoints` property. For more information, see “Limit Data Points to Last”.

Example: `data.LoggingInfo.LimitDataPoints = true;`

**`LoggingInfo.MaxPoints` — Maximum number of data points to log**
`5000` (default) | scalar

Maximum number of data points to log, specified as an integer scalar. The default value of `5000` means the chart logs the last 5000 data points generated by the simulation. For more information, see “Limit Data Points to Last”.

Example: `data.LoggingInfo.MaxPoints = 100;`

**`LoggingInfo.NameMode` — Source of signal name**
`'SignalName'` (default) | `'Custom'`

Source of the signal name used to log the data object, specified as one of these values:

- `'SignalName'` — Use the name of the data object.
- `'Custom'` — Use the custom signal name specified by the `LoggingInfo.LoggingName` property.

For more information, see “Logging Name”.

Example: `data.LoggingInfo.NameMode = 'Custom';`

**`LoggingInfo.LoggingName` — Custom signal name**
character vector

Custom signal name used for logging the data object, specified as a character vector. This property applies only when the `LoggingInfo.NameMode` property is `'Custom'`. For more information, see “Logging Name”.

Example: `data.LoggingInfo.LoggingName = 'Data';`

**Debugging**

**`Debug.Watch` — Whether to track data object**
`false` or 0 (default) | `true` or 1

Whether to track the value of the data object in the Breakpoints and Watch window, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “View Data in the Breakpoints and Watch Window”.

**`TestPoint` — Whether to set data object as test point**
`false` or 0 (default) | `true` or 1

Whether to set the data object as a test point, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Monitor Test Points in Stateflow Charts”.

**Hierarchy**

**`Machine` — Machine that contains data object**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the data object, specified as a `Stateflow.Machine` object.

**`Path` — Location of parent in model hierarchy**
character vector

This property is read-only.

Location of the parent of the data object in the model hierarchy, specified as a character vector.

**Identification**

**`Description` — Description**
`''` (default) | character vector

Description for the data object, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the data object, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the data object, specified as data of any type.

**`SSIdNumber` — Session-independent identifier**
scalar

This property is read-only.

Session-independent identifier, specified as an integer scalar. Use this property to distinguish the data object from other objects in the model.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Unlike `SSIdNumber`, the value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| get | Return MATLAB structure containing property settings of object or array of objects |
| set | Set properties with specified values |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add Data to Chart

Add a data object to the chart `ch`. Specify its name, scope, and data type.

```
data = Stateflow.Data(ch);
data.Name = 'x';
data.Scope = 'Input';
data.Props.Type.Method = 'Built-in';
data.DataType = 'single';
```

## See Also

`Stateflow.Box` | `Stateflow.Chart` | `Stateflow.EMFunction` | `Stateflow.Function` | `Stateflow.SLFunction` | `Stateflow.SimulinkBasedState` | `Stateflow.State` | `Stateflow.TruthTable`

**Topics**

“Overview of the Stateflow API” on page 1-2
“Add Stateflow Data”
“Set Data Properties”
“List of Stateflow API Properties” on page 4-2

**Introduced before R2006a**

# Stateflow.Editor

Graphical aspects of a chart or state transition table

## Description

Use the `Stateflow.Editor` object to access the graphical aspects of a Stateflow chart or state transition table. You can use the `Stateflow.Editor` object to control the position, size, and magnification level of the Stateflow Editor window.

## Creation

Each chart has its own `Stateflow.Editor` object. When you create a chart, an `Stateflow.Editor` object is automatically created for it. To access the `Stateflow.Editor` object, use the `Editor` property for the chart. For example, if `ch` is a `Stateflow.Chart` or `Stateflow.StateTransitionTableChart` object, enter:

```
editor = ch.Editor;
```

### Properties

**`WindowPosition` — Position and size of window**
`[left top width height]`

Position and size of the Stateflow editor window, specified as a four-element numeric vector of the form `[left top width height]`.

**`ZoomFactor` — Magnification level**
scalar

Magnification level of the chart or state transition table in the editor, specified as a scalar value between `0.5` and `10`. A value of `1` corresponds to a magnification of 100%.

### Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| set | Set properties with specified values |
| zoomIn | Zoom in on Stateflow chart |
| zoomOut | Zoom out on Stateflow chart |

### Examples

#### Zoom in on Stateflow Chart

Increase the magnification level of a nonempty chart `ch`.

```
editor = ch.Editor;
zoomIn(editor)
```

If the magnification level for the chart was initially 100%, this command increases it to 130%.

#### Zoom out on Stateflow Chart

Decrease the magnification level of a nonempty chart `ch`.

```
editor = ch.Editor;
zoomOut(editor)
```

If the magnification level for the chart was initially 100%, this command decreases it to 76.9%.

#### Set Zoom Factor

Set the `ZoomFactor` property for a nonempty chart `ch` to an absolute magnification level of 150%.

```
editor = ch.Editor;
editor.ZoomFactor = 1.5;
```

## See Also

`Stateflow.Chart` | `Stateflow.StateTransitionTableChart`

#### Topics

“Overview of the Stateflow API” on page 1-2
“List of Stateflow API Properties” on page 4-2

#### Introduced before R2006a

# Stateflow.EMChart

Stateflow interface to MATLAB Function block

## Description

Use `Stateflow.EMChart` objects to configure MATLAB Function blocks through the Stateflow programmatic interface.

MATLAB Function blocks define custom functionality in Simulink models. Use these blocks when:

- You have an existing MATLAB function that models custom functionality, or it would be easy for you to create such a function.
- Your model requires custom functionality that is not or cannot be captured in the Simulink graphical language.
- You find it easier to model custom functionality by using a MATLAB function than by using a Simulink block diagram.
- The custom functionality that you want to model does not include continuous or discrete dynamic states. To model dynamic states, use S-functions.

For more information, see “Implementing MATLAB Functions Using Blocks” (Simulink) and “Create Custom Functionality Using MATLAB Function Block” (Simulink).

---

**Tip** You can also configure the properties of a MATLAB Function block programmatically by using a `MATLABFunctionConfiguration` object. This object provides a direct interface to the properties of a MATLAB Function block. For more information, see “Configure MATLAB Function Block Programmatically” (Simulink).

---

## Creation

Each MATLAB Function block has its own `Stateflow.EMChart` object. When you add a MATLAB Function block to a Simulink model, a `Stateflow.EMChart` object is automatically created for it. For example, you can use the function `add_block`:

```
add_block('simulink/User-Defined Functions/MATLAB Function', ...
   'myModel/MATLAB Function')
```

Then, to access the `Stateflow.EMChart` object, call the `find` function for the `Simulink.Root` object:

```
rt = sfroot;
block = find(rt,'-isa','Stateflow.EMChart', ...
    'Path','myModel/MATLAB Function')
```

## Properties

### Content

#### `Name` — Name of MATLAB Function block

`'MATLAB Function'` (default) | character vector

Name of the MATLAB Function block, specified as a character vector.

#### `Script` — Code for MATLAB Function block

character vector

Code for the MATLAB Function block, specified as a character vector. To enter multiple lines of code, you can:

- Call the MATLAB function `sprintf` and use `\n` to insert newline characters:

  ```
  str = sprintf('function y=f(x)\ny=x+1;\nend');
  block.Script = str;
  ```

- Enter a concatenated text expression that uses the integer 10 as the ASCII equivalent of a newline character:

  ```
  str = ['function y=f(x)',10, ...
      'y=x+1;',10, ...
      'end'];
  block.Script = str;
  ```

#### `SupportVariableSizing` — Whether MATLAB Function block supports variable-size data

`true` or 1 (default) | `false` or 0

Whether the MATLAB Function block supports variable-size data, specified as a numeric or logical 1 (`true`) or 0 (`false`). Only variable-size data can change dimension during simulation. For more information, see “Declare Variable-Size Inputs and Outputs” (Simulink).

#### `AllowDirectFeedthrough` — Whether MATLAB Function block supports direct feedthrough semantics

`true` or 1 (default) | `false` or 0

Whether the MATLAB Function block supports direct feedthrough semantics, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Allow direct feedthrough” (Simulink).

### Interface

#### `Inputs` — Input arguments

array of `Stateflow.Data` objects

This property is read-only.

Input arguments of the MATLAB Function block, specified as an array of `Stateflow.Data` objects. The value of this property depends on the inputs defined in the `Script` property for the block.

#### `Outputs` — Output arguments

array of `Stateflow.Data` objects

This property is read-only.

Output arguments of the MATLAB Function block, specified as an array of `Stateflow.Data` objects. The value of this property depends on the outputs defined in the `Script` property for the block.

**Discrete and Continuous-Time Semantics**

**`ChartUpdate` — Activation method for MATLAB Function block**
`'INHERITED'` (default) | `'CONTINUOUS'` | `'DISCRETE'`

Activation method for the MATLAB Function block, specified as `'CONTINUOUS'`, `'DISCRETE'`, or `'INHERITED'`. For more information, see “Update method” (Simulink).

**`SampleTime` — Sample time for activating MATLAB Function block**
`'-1'` (default) | character vector

Sample time for activating the MATLAB Function block, specified as a character vector. This property applies only when the `ChartUpdate` property for the MATLAB function is `'DISCRETE'`.

**Integer and Fixed-Point Data**

**`SaturateOnIntegerOverflow` — Whether data saturates on integer overflow**
`true` or 1 (default) | `false` or 0

Whether the data in the MATLAB Function block saturates on integer overflow, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is disabled, the data in the function wraps on integer overflow. For more information, see “Saturate on integer overflow” (Simulink).

**`TreatAsFi` — Inherited Simulink signals to treat as `fi` objects**
`'Fixed-point'` (default) | `'Fixed-point & Integer'`

Inherited Simulink signals to treat as Fixed-Point Designer `fi` objects, specified as one of these values:

- `'Fixed-point'` — The MATLAB Function block treats all fixed-point inputs as `fi` objects.
- `'Fixed-point & Integer'` — The MATLAB Function block treats all fixed-point and integer inputs as `fi` objects.

**`EmlDefaultFimath` — Default `fimath` properties**
`'Same as MATLAB Default'` (default) | `'Other:UserSpecified'`

Default `fimath` properties for the MATLAB Function block, specified as one of these values:

- `'Same as MATLAB Default'` — Use the same `fimath` properties as the current default `fimath` object.
- `'Other:UserSpecified'` — Use the `InputFimath` property to specify the default `fimath` object.

**`InputFimath` — Default `fimath` object**
character vector

Default `fimath` object, specified as a character vector. When the `EmlDefaultFimath` property for the MATLAB Function block is `'Other:UserSpecified'`, you can use this property to:

- Enter an expression that constructs a `fimath` object.
- Enter the variable name for a `fimath` object in the MATLAB or model workspace.

### Hierarchy

**`Machine` — Machine that contains MATLAB Function block**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the MATLAB Function block, specified as a `Stateflow.Machine` object.

**`Path` — Location of MATLAB Function block in model hierarchy**
character vector

This property is read-only.

Location of the MATLAB Function block in the model hierarchy, specified as a character vector.

**`Dirty` — Whether MATLAB Function block has changed**
`true` or 1 | `false` or 0

Whether the MATLAB Function block has changed after being opened or saved, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`Locked` — Whether MATLAB Function block is locked**
`false` or 0 (default) | `true` or 1

Whether the MATLAB Function block is locked, specified as a numeric or logical 1 (`true`) or 0 (`false`). Enable this property to prevent changes in the MATLAB Function block.

**`Iced` — Whether MATLAB Function block is locked**
`false` or 0 (default) | `true` or 1

This property is read-only.

Whether the MATLAB Function block is locked, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property is equivalent to the property `Locked`, but is used internally to prevent changes in the MATLAB Function block during simulation.

### Identification

**`Description` — Description**
`''` (default) | character vector

Description for the MATLAB Function block, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the MATLAB Function block, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the MATLAB Function block, specified as data of any type.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Use this property to distinguish the MATLAB Function block from other objects in the model. The value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| find | Specified objects in hierarchy |
| get | Return MATLAB structure containing property settings of object or array of objects |
| parse | Parse single chart or all charts in model |
| set | Set properties with specified values |
| view | Display object in editing environment |

## Examples

### Program MATLAB Function Block

Access the `Stateflow.EMChart` object for a MATLAB Function block.

```
block = find(sfroot,'-isa','Stateflow.EMChart');
```

Store the MATLAB code to calculate the mean and standard deviation for a vector of values as a character vector.

```
str = ['function [mean,stdev] = stats(vals)',10, ...
'% Calculates a statistical mean and a standard',10, ...
'% deviation for the values in vals.',10,10, ...
'len = length(vals);',10, ...
'mean = avg(vals,len);',10, ...
'stdev = sqrt(sum(((vals-avg(vals,len)).^2))/len);',10, ...
'plot(vals,''-+'');',10,10,...
'function mean = avg(array,size)',10, ...
'mean = sum(array)/size;'];
```

Populate the block with code by modifying the `Script` property of the corresponding `Stateflow.EMChart` object.

```
block.Script = str;
```

Open the function in the MATLAB Function Block Editor.

```
view(block)
```

The editor shows this code.

```
function [mean,stdev] = stats(vals)
% Calculates a statistical mean and a standard
% deviation for the values in vals.

len = length(vals);
mean = avg(vals,len);
```

```
stdev = sqrt(sum(((vals-avg(vals,len)).^2))/len);
plot(vals,'-+');

function mean = avg(array,size)
mean = sum(array)/size;
```

#### Import Code from MATLAB Function

Open a Simulink model called `myModel`.

```
open_system('myModel')
```

Add a MATLAB Function block to `myModel`.

```
blockPath = 'myModel/My Function';
add_block('simulink/User-Defined Functions/MATLAB Function',blockPath)
```

Populate the block with code from the MATLAB function `myFunction.m`.

```
block = find(sfroot,'-isa','Stateflow.EMChart', ...
    'Path',newBlockPath);
block.Script = fileread('myFunction.m');
```

#### Find Number of MATLAB Function Blocks in Model

Open a Simulink model called `myModel`.

```
open_system('myModel')
```

Access the `Simulink.Root` object at the top level of the Stateflow hierarchy.

```
rt = sfroot;
```

Find the MATLAB Function blocks in the model.

```
blocks = find(rt,'-isa','Stateflow.EMChart');
```

Count the number of blocks.

```
numel(blocks);
```

## See Also

**Blocks**
MATLAB Function

**Functions**
`add_block` | `fileread` | `find` | `numel` | `sfroot`

**Objects**
`MATLABFunctionConfiguration`

**Topics**
“Overview of the Stateflow API” on page 1-2

“Implementing MATLAB Functions Using Blocks” (Simulink)
“Create Custom Functionality Using MATLAB Function Block” (Simulink)
“MATLAB Function Block Properties” (Simulink)
“List of Stateflow API Properties” on page 4-2

**Introduced in R2011a**

# Stateflow.EMFunction

MATLAB function in chart, state, box, or function

## Description

Use `Stateflow.EMFunction` objects to create MATLAB functions for coding algorithms that are more easily expressed by using MATLAB code instead of the graphical Stateflow constructs. Typical applications include:

- Matrix-oriented calculations
- Data analysis and visualization

You can call a MATLAB function in the actions of states and transitions. For more information, see “Reuse MATLAB Code by Defining MATLAB Functions”.

## Creation

### Syntax

```
function = Stateflow.EMFunction(parent)
```

#### Description

`function = Stateflow.EMFunction(parent)` creates a `Stateflow.EMFunction` object in a parent chart, state, box, or function.

#### Input Arguments

**`parent` — Parent for new MATLAB function**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

Parent for the new MATLAB function, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.Function`
- `Stateflow.State`

### Properties

#### Content

**`Name` — Name of MATLAB function**
`''` (default) | character vector

Name of the MATLAB function, specified as a character vector.

### `LabelString` — Full label for MATLAB function

`'?'` (default) | character vector

Full label for the MATLAB function, specified as a character vector.

### `Script` — Code for MATLAB function

character vector

Code for the MATLAB function, specified as a character vector. To enter multiple lines of code, you can:

- Call the MATLAB function `sprintf` and use the escape sequence `\n` to insert newline characters:

```
str = sprintf('function y=f(x)\ny=x+1;\nend');
function.Script = str;
```

- Enter a concatenated text expression that uses the integer 10 as the ASCII equivalent of a newline character:

```
str = ['function y=f(x)',10, ...
    'y=x+1;',10, ...
    'end'];
function.Script = str;
```

### `IsExplicitlyCommented` — Whether to comment out MATLAB function

`false` or 0 (default) | `true` or 1

Whether to comment out the MATLAB function, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the MATLAB function and selecting **Comment Out**. For more information, see “Commenting Stateflow Objects in a Chart”.

### `IsImplicitlyCommented` — Whether MATLAB function is implicitly commented out

`true` or 1 | `false` or 0

This property is read-only.

Whether the MATLAB function is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The MATLAB function is implicitly commented out when you comment out a superstate in its hierarchy.

### `CommentText` — Comment text

`''` (default) | character vector

Comment text for the MATLAB function, specified as a character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge  on the MATLAB function, the text appears as a tooltip. When you set the `IsExplicitlyCommented` property to `false`, the value of `CommentText` reverts to `''`.

### Graphical Appearance

### `Position` — Position and size of MATLAB function

`[0 0 90 60]` (default) | `[left top width height]`

Position and size of the MATLAB function, specified as a four-element numeric vector of the form `[left top width height]`.

**`BadIntersection` — Whether function intersects a box, state, or function**
`true` or 1 | `false` or 0

This property is read-only.

Whether the MATLAB function graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`FontSize` — Font size for MATLAB function label**
scalar

Font size for the MATLAB function label, specified as a scalar. The `StateFont.Size` property of the chart that contains the graphical function sets the initial value of this property.

**Integer and Fixed-Point Data**

**`SaturateOnIntegerOverflow` — Whether data saturates on integer overflow**
`true` or 1 (default) | `false` or 0

Whether the data in the MATLAB function saturates on integer overflow, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is disabled, the data in the function wraps on integer overflow. For more information, see “Handle Integer Overflow for Chart Data”.

This property applies only when the `ActionLanguage` of the chart that contains the function is `'C'`. Otherwise, the behavior of data depends on the value of the `SaturateOnIntegerOverflow` property for the chart.

**`EmlDefaultFimath` — Default `fimath` properties**
`'Same as MATLAB Default'` (default) | `'Other:UserSpecified'`

Default `fimath` properties for the MATLAB function, specified as one of these values:

- `'Same as MATLAB Default'` — Use the same `fimath` properties as the current default `fimath` object.
- `'Other:UserSpecified'` — Use the `InputFimath` property to specify the default `fimath` object.

This property applies only when the `ActionLanguage` of the chart that contains the function is `'C'`. Otherwise, the behavior of data depends on the value of the `EmlDefaultFimath` property for the chart.

**`InputFimath` — Default `fimath` object**
character vector

Default `fimath` object, specified as a character vector. When the `EmlDefaultFimath` property for the MATLAB function is `'Other:UserSpecified'`, you can use this property to:

- Enter an expression that constructs a `fimath` object.
- Enter the variable name for a `fimath` object in the MATLAB or model workspace.

This property applies only when the `ActionLanguage` of the chart that contains the function is `'C'`. Otherwise, the behavior of data depends on the value of the `InputFimath` property for the chart.

### Code Generation

#### `InlineOption` — Appearance in generated code

`'Auto'` (default) | `'Function'` | `'Inline'`

Appearance of the MATLAB function in generated code, specified as one of these values:

- `'Auto'` — An internal calculation determines the appearance of the function in generated code.
- `'Function'` — The function is implemented as a separate C function.
- `'Inline'` — Calls to the function are replaced by code.

For more information, see “Inline State Functions in Generated Code” (Simulink Coder).

### Hierarchy

#### `Chart` — Chart that contains MATLAB function

`Stateflow.Chart` object

This property is read-only.

Chart that contains the MATLAB function, specified as a `Stateflow.Chart` object.

#### `Subviewer` — Subviewer for MATLAB function

`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

This property is read-only.

Subviewer for the MATLAB function, specified as a `Stateflow.Chart`, `Stateflow.State`, `Stateflow.Box`, or `Stateflow.Function` object. The subviewer is the chart or subchart where you can graphically view the MATLAB function.

#### `Machine` — Machine that contains MATLAB function

`Stateflow.Machine` object

This property is read-only.

Machine that contains the MATLAB function, specified as a `Stateflow.Machine` object.

#### `Path` — Location of parent in model hierarchy

character vector

This property is read-only.

Location of the parent of the MATLAB function in the model hierarchy, specified as a character vector.

### Identification

#### `Description` — Description

`''` (default) | character vector

Description for the MATLAB function, specified as a character vector.

#### `Document` — Document link

`''` (default) | character vector

Document link for the MATLAB function, specified as a character vector.

**`Tag` — User-defined tag**
[ ] (default) | any data type

User-defined tag for the MATLAB function, specified as data of any type.

**`SSIdNumber` — Session-independent identifier**
scalar

This property is read-only.

Session-independent identifier, specified as an integer scalar. Use this property to distinguish the MATLAB function from other objects in the model.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Unlike `SSIdNumber`, the value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| find | Specified objects in hierarchy |
| fitToView | Zoom in on graphical object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| highlight | Highlight graphical object |
| isCommented | Determine if object is commented out |
| set | Set properties with specified values |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add MATLAB Function to Chart

Add a MATLAB function in the chart `ch`. Set its label to `'[y1,y2] = f(x1,x2,x3)'`.

```
function = Stateflow.EMFunction(ch);
function.LabelString = '[y1,y2] = f(x1,x2,x3)';
```

## See Also

`Stateflow.Box` | `Stateflow.Chart` | `Stateflow.Function` | `Stateflow.State`

**Topics**
“Overview of the Stateflow API” on page 1-2
“Reuse MATLAB Code by Defining MATLAB Functions”
“List of Stateflow API Properties” on page 4-2

**Introduced before R2006a**

# Stateflow.Event

Event in chart, state, or box

## Description

Use `Stateflow.Event` objects to trigger actions in one of these objects:

- A parallel state in a Stateflow chart
- Another Stateflow chart
- A Simulink triggered or function-call subsystem

For more information, see “Synchronize Model Components by Broadcasting Events”.

## Creation

### Syntax

```
event = Stateflow.Event(parent)
```

#### Description

`event = Stateflow.Event(parent)` creates a `Stateflow.Event` object in a parent chart, state, or box.

#### Input Arguments

**`parent` — Parent for new event**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object

Parent for the new event, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.State`

### Properties

#### Interface

**`Name` — Name of event**
`'event'` (default) | character vector

Name of the event, specified as a character vector.

**`Scope` — Scope of event**
`'Local'` (default) | `'Input'` | `'Output'`

Scope of the event, specified as `'Local'`, `'Input'`, or `'Output'`. For more information, see "Scope".

**`Trigger` — Type of trigger**
`'Function call'` (default) | `'Rising'` | `'Falling'` | `'Either'`

Type of trigger associated with the event, specified as a character vector that depends on the scope of the data:

- For input events, use `'Function call'`, `'Rising'`, `'Falling'`, or `'Either'`.
- For output events, use `'Function call'` or `'Either'`.

This property does not apply to local events. For more information, see "Trigger".

**`Port` — Port index for event**
scalar

Port index for the event, specified as an integer scalar. This property applies only to input and output events. For more information, see "Port".

**Debugging**

**`Debug.Breakpoints.StartBroadcast` — Whether to set `Start of Broadcast` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `Start of Broadcast` breakpoint for the event, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see "Debugger Breakpoints".

Example: `event.Debug.Breakpoints.StartBroadcast = true;`

**`Debug.Breakpoints.EndBroadcast` — Whether to set `End of Broadcast` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `End of Broadcast` breakpoint for the event, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see "Debugger Breakpoints".

Example: `event.Debug.Breakpoints.EndBroadcast = true;`

**Hierarchy**

**`Machine` — Machine that contains event**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the event, specified as a `Stateflow.Machine` object.

**`Path` — Location of parent in model hierarchy**
character vector

This property is read-only.

Location of the parent of the event in the model hierarchy, specified as a character vector.

### Identification

**`Description` — Description**
`''` (default) | character vector

Description for the event, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the event, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the event, specified as data of any type.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Use this property to distinguish the event from other objects in the model. The value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| get | Return MATLAB structure containing property settings of object or array of objects |
| set | Set properties with specified values |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add Event to Chart

Add a event to the chart `ch`. Specify its name and scope.

```
event = Stateflow.Event(ch);
event.Name = 'E';
event.Scope = 'Input';
```

## See Also

`Stateflow.Box` | `Stateflow.Chart` | `Stateflow.State`

### Topics

“Overview of the Stateflow API” on page 1-2
“Synchronize Model Components by Broadcasting Events”
“Set Properties for an Event”
“List of Stateflow API Properties” on page 4-2

**Introduced before R2006a**

# Stateflow.Function

Graphical function in chart, state, box, or function

## Description

Use `Stateflow.Function` objects to create graphical functions that contain control-flow logic and iterative loops. You create graphical functions with flow charts that use connective junctions and transitions. You can call a graphical function in the actions of states and transitions. For more information, see “Reuse Logic Patterns by Defining Graphical Functions”.

## Creation

### Syntax

```
function = Stateflow.Function(parent)
```

#### Description

`function = Stateflow.Function(parent)` creates a `Stateflow.Function` object in a parent chart, state, box, or function.

#### Input Arguments

**`parent` — Parent for new graphical function**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

Parent for the new graphical function, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.Function`
- `Stateflow.State`

### Properties

#### Content

**`Name` — Name of graphical function**
`''` (default) | character vector

Name of the graphical function, specified as a character vector.

**`LabelString` — Full label for graphical function**
`'?'` (default) | character vector

Full label for the graphical function, specified as a character vector.

**`IsExplicitlyCommented` — Whether to comment out graphical function**
`false` or 0 (default) | `true` or 1

Whether to comment out the graphical function, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the graphical function and selecting **Comment Out**. For more information, see "Commenting Stateflow Objects in a Chart".

**`IsImplicitlyCommented` — Whether graphical function is implicitly commented out**
`true` or 1 | `false` or 0

This property is read-only.

Whether the graphical function is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The graphical function is implicitly commented out when you comment out a superstate in its hierarchy.

**`CommentText` — Comment text**
`''` (default) | character vector

Comment text for the graphical function, specified as a character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge  on the graphical function, the text appears as a tooltip. When you set the `IsExplicitlyCommented` property to `false`, the value of `CommentText` reverts to `''`.

**Graphical Appearance**

**`Position` — Position and size of graphical function**
`[0 0 90 60]` (default) | `[left top width height]`

Position and size of the graphical function, specified as a four-element numeric vector of the form `[left top width height]`.

**`BadIntersection` — Whether function intersects a box, state, or function**
`true` or 1 | `false` or 0

This property is read-only.

Whether the graphical function graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`IsGrouped` — Whether function is a grouped function**
`false` or 0 (default) | `true` or 1

Whether the function is a grouped function, specified as a numeric or logical 1 (`true`) or 0 (`false`). When you copy and paste a grouped function, you copy not only the function but all of its contents. For more information, see "Copy and Paste by Grouping" on page 2-24.

**`IsSubchart` — Whether function is a subchart**
`false` or 0 (default) | `true` or 1

Whether the function is a subchart, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`ContentPreviewEnabled` — Whether to display preview of function contents**
`false` or 0 (default) | `true` or 1

Whether to display a preview of the graphical function contents, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `IsSubchart` property is `true`.

**`FontSize` — Font size for graphical function label**
scalar

Font size for the graphical function label, specified as a scalar. The `StateFont.Size` property of the chart that contains the graphical function sets the initial value of this property.

**Debugging**

**`Debug.Breakpoints.OnDuring` — Whether to set `During Function Call` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `During Function Call` breakpoint for the graphical function, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `function.Debug.Breakpoints.OnDuring = true;`

**Code Generation**

**`InlineOption` — Appearance in generated code**
`'Auto'` (default) | `'Function'` | `'Inline'`

Appearance of the graphical function in generated code, specified as one of these values:

- `'Auto'` — An internal calculation determines the appearance of the function in generated code.
- `'Function'` — The function is implemented as a separate C function.
- `'Inline'` — Calls to the function are replaced by code.

For more information, see “Inline State Functions in Generated Code” (Simulink Coder).

**Hierarchy**

**`Chart` — Chart that contains graphical function**
`Stateflow.Chart` object

This property is read-only.

Chart that contains the graphical function, specified as a `Stateflow.Chart` object.

**`Subviewer` — Subviewer for graphical function**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

This property is read-only.

Subviewer for the graphical function, specified as a `Stateflow.Chart`, `Stateflow.State`, `Stateflow.Box`, or `Stateflow.Function` object. The subviewer is the chart or subchart where you can graphically view the graphical function.

**`Machine` — Machine that contains graphical function**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the graphical function, specified as a `Stateflow.Machine` object.

**`Path` — Location of parent in model hierarchy**
character vector

This property is read-only.

Location of the parent of the graphical function in the model hierarchy, specified as a character vector.

**Identification**

**`Description` — Description**
`''` (default) | character vector

Description for the graphical function, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the graphical function, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the graphical function, specified as data of any type.

**`SSIdNumber` — Session-independent identifier**
scalar

This property is read-only.

Session-independent identifier, specified as an integer scalar. Use this property to distinguish the graphical function from other objects in the model.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Unlike `SSIdNumber`, the value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| defaultTransitions | Return default transitions in object at top level of containment |
| dialog | Open properties dialog box |
| find | Specified objects in hierarchy |
| fitToView | Zoom in on graphical object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| highlight | Highlight graphical object |
| isCommented | Determine if object is commented out |

| | |
|---|---|
| set | Set properties with specified values |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add Graphical Function to Chart

Add a graphical function in the chart `ch`. Set its label to `'[y1,y2] = f(x1,x2,x3)'`.

```
function = Stateflow.Function(ch);
function.LabelString = '[y1,y2] = f(x1,x2,x3)';
```

## See Also

`Stateflow.Box` | `Stateflow.Chart` | `Stateflow.State`

### Topics

“Overview of the Stateflow API” on page 1-2
“Reuse Logic Patterns by Defining Graphical Functions”
“List of Stateflow API Properties” on page 4-2

### Introduced before R2006a

# Stateflow.Junction

Junction in chart, state, box, or function

## Description

Use `Stateflow.Junction` objects to create junctions that:

- Represent decision points in a transition path
- Record the activity of substates inside a superstate

For more information, see “Combine Transitions and Junctions to Create Branching Paths” and “Record State Activity by Using History Junctions”.

## Creation

### Syntax

```
junction = Stateflow.Junction(parent)
```

#### Description

`junction = Stateflow.Junction(parent)` creates a `Stateflow.Junction` object in a parent chart, state, box, or graphical function.

#### Input Arguments

**`parent` — Parent for new junction**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

Parent for the new junction, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.Function`
- `Stateflow.State`

### Properties

#### Content

**Type — Type of junction**
`'CONNECTIVE'` (default) | `'HISTORY'`

Type of junction, specified as one of these values:

- `'CONNECTIVE'` — Connective junction that represents a decision point in a transition path
- `'HISTORY'` — History junction that records the activity of substates inside a superstate

**`IsExplicitlyCommented` — Whether to comment out junction**
`false` or 0 (default) | `true` or 1

Whether to comment out the junction, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the junction and selecting **Comment Out**. For more information, see “Commenting Stateflow Objects in a Chart”.

**`IsImplicitlyCommented` — Whether junction is implicitly commented out**
`true` or 1 | `false` or 0

This property is read-only.

Whether the junction is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The junction is implicitly commented out when you comment out a superstate in its hierarchy.

**`CommentText` — Comment text**
`''` (default) | character vector

Comment text added to the junction, specified as a character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge  on the junction, the text appears as a tooltip. When you set the `IsExplicitlyCommented` property to `false`, the value of `CommentText` reverts to `''`.

**Graphical Appearance**

**`Position.Center` — Position of center of junction**
`[7 7]` (default) | `[x y]`

Position of the center of the junction, specified as a two-element numeric vector `[x y]` of coordinates relative to the upper left corner of the chart.

Example: `junction.Position.Center = [31.41 27.18];`

**`Position.Radius` — Radius of junction**
7 (default) | scalar

Radius of the junction, specified as a scalar.

Example: `junction.Position.Radius = 16.18;`

**`ArrowSize` — Size of incoming transition arrows**
8 (default) | scalar

Size of incoming transition arrows, specified as a scalar.

**Hierarchy**

**`Chart` — Chart that contains junction**
`Stateflow.Chart` object

This property is read-only.

Chart that contains the junction, specified as a `Stateflow.Chart` object.

**`Subviewer` — Subviewer for junction**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

This property is read-only.

Subviewer for the junction, specified as a `Stateflow.Chart`, `Stateflow.State`, `Stateflow.Box`, or `Stateflow.Function` object. The subviewer is the chart or subchart where you can graphically view the junction.

**`Machine` — Machine that contains junction**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the junction, specified as a `Stateflow.Machine` object.

**`Path` — Location of parent in model hierarchy**
character vector

This property is read-only.

Location of the parent of the junction in the model hierarchy, specified as a character vector.

**Identification**

**`Description` — Description**
`''` (default) | character vector

Description for the junction, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the junction, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the junction, specified as data of any type.

**`SSIdNumber` — Session-independent identifier**
scalar

This property is read-only.

Session-independent identifier, specified as an integer scalar. Use this property to distinguish the junction from other objects in the model.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Unlike `SSIdNumber`, the value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| fitToView | Zoom in on graphical object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| highlight | Highlight graphical object |
| isCommented | Determine if object is commented out |
| set | Set properties with specified values |
| sinkedTransitions | Return transitions that have object as destination |
| sourcedTransitions | Return transitions that have object as source |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add Connective Junction to Chart

Add a connective junction in the chart `ch`. Change its size and position.

```
junction = Stateflow.Junction(ch);
junction.Position.Radius = 16.18;
junction.Position.Center = [31.41 27.18];
```

### Add History Junction to Chart

Add a history junction in the chart `ch`.

```
junction = Stateflow.Junction(ch);
junction.Type = 'HISTORY';
```

## See Also

`Stateflow.AtomicBox` | `Stateflow.AtomicSubchart` | `Stateflow.Box` | `Stateflow.Chart` | `Stateflow.Function` | `Stateflow.SimulinkBasedState` | `Stateflow.State`

### Topics

“Overview of the Stateflow API” on page 1-2
“Combine Transitions and Junctions to Create Branching Paths”
“Record State Activity by Using History Junctions”
“List of Stateflow API Properties” on page 4-2

### Introduced before R2006a

# Stateflow.Machine

Container for Stateflow blocks in a Simulink model

## Description

From a Stateflow perspective, `Stateflow.Machine` objects are equivalent to Simulink models. A `Stateflow.Machine` object contains `Stateflow.Chart`, `Stateflow.StateTransitionTableChart`, `Stateflow.TruthTableChart`, and `Stateflow.EMChart` objects that represent the Stateflow charts, State Transition Table blocks, Truth Table blocks, and MATLAB Function blocks in a Simulink model. For more information, see “Overview of the Stateflow API” on page 1-2.

## Creation

You automatically create a `Stateflow.Machine` object when you load a model that contains a Stateflow block or call the function `sfnew`. To access the `Stateflow.Machine` object, call the `find` function for the `Simulink.Root` object. For example, if your Simulink model is named `myModel`, enter:

```
rt = sfroot;
machine = find(rt,'-isa','Stateflow.Machine','Name','myModel');
```

### Properties

#### Content

**`Name` — Name of Simulink model**
character vector

This property is read-only.

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

**`FullFileName` — Full file path of Simulink model**
character vector

This property is read-only.

Full file path of the Simulink model for the machine, specified as a character vector.

**`IsLibrary` — Whether model builds library**
`false` or 0 (default) | `true` or 1

This property is read-only.

Whether the Simulink model for the machine builds a library and not an application, specified as a numeric or logical 1 (`true`) or 0 (`false`).

### Debugging

#### `Debug.Animation.Enabled` — Whether to animate charts during simulation

`true` or 1 (default) | `false` or 0

Whether to animate the charts in the machine during simulation, specified as a numeric or logical 1 (`true`) or 0 (`false`). Disabling this property is equivalent to selecting `None` in the **Animation Speed** drop-down list in the **Debug** tab.

Example: `machine.Debug.Animation.Enabled = false;`

#### `Debug.Animation.Delay` — Delay for highlighting transitions

0 (default) | scalar

Delay that the chart animation uses for highlighting each transition segment in the machine, specified as a scalar. These values correspond to the settings of the **Animation Speed** drop-down list in the **Debug** tab:

| Delay Value | Animation Speed |
|---|---|
| 0.5 | `Slow` |
| 0.2 | `Medium` |
| 0 | `Fast` |
| -1 | `Lightning Fast` |

This property applies only when the `Debug.Animation.Enable` property of the machine is `true`.

Example: `machine.Debug.Animation.Delay = 1;`

#### `Debug.Animation.MaintainHighlighting` — Whether to maintain highlighting of active states

`false` or 0 (default) | `true` or 1

This property is read-only.

Whether to maintain the highlighting of active states in the machine after the simulation ends, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `machine.Debug.Animation.MaintainHighlighting = true;`

### Hierarchy

#### `Path` — Location of machine in model hierarchy

character vector

This property is read-only.

Location of the machine in the model hierarchy, specified as a character vector.

#### `Dirty` — Whether model has changed

`true` or 1 | `false` or 0

Whether the Simulink model for the machine has changed after being opened or saved, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`Locked` — Whether machine is locked**
`false` or 0 (default) | `true` or 1

Whether the machine is locked, specified as a numeric or logical 1 (`true`) or 0 (`false`). Enable this property to prevent changes in the Stateflow charts, state transition tables, and truth table blocks in this machine.

**`Iced` — Whether machine is locked**
`false` or 0 (default) | `true` or 1

This property is read-only.

Whether the machine is locked, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property is equivalent to the property `Locked`, but is used internally to prevent changes in the machine during simulation.

**Identification**

**`Created` — Date of creation**
character vector

This property is read-only.

Date of the creation of the machine, specified as a character vector.

**`Creator` — Creator**
`'Unknown'` (default) | character vector

Creator of the machine, specified as a character vector.

**`Modified` — Record of modifications**
`''` (default) | character vector

Record of modifications to the machine, specified as a character vector.

**`Version` — Version**
`'none'` (default) | character vector

Version of the machine, specified as a character vector.

**`Description` — Description**
`''` (default) | character vector

Description for the machine, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the machine, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the machine, specified as data of any type.

### `Id` — Unique identifier

scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Use this property to distinguish the machine from other objects in the model. The value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| find | Specified objects in hierarchy |
| get | Return MATLAB structure containing property settings of object or array of objects |
| parse | Parse single chart or all charts in model |
| set | Set properties with specified values |

## Examples

### Update Machine Version

Update the `Modified` and `Version` properties of machine machine.

```
machine.Modified = string(datetime);
oldVersion = str2num(machine.Version);
if isempty(oldVersion)
    machine.Version = '1';
else
    machine.Version = num2str(oldVersion+1);
end
```

## See Also

### Functions

`find` | `sfroot`

### Objects

`Stateflow.Chart` | `Stateflow.StateTransitionTableChart` | `Stateflow.TruthTableChart`

### Topics

“Overview of the Stateflow API” on page 1-2
“Machine Properties”
“List of Stateflow API Properties” on page 4-2

### Introduced before R2006a

# Stateflow.Message

Message in chart, state, or box

## Description

Use `Stateflow.Message` objects to communicate data locally or between Stateflow charts in Simulink models. For more information, see “Communicate with Stateflow Charts by Sending Messages”.

## Creation

### Syntax

```
message = Stateflow.Message(parent)
```

#### Description

`message = Stateflow.Message(parent)` creates a `Stateflow.Message` object in a parent chart, state, or box.

#### Input Arguments

**`parent` — Parent for new message**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object

Parent for the new message, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.State`

### Properties

#### Interface

**`Name` — Name of message**
`'message'` (default) | character vector

Name of the message, specified as a character vector.

**Scope — Scope of message**
`'Output'` (default) | `'Input'` | `'Local'`

Scope of the message, specified as specified as `'Local'`, `'Input'`, or `'Output'`. For more information, see “Scope”.

**`Port` — Port index for message**
scalar

Port index for the message, specified as an integer scalar. This property applies only to input and output messages. For more information, see “Port”.

**`InitializeMethod` — Method for initializing message data**
`'Expression'` (default) | `'Parameter'` | `'Not Needed'`

Method for initializing the value of the message data, specified as a character vector that depends on the scope of the message:

- For local and output messages, use `'Expression'` or `'Parameter'`.
- For input messages, use `'Not Needed'`.

For more information, see “Initial Value”.

**`Props.InitialValue` — Initial value of message data**
`''` (default) | character vector

Initial value of the message data, specified as a character vector. For more information, see “Initial Value”.

Example: `message.Props.InitialValue = '1.5';`

**`Props.Complexity` — Whether message data accepts complex values**
`'Off'` (default) | `'On'`

Whether the message data accepts complex values, specified as `'On'` or `'Off'`. For more information, see “Complex Data in Stateflow Charts”.

Example: `message.Props.Complexity = 'On';`

**`Priority` — Priority**
`'300'` (default) | character vector

Priority for the message, specified as a character vector. If two distinct messages occur at the same time, this property determines which message is processed first. A smaller numeric value indicates a higher priority. This property applies only to local and output messages in discrete-event charts. For more information, see “Create Custom Queuing Systems Using Discrete-Event Stateflow Charts” (SimEvents).

**Queue**

**`UseInternalQueue` — Whether chart maintains internal queue for message**
`true` or 1 (default) | `false` or 0

Whether the Stateflow chart maintains an internal receiving queue for the input message, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only to input messages. For more information, see “Use Internal Queue”.

**`QueueType` — Order in which messages are removed from queue**
`'FIFO'` (default) | `'LIFO'` | `'Priority'`

Order in which messages are removed from the receiving queue, specified as one of these values:

- `'FIFO'` — First in, first out.
- `'LIFO'` — Last in, first out.

- `'Priority'` — Remove messages according to the value in the data field. To specify the order, use the `MessagePriorityOrder` property for the message.

This property applies only to local messages and to input messages that have `UseInternalQueue` set to `true`. For more information, see “Queue Type”.

**`MessagePriorityOrder` — Type of priority queue**
`'Ascending'` (default) | `'Descending'`

Type of priority queue for the message, specified as one of these values:

- `'Ascending'` — Messages are received in ascending order of the message data value.
- `'Descending'` — Messages are received in descending order of the message data value.

This property applies only when the `QueueType` property of the message is `'Priority'`. For more information, see “Queue Type”.

**`QueueCapacity` — Length of internal queue**
`10` (default) | scalar

Length of the internal queue for the message, specified as an integer scalar. This property applies only to local messages and to input messages that have `UseInternalQueue` set to `true`. For more information, see “Queue Capacity”.

**`QueueOverflowDiagnostic` — Level of diagnostic when number of messages exceeds queue capacity**
`'Error'` (default) | `'Warning'` | `'None'`

Level of diagnostic action when the number of incoming messages exceeds the queue capacity for the message, specified as `'Error'`, `'Warning'`, or `'None'`. This property applies only to local messages and to input messages that have `UseInternalQueue` set to `true`. For more information, see “Queue Overflow Diagnostic”.

**Data Type**

**`DataType` — Data type of message**
`'Inherit: Same as Simulink'` (default) | `'double'` | `'single'` | `'int32'` | `'uint32'` | `'boolean'` | ...

Data type of the message, specified as a character vector that depends on the `Props.Type.Method` property of the message:

- If the `Props.Type.Method` property of the message is `'Inherit'`, the value of this property is `'Inherit: Same as Simulink'`.
- If the `Props.Type.Method` property of the message is `'Built-in'`, you can specify this property with one of these options:
  - `'double'`
  - `'single'`
  - `'int8'`
  - `'int16'`
  - `'int32'`

- `'int64'`
- `'uint8'`
- `'uint16'`
- `'uint32'`
- `'uint64'`
- `'boolean'`
- `'ml'` (Supported only in charts that use C as the action language)
- `'string'` (Supported only in charts that use C as the action language)

- Otherwise, the `Props.Type` properties of the message determine the value of this property.

For more information, see the section Add Data on page 1-0 in "Create Charts by Using the Stateflow API" on page 1-19.

**`Props.Type.Method` — Method for setting data type**
`'Inherited'` (default) | `'Built-in'` | `'Bus Object'` | `'Enumerated'` | `'Expression'` | `'Fixed point'`

Method for setting the data type of the message, specified as `'Inherited'`, `'Built-in'`, `'Bus Object'`, `'Enumerated'`, `'Expression'`, or `'Fixed point'`.

This property is equivalent to the **Mode** field of the Data Type Assistant in the Message properties dialog box. For more information, see "Specify Type of Stateflow Data".

Example: `message.Props.Method = 'Built-in';`

**`Props.Type.BusObject` — Name of `Simulink.Bus` object**
`''` (default) | character vector

Name of the `Simulink.Bus` object that defines the message data, specified as a character vector. This property applies only when the `Props.Type.Method` property of the message is `'Bus Object'`. For more information, see "Access Bus Signals Through Stateflow Structures".

Example: `message.Props.Type.BusObject = 'COUNTERBUS';`

**`Props.Type.EnumType` — Name of enumerated type**
`''` (default) | character vector

Name of the enumerated type that defines the message data, specified as a character vector. This property applies only when the `Props.Type.Method` property of the message is `'Enumerated'`. For more information, see "Reference Values by Name by Using Enumerated Data".

Example: `message.Props.Type.EnumType = 'BasicColors';`

**`Props.Type.Expression` — Expression that evaluates to data type**
`''` (default) | character vector

Expression that evaluates to the data type of the message data, specified as a character vector. This property applies only when the `Props.Type.Method` property of the message is `'Expression'`. For more information, see "Specify Data Properties by Using MATLAB Expressions".

Example: `message.Props.Type.Expression = 'type(y)';`

**`Props.Type.Signed` — Signedness of fixed-point data**
`true` or 1 (default) | `false` or 0

Signedness of the fixed-point message data, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `Props.Type.Method` property of the message is `'Fixed point'`. For more information, see “Fixed-Point Data in Stateflow Charts”.

Example: `message.Props.Type.Signed = false;`

**`Props.Type.WordLength` — Word length of fixed-point data**
`'16'` (default) | character vector

Word length, in bits, of the fixed-point message data, specified as a character vector. This property applies only when the `Props.Type.Method` property of the message is `'Fixed point'`. For more information, see “Fixed-Point Data in Stateflow Charts”.

Example: `message.Props.Type.WordLength = '32';`

**`Props.Type.Fixpt.ScalingMode` — Method for scaling fixed-point data**
`'None'` (default) | `'Binary point'` | `'Slope and bias'`

Method for scaling the fixed-point message data, specified as `'Binary point'`, `'Slope and bias'`, or `'None'`. This property applies only when the `Props.Type.Method` property of the message is `'Fixed point'`. For more information, see “Fixed-Point Data in Stateflow Charts”.

Example: `message.Props.Type.Fixpt.ScalingMode = 'Binary point';`

**`Props.Type.Fixpt.FractionLength` — Fraction length of fixed-point data**
`''` (default) | character vector

Fraction length, in bits, of the fixed-point message data, specified as a character vector. This property applies only to fixed-point message data when the `Props.Type.Fixpt.ScalingMode` property is `'Binary point'`. For more information, see “Fixed-Point Data in Stateflow Charts”.

Example: `message.Props.Type.Fixpt.FractionLength = '2';`

**`Props.Type.Fixpt.Slope` — Slope of fixed-point data**
`''` (default) | character vector

Slope of the fixed-point message data, specified as a character vector. This property applies only to fixed-point message data when the `Props.Type.Fixpt.ScalingMode` property is `'Slope and bias'`. For more information, see “Fixed-Point Data in Stateflow Charts”.

Example: `message.Props.Type.Fixpt.Slope = '2^-2';`

**`Props.Type.Fixpt.Bias` — Bias of fixed-point data**
`''` (default) | character vector

Bias of the fixed-point message data, specified as a character vector. This property applies only to fixed-point message data when the `Props.Type.Fixpt.ScalingMode` property is `'Slope and bias'`. For more information, see “Fixed-Point Data in Stateflow Charts”.

Example: `message.Props.Type.Fixpt.Bias = '0';`

**`Props.Type.Fixpt.Lock` — Whether to prevent replacement of fixed-point type**
`false` or 0 (default) | `true` or 1

Whether to prevent replacement of the fixed-point type of the message data with an autoscaled type chosen by the Fixed-Point Tool (Fixed-Point Designer), specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Autoscaling Using the Fixed-Point Tool” (Fixed-Point Designer).

Example: `message.Props.Type.Fixpt.Lock = true;`

**`CompiledType` — Data type as determined by compiler**
`'unknown'` (default) | character vector

This property is read-only.

Data type as determined by the compiler, specified as a character vector.

**Data Size**

**`Props.Array.Size` — Size of message data**
`'-1'` (default) | character vector

Size of the message data, specified as a character vector. For more information, see “Specify Size of Stateflow Data”.

**`CompiledSize` — Message data size as determined by compiler**
`''` (default) | character vector

This property is read-only.

Message data size as determined by the compiler, specified as a character vector.

**Hierarchy**

**`Machine` — Machine that contains message**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the message, specified as a `Stateflow.Machine` object.

**`Path` — Location of parent in model hierarchy**
character vector

This property is read-only.

Location of the parent of the message in the model hierarchy, specified as a character vector.

**Identification**

**`Description` — Description**
`''` (default) | character vector

Description for the message, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the message, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the message, specified as data of any type.

### `SSIdNumber` — Session-independent identifier

scalar

This property is read-only.

Session-independent identifier, specified as an integer scalar. Use this property to distinguish the message from other objects in the model.

### `Id` — Unique identifier

scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Unlike `SSIdNumber`, the value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| get | Return MATLAB structure containing property settings of object or array of objects |
| set | Set properties with specified values |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add Message to Chart

Add a message to the chart `ch`. Specify its name, scope, and data type.

```
message = Stateflow.Message(ch);
message.Name = 'M';
message.Scope = 'Input';
message.Props.Type.Method = 'Built-in';
message.DataType = 'int32';
```

## See Also

`Stateflow.Box` | `Stateflow.Chart` | `Stateflow.State`

### Topics

“Overview of the Stateflow API” on page 1-2
“Communicate with Stateflow Charts by Sending Messages”
“Set Properties for a Message”
“List of Stateflow API Properties” on page 4-2

### Introduced in R2015b

# Stateflow.SimulinkBasedState

Simulink based state in chart, state, or box

## Description

Use `Stateflow.SimulinkBasedState` objects to create Simulink subsystems within a Stateflow state. With Simulink based states, you can model hybrid dynamic systems or systems that switch between periodic and continuous time dynamics. For more information, see “Simulink Subsystems as States”.

## Creation

### Syntax

```
simulinkBasedState = Stateflow.SimulinkBasedState(parent)
```

#### Description

`simulinkBasedState = Stateflow.SimulinkBasedState(parent)` creates a `Stateflow.SimulinkBasedState` object in a parent chart, state, or box.

#### Input Arguments

**`parent` — Parent for new Simulink based state**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object

Parent for the new Simulink based state, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.State`

### Properties

#### Content

**`Name` — Name of Simulink based state**
`''` (default) | character vector

Name of the Simulink based state, specified as a character vector.

**`IsExplicitlyCommented` — Whether to comment out Simulink based state**
`false` or 0 (default) | `true` or 1

Whether to comment out the Simulink based state, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the Simulink based state and selecting **Comment Out**. For more information, see “Commenting Stateflow Objects in a Chart”.

**`IsImplicitlyCommented` — Whether Simulink based state is implicitly commented out**
`true` or 1 | `false` or 0

This property is read-only.

Whether the Simulink based state is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The Simulink based state is implicitly commented out when you comment out a superstate in its hierarchy.

**`CommentText` — Comment text**
`''` (default) | character vector

Comment text added to the Simulink based state, specified as a character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge  on the Simulink based state, the text appears as a tooltip. When you set the `IsExplicitlyCommented` property to `false`, the value of `CommentText` reverts to `''`.

**Graphical Appearance**

**`Position` — Position and size of Simulink based state**
`[0 0 90 60]` (default) | `[left top width height]`

Position and size of the Simulink based state, specified as a four-element numeric vector of the form `[left top width height]`.

**`BadIntersection` — Whether Simulink based state intersects a box, state, or function**
`true` or 1 | `false` or 0

This property is read-only.

Whether the Simulink based state graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`ContentPreviewEnabled` — Whether to display preview of Simulink based state contents**
`true` or 1 (default) | `false` or 0

Whether to display a preview of the Simulink based state contents, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`ArrowSize` — Size of incoming transition arrows**
8 (default) | scalar

Size of incoming transition arrows, specified as a scalar.

**`FontSize` — Font size for Simulink based state label**
scalar

Font size for the Simulink based state label, specified as a scalar. The `StateFont.Size` property of the chart that contains the Simulink based state sets the initial value of this property.

**State Decomposition**

**`Type` — Decomposition of sibling states**
`'AND'` | `'OR'`

This property is read-only.

Decomposition of sibling states, specified as `'OR'` or `'AND'`. The Simulink based state inherits this property from the `Decomposition` property of its parent state or chart.

**`ExecutionOrder` — Execution order in parallel (AND) decomposition**
scalar

Execution order for the Simulink based state in parallel (AND) decomposition, specified as an integer scalar. This property applies only when both of these conditions are satisfied:

- The `Type` property of the Simulink based state is `'AND'`.
- The `UserSpecifiedStateTransitionExecutionOrder` property of the chart that contains the Simulink based state is `true`.

**Active State Output**

**`HasOutputData` — Whether to create active state data output**
`false` or 0 (default) | `true` or 1

Whether to create an active state data output port for the Simulink based state, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Monitor State Activity Through Active State Data”.

**`OutputData` — Active state data object**
`Stateflow.Data` object

This property is read-only.

Active state data object for the Simulink based state, specified as a `Stateflow.Data` object. This property applies only when the `HasOutputData` property for the Simulink based state is `true`.

**`OutputPortName` — Name of active state data object**
character vector

Name of the active state data object for the Simulink based state, specified as a character vector. This property applies only when the `HasOutputData` property for the Simulink based state is `true`.

**`OutputMonitoringMode` — Monitoring mode for active state output**
`'SelfActivity'`

Monitoring mode for the active state output data, specified as a character vector. For Simulink based states, the only option is `'SelfActivity'`.

**Signal Logging**

**`LoggingInfo.DataLogging` — Whether to enable signal logging for Simulink based state**
`false` or 0 (default) | `true` or 1

Whether to enable signal logging for the Simulink based state, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Log Simulation Output for States and Data”.

Example: `simulinkBasedState.LoggingInfo.DataLogging = true;`

**`LoggingInfo.DecimateData` — Whether to limit logged data**
`false` or 0 (default) | `true` or 1

Whether to limit the amount of logged data, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is `true`, signal logging skips samples by using the interval size specified by the `LoggingInfo.Decimation` property.

Example: `simulinkBasedState.LoggingInfo.DeimateData = true;`

**`LoggingInfo.Decimation` — Decimation interval**
2 (default) | scalar

Decimation interval, specified as an integer scalar. The default value of 2 means that the chart logs every other sample.

Example: `simulinkBasedState.LoggingInfo.Decimation = 5;`

**`LoggingInfo.LimitDataPoints` — Whether to limit number of data points to log**
`false` or 0 (default) | `true` or 1

Whether to limit the number of data points to log, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is `true`, signal logging limits the number of data points by using the value specified by the `LoggingInfo.MaxPoints` property.

Example: `simulinkBasedState.LoggingInfo.LimitDataPoints = true;`

**`LoggingInfo.MaxPoints` — Maximum number of data points to log**
`5000` (default) | scalar

Maximum number of data points to log, specified as an integer scalar. The default value of `5000` means the chart logs the last 5000 data points generated by the simulation.

Example: `simulinkBasedState.LoggingInfo.MaxPoints = 100;`

**`LoggingInfo.NameMode` — Source of signal name**
`'SignalName'` (default) | `'Custom'`

Source of the signal name used to log the Simulink based state, specified as one of these values:

- `'SignalName'` — Use the name of the Simulink based state.
- `'Custom'` — Use the custom signal name specified by the `LoggingInfo.LoggingName` property.

Example: `simulinkBasedState.LoggingInfo.NameMode = 'Custom';`

**`LoggingInfo.LoggingName` — Custom signal name**
character vector

Custom signal name used for logging the Simulink based state, specified as a character vector. This property applies only when the `LoggingInfo.NameMode` property is `'Custom'`.

Example: `simulinkBasedState.LoggingInfo.LoggingName = 'State';`

**Debugging**

**`Debug.Breakpoints.OnDuring` — Whether to set `During State` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `During State` breakpoint for the Simulink based state, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `simulinkBasedState.Debug.Breakpoints.OnDuring = true;`

**`Debug.Breakpoints.OnEntry` — Whether to set `On State Entry` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `On State Entry` breakpoint for the Simulink based state, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `simulinkBasedState.Debug.Breakpoints.OnEntry = true;`

**`Debug.Breakpoints.OnExit` — Whether to set `On State Exit` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `On State Exit` breakpoint for the Simulink based state, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `simulinkBasedState.Debug.Breakpoints.OnExit = true;`

**`TestPoint` — Whether to set Simulink based state as test point**
`false` or 0 (default) | `true` or 1

Whether to set the Simulink based state as a test point, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Monitor Test Points in Stateflow Charts”.

**Hierarchy**

**`Chart` — Chart that contains Simulink based state**
`Stateflow.Chart` object

This property is read-only.

Chart that contains the Simulink based state, specified as a `Stateflow.Chart` object.

**`Subviewer` — Subviewer for Simulink based state**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object

This property is read-only.

Subviewer for the Simulink based state, specified as a `Stateflow.Chart`, `Stateflow.State`, or `Stateflow.Box` object. The subviewer is the chart or subchart where you can graphically view the Simulink based state.

**`Machine` — Machine that contains Simulink based state**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the Simulink based state, specified as a `Stateflow.Machine` object.

**`Path` — Location of parent in model hierarchy**
character vector

This property is read-only.

Location of the parent of the Simulink based state in the model hierarchy, specified as a character vector.

### Identification

**`Description` — Description**
`''` (default) | character vector

Description for the Simulink based state, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the Simulink based state, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the Simulink based state, specified as data of any type.

**`SSIdNumber` — Session-independent identifier**
scalar

This property is read-only.

Session-independent identifier, specified as an integer scalar. Use this property to distinguish the Simulink based state from other objects in the model.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Unlike `SSIdNumber`, the value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| find | Specified objects in hierarchy |
| fitToView | Zoom in on graphical object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| highlight | Highlight graphical object |
| isCommented | Determine if object is commented out |
| set | Set properties with specified values |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add Simulink Based State to Chart

Add a Simulink based state in the chart `ch`. Set its name to `'A'`.

```
simulinkBasedState = Stateflow.SimulinkBasedState(ch);
simulinkBasedState.Name = 'A';
```

## See Also

`Stateflow.Box` | `Stateflow.Chart` | `Stateflow.State`

**Topics**

“Overview of the Stateflow API” on page 1-2
“Simulink Subsystems as States”
“List of Stateflow API Properties” on page 4-2

**Introduced in R2017b**

# Stateflow.SLFunction

Simulink function in chart, state, box, or function

## Description

Use `Stateflow.SLFunction` objects to create Simulink functions that enable you to call Simulink subsystems in the actions of states and transitions. Typical applications include:

- Defining a function that requires Simulink blocks
- Scheduling execution of multiple controllers

For more information, see “Reuse Simulink Components in Stateflow Charts”.

## Creation

### Syntax

```
function = Stateflow.SLFunction(parent)
```

#### Description

`function = Stateflow.SLFunction(parent)` creates a `Stateflow.SLFunction` object in a parent chart, state, box, or function.

#### Input Arguments

**`parent` — Parent for new Simulink function**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

Parent for the new Simulink function, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.Function`
- `Stateflow.State`

### Properties

#### Content

**`Name` — Name of Simulink function**
`''` (default) | character vector

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

**`LabelString` — Full label for Simulink function**
`'?'` (default) | character vector

Full label for the Simulink function, specified as a character vector.

**`IsExplicitlyCommented` — Whether to comment out Simulink function**
`false` or 0 (default) | `true` or 1

Whether to comment out the Simulink function, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the Simulink function and selecting **Comment Out**. For more information, see "Commenting Stateflow Objects in a Chart".

**`IsImplicitlyCommented` — Whether Simulink function is implicitly commented out**
`true` or 1 | `false` or 0

This property is read-only.

Whether the Simulink function is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The Simulink function is implicitly commented out when you comment out a superstate in its hierarchy.

**`CommentText` — Comment text**
`''` (default) | character vector

Comment text added to the Simulink function, specified as a character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge  on the Simulink function, the text appears as a tooltip. When you set the `IsExplicitlyCommented` property to `false`, the value of `CommentText` reverts to `''`.

**Graphical Appearance**

**`Position` — Position and size of Simulink function**
`[0 0 90 60]` (default) | `[left top width height]`

Position and size of the Simulink function, specified as a four-element numeric vector of the form `[left top width height]`.

**`BadIntersection` — Whether function intersects a box, state, or function**
`true` or 1 | `false` or 0

This property is read-only.

Whether the Simulink function graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`ContentPreviewEnabled` — Whether to display preview of Simulink function contents**
`true` or 1 (default) | `false` or 0

Whether to display a preview of the Simulink function contents, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`FontSize` — Font size for Simulink function label**
scalar

Font size for the Simulink function label, specified as a scalar. The `StateFont.Size` property of the chart that contains the graphical function sets the initial value of this property.

### Hierarchy

**`Chart` — Chart that contains Simulink function**
`Stateflow.Chart` object

This property is read-only.

Chart that contains the Simulink function, specified as a `Stateflow.Chart` object.

**`Subviewer` — Subviewer for Simulink function**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

This property is read-only.

Subviewer for the Simulink function, specified as a `Stateflow.Chart`, `Stateflow.State`, `Stateflow.Box`, or `Stateflow.Function` object. The subviewer is the chart or subchart where you can graphically view the Simulink function.

**`Machine` — Machine that contains Simulink function**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the Simulink function, specified as a `Stateflow.Machine` object.

**`Path` — Location of parent in model hierarchy**
character vector

This property is read-only.

Location of the parent of the Simulink function in the model hierarchy, specified as a character vector.

### Identification

**`Description` — Description**
`''` (default) | character vector

Description for the Simulink function, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the Simulink function, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the Simulink function, specified as data of any type.

**`SSIdNumber` — Session-independent identifier**
scalar

This property is read-only.

Session-independent identifier, specified as an integer scalar. Use this property to distinguish the MATLAB function from other objects in the model.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Unlike `SSIdNumber`, the value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| find | Specified objects in hierarchy |
| fitToView | Zoom in on graphical object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| highlight | Highlight graphical object |
| isCommented | Determine if object is commented out |
| set | Set properties with specified values |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add Simulink Function to Chart

Add a Simulink function in the chart `ch`. Set its label to `'[y1,y2] = f(x1,x2,x3)'`.

```
function = Stateflow.SLFunction(ch);
function.LabelString = '[y1,y2] = f(x1,x2,x3)';
```

## See Also

`Stateflow.Box` | `Stateflow.Chart` | `Stateflow.Function` | `Stateflow.State`

### Topics

“Overview of the Stateflow API” on page 1-2
“Reuse Simulink Components in Stateflow Charts”
“List of Stateflow API Properties” on page 4-2

### Introduced in R2008b

# Stateflow.State

State in chart, state, or box

# Description

Use `Stateflow.State` objects to describe an operating mode of a reactive system. For more information, see “Represent Operating Modes by Using States”.

# Creation

## Syntax

```
state = Stateflow.State(parent)
```

### Description

`state = Stateflow.State(parent)` creates a `Stateflow.State` object in a parent chart, state, or box.

### Input Arguments

#### `parent` — Parent for new state

`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object

Parent for the new state, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.State`

## Properties

### Content

#### `Name` — Name of state

`''` (default) | character vector

Name of the state, specified as a character vector.

#### `LabelString` — Full label for state

`'?'` (default) | character vector

Full label for the state, specified as a character vector. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16.

#### `DuringAction` — State during action

character vector

This property is read-only.

State `during` action, specified as a character vector. The value of this property depends on the `LabelString` property for the state. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16. This property is not supported in Moore charts.

**`EntryAction` — State `entry` action**
character vector

This property is read-only.

State `entry` action, specified as a character vector. The value of this property depends on the `LabelString` property for the state. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16. This property is not supported in Moore charts.

**`ExitAction` — State `exit` action**
character vector

This property is read-only.

State `exit` action, specified as a character vector. The value of this property depends on the `LabelString` property for the state. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16. This property is not supported in Moore charts.

**`MooreAction` — State action in Moore chart**
character vector

This property is read-only.

State action in a Moore chart, specified as a character vector. The value of this property depends on the `LabelString` property for the state. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16. This property is supported only in Moore charts. For more information, see “Design Rules for Moore Charts”.

**`OnAction` — State `on` actions**
cell array of character vectors

This property is read-only.

State `on` actions, specified as a cell array of character vectors in the form

```
{'trigger1','action1',...,'triggerN','actionN'}
```

The value of this property depends on the `LabelString` property for the state. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16. This property is not supported in Moore charts.

**`IsExplicitlyCommented` — Whether to comment out state**
`false` or 0 (default) | `true` or 1

Whether to comment out the state, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the state and selecting **Comment Out**. For more information, see “Commenting Stateflow Objects in a Chart”.

**`IsImplicitlyCommented` — Whether state is implicitly commented out**
`true` or 1 | `false` or 0

This property is read-only.

Whether the state is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The state is implicitly commented out when you comment out a superstate in its hierarchy.

**`CommentText` — Comment text**
`''` (default) | character vector

Comment text added to the state, specified as a character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge on the state, the text appears as a tooltip. When you set the `IsExplicitlyCommented` property to `false`, the value of `CommentText` reverts to `''`.

**Graphical Appearance**

**`Position` — Position and size of state**
`[0 0 90 60]` (default) | `[left top width height]`

Position and size of the state, specified as a four-element numeric vector of the form `[left top width height]`.

**`BadIntersection` — Whether state intersects a box, state, or function**
`true` or 1 | `false` or 0

This property is read-only.

Whether the state graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`IsGrouped` — Whether state is a grouped state**
`false` or 0 (default) | `true` or 1

Whether the state is a grouped state, specified as a numeric or logical 1 (`true`) or 0 (`false`). When you copy and paste a grouped state, you copy not only the state but all of its contents. For more information, see “Copy and Paste by Grouping” on page 2-24.

**`IsSubchart` — Whether state is a subchart**
`false` or 0 (default) | `true` or 1

Whether the state is a subchart, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`ContentPreviewEnabled` — Whether to display preview of state contents**
`false` or 0 (default) | `true` or 1

Whether to display a preview of the state contents, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `IsSubchart` property is `true`.

**`ArrowSize` — Size of incoming transition arrows**
8 (default) | scalar

Size of incoming transition arrows, specified as a scalar.

**`FontSize` — Font size for state label**
scalar

Font size for the state label, specified as a scalar. The `StateFont.Size` property of the chart that contains the state sets the initial value of this property.

### State Decomposition

#### `Decomposition` — Decomposition of substates

`'EXCLUSIVE_OR'` (default) | `'PARALLEL_AND'`

Decomposition of substates at the top level of containment in the state, specified as `'EXCLUSIVE_OR'` or `'PARALLEL_AND'`. For more information, see “Specify Substate Decomposition”.

#### `Type` — Decomposition of sibling states

`'AND'` | `'OR'`

This property is read-only.

Decomposition of sibling states, specified as `'OR'` or `'AND'`. The state inherits this property from the `Decomposition` property of its parent state or chart.

#### `ExecutionOrder` — Execution order in parallel (AND) decomposition

scalar

Execution order for the state in parallel (AND) decomposition, specified as an integer scalar. This property applies only when both of these conditions are satisfied:

- The `Type` property of the state is `'AND'`.
- The `UserSpecifiedStateTransitionExecutionOrder` property of the chart that contains the state is `true`.

### Active State Output

#### `HasOutputData` — Whether to create active state data output

`false` or 0 (default) | `true` or 1

Whether to create an active state data output port for the state, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Monitor State Activity Through Active State Data”.

#### `OutputData` — Active state data object

`Stateflow.Data` object

This property is read-only.

Active state data object for the state, specified as a `Stateflow.Data` object. This property applies only when the `HasOutputData` property for the state is `true`.

#### `OutputPortName` — Name of active state data object

character vector

Name of the active state data object for the state, specified as a character vector. This property applies only when the `HasOutputData` property for the state is `true`.

#### `OutputMonitoringMode` — Monitoring mode for active state output

`'SelfActivity'` (default) | `'ChildActivity'` | `'LeafStateActivity'`

Monitoring mode for the active state output data, specified as `'SelfActivity'`, `'ChildActivity'`, or `'LeafStateActivity'`.

**`EnumTypeName` — Name of enumerated data type for active state data object**
character vector

Name of the enumerated data type for the active state data object for the state, specified as a character vector. This property applies only when the `OutputMonitoringMode` property for the state is `'ChildActivity'` or `'LeafStateActivity'`. For more information, see “Enum Name”.

**`DoNotAutogenerateEnum` — Whether to define enumerated data type manually**
`false` or 0 (default) | `true` or 1

Whether to define the enumerated data type for the active state data output manually, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `OutputMonitoringMode` property for the state is `'ChildActivity'` or `'LeafStateActivity'`. For more information, see “Define State Activity Enumeration Type”.

**Signal Logging**

**`LoggingInfo.DataLogging` — Whether to enable signal logging for state**
`false` or 0 (default) | `true` or 1

Whether to enable signal logging for the state, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Log Simulation Output for States and Data”.

Example: `state.LoggingInfo.DataLogging = true;`

**`LoggingInfo.DecimateData` — Whether to limit logged data**
`false` or 0 (default) | `true` or 1

Whether to limit the amount of logged data, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is `true`, signal logging skips samples by using the interval size specified by the `LoggingInfo.Decimation` property.

Example: `state.LoggingInfo.DeimateData = true;`

**`LoggingInfo.Decimation` — Decimation interval**
2 (default) | scalar

Decimation interval, specified as an integer scalar. The default value of 2 means that the chart logs every other sample.

Example: `state.LoggingInfo.Decimation = 5;`

**`LoggingInfo.LimitDataPoints` — Whether to limit number of data points to log**
`false` or 0 (default) | `true` or 1

Whether to limit the number of data points to log, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is `true`, signal logging limits the number of data points by using the value specified by the `LoggingInfo.MaxPoints` property.

Example: `state.LoggingInfo.LimitDataPoints = true;`

**`LoggingInfo.MaxPoints` — Maximum number of data points to log**
`5000` (default) | scalar

Maximum number of data points to log, specified as an integer scalar. The default value of `5000` means the chart logs the last 5000 data points generated by the simulation.

Example: `state.LoggingInfo.MaxPoints = 100;`

**`LoggingInfo.NameMode` — Source of signal name**
`'SignalName'` (default) | `'Custom'`

Source of the signal name used to log the state, specified as one of these values:

- `'SignalName'` — Use the name of the state.
- `'Custom'` — Use the custom signal name specified by the `LoggingInfo.LoggingName` property.

Example: `state.LoggingInfo.NameMode = 'Custom';`

**`LoggingInfo.LoggingName` — Custom signal name**
character vector

Custom signal name used for logging the state, specified as a character vector. This property applies only when the `LoggingInfo.NameMode` property is `'Custom'`.

Example: `state.LoggingInfo.LoggingName = 'State';`

**Debugging**

**`Debug.Breakpoints.OnDuring` — Whether to set `During State` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `During State` breakpoint for the state, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `state.Debug.Breakpoints.OnDuring = true;`

**`Debug.Breakpoints.OnEntry` — Whether to set `On State Entry` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `On State Entry` breakpoint for the state, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `state.Debug.Breakpoints.OnEntry = true;`

**`Debug.Breakpoints.OnExit` — Whether to set `On State Exit` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `On State Exit` breakpoint for the state, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `state.Debug.Breakpoints.OnExit = true;`

**`TestPoint` — Whether to set state as test point**
`false` or 0 (default) | `true` or 1

Whether to set the state as a test point, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Monitor Test Points in Stateflow Charts”.

### Code Generation

**`InlineOption` — Appearance in generated code**
`'Auto'` (default) | `'Function'` | `'Inline'`

Appearance of the state functions in generated code, specified as one of these values:

- `'Auto'` — An internal calculation determines the appearance of state functions in generated code.
- `'Function'` — State functions are implemented as separate C functions.
- `'Inline'` — Calls to state functions are replaced by code.

For more information, see “Inline State Functions in Generated Code” (Simulink Coder).

### Hierarchy

**`Chart` — Chart that contains state**
`Stateflow.Chart` object

This property is read-only.

Chart that contains the state, specified as a `Stateflow.Chart` object.

**`Subviewer` — Subviewer for state**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

This property is read-only.

Subviewer for the state, specified as a `Stateflow.Chart`, `Stateflow.State`, or `Stateflow.Box` object. The subviewer is the chart or subchart where you can graphically view the state.

**`Machine` — Machine that contains state**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the state, specified as a `Stateflow.Machine` object.

**`Path` — Location of parent in model hierarchy**
character vector

This property is read-only.

Location of the parent of the state in the model hierarchy, specified as a character vector.

### Identification

**`Description` — Description**
`''` (default) | character vector

Description for the state, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the state, specified as a character vector.

**`Tag` — User-defined tag**
[ ] (default) | any data type

User-defined tag for the state, specified as data of any type.

**`SSIdNumber` — Session-independent identifier**
scalar

This property is read-only.

Session-independent identifier, specified as an integer scalar. Use this property to distinguish the state from other objects in the model.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Unlike `SSIdNumber`, the value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| defaultTransitions | Return default transitions in object at top level of containment |
| dialog | Open properties dialog box |
| find | Specified objects in hierarchy |
| fitToView | Zoom in on graphical object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| highlight | Highlight graphical object |
| innerTransitions | Return inner transitions that originate with chart or state and terminate on contained object |
| isCommented | Determine if object is commented out |
| outerTransitions | Return array of outer transitions for object |
| set | Set properties with specified values |
| sinkedTransitions | Return transitions that have object as destination |
| sourcedTransitions | Return transitions that have object as source |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add State to Chart

Add a state in the chart `ch`. Set its name to `'A'`.

```
state = Stateflow.State(ch);
state.Name = 'A';
```

### Enter Multiline Label in State

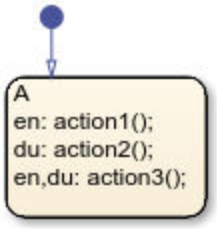
To enter a multiline label in the state `state`, you can:

- Call the MATLAB function `sprintf` and use the escape sequence `\n` to insert newline characters:

```
str = sprintf('A\nen: action1();\ndu: action2();\nen,du: action3();');
state.LabelString = str;
```

- Enter a concatenated text expression that uses the integer 10 as the ASCII equivalent of a newline character:

```
str = ['A',10, ...
    'en: action1();',10, ...
    'du: action2();',10, ...
    'en,du: action3();'];
state.LabelString = str;
```




To extract the state name, `entry` action, and `during` action specified by the state label, enter:

```
name = sA.Name

name =

    'A'

entry = sA.EntryAction

entry =

    ' action1();
      action3();'

during = sA.DuringAction

during =

    ' action2();
      action3();'
```

For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16.

### Add Supertransition from Subchart

Create a supertransition that connects junction `j1`, which is inside a subchart, to junction `j2`, which is outside the subchart.

Save the original position of subchart `st` to a temporary workspace variable `subchartPosition`.

```
subchartPosition = st.Position;
```

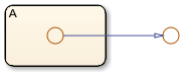
Convert the subchart to a normal state by setting its `IsSubchart` and `IsGrouped` properties to `false`.

```
st.IsSubchart = false;
st.IsGrouped = false;
```

When you convert a subchart to a normal state, it may change size to display its contents.

Add a transition that connects junction `j1` to junction `j2` in the chart `ch`.

```
tr = Stateflow.Transition(ch);
tr.Source = j1;
tr.Destination = j2;
```



Revert the state to a subchart by setting its `IsSubchart` property to `true`. Restore the subchart to its original position.

```
st.IsSubchart = true;
st.Position = subchartPosition;
```

The state A is now a subchart and the transition between the junctions is now a supertransition.

For more information, see “Move Between Levels of Hierarchy by Using Supertransitions”.

## See Also

`Stateflow.Box` | `Stateflow.Chart` | `Stateflow.Transition`

**Topics**

“Overview of the Stateflow API” on page 1-2
“Represent Operating Modes by Using States”
“Specify Labels in States and Transitions Programmatically” on page 1-16
“List of Stateflow API Properties” on page 4-2

**Introduced before R2006a**

# Stateflow.StateTransitionTableChart

Tabular representation of state machine for modal logic

## Description

Use a `Stateflow.StateTransitionTableChart` object to create a tabular representation of a finite state machine for modal logic. The benefits of using state transition tables include:

- The ease of modeling train-like state machines, where the modal logic involves transitions from one state to its neighbor
- A concise, compact format for a state machine
- Reduced maintenance of graphical objects

For more information, see “State Transition Tables in Stateflow”.

## Creation

To create a `Stateflow.StateTransitionTableChart` object, call the function `sfnew` with the `-STT` argument. For example, to create a State Transition Table block in a new Simulink model called `myModel`, enter:

```
sfnew -STT myModel
```

Alternatively, you can add a new State Transition Table block to an existing model by using the function `add_block`:

```
add_block('sflib/State Transition Table','myModel/State Transition Table')
```

Then, to access the `Stateflow.StateTransitionTableChart` object, call the `find` function for the `Simulink.Root` object:

```
rt = sfroot;
table = find(rt,'-isa','Stateflow.StateTransitionTableChart', ...
    'Path','myModel/State Transition Table');
```

### Properties

#### Content

**`Name` — Name of state transition table**
`'State Transition Table'` (default) | character vector

Name of the state transition table, specified as a character vector.

**`ActionLanguage` — Action language**
`'MATLAB'` (default) | `'C'`

Action language used to program the state transition table, specified as `'MATLAB'` or `'C'`. For more information, see “Differences Between MATLAB and C as Action Language Syntax”.

**`StateMachineType` — State machine semantics**
`'Classic'` (default) | `'Mealy'` | `'Moore'`

State machine semantics implemented by the state transition table, specified as `'Classic'`, `'Mealy'`, or `'Moore'`. For more information, see “Overview of Mealy and Moore Machines”.

**`SupportVariableSizing` — Whether state transition table supports variable-size data**
`true` or 1 (default) | `false` or 0

Whether the state transition table supports variable-size data, specified as a numeric or logical 1 (`true`) or 0 (`false`). Only variable-size data can change dimension during simulation. For more information, see “Declare Variable-Size Data in Stateflow Charts”.

**Chart Initialization**

**`ExecuteAtInitialization` — Whether to initialize state configuration**
`false` or 0 (default) | `true` or 1

Whether to initialize the state configuration of the state transition table at time zero instead of at the first input event, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Execution of a Chart at Initialization”.

**`StatesWhenEnabling` — Behavior of states when event reenables state transition table**
`''` (default) | `'held'` | `'reset'`

Behavior of the states when a function-call input event reenables the state transition table, specified as one of these values:

- `''` — The state transition table does not contain function-call input events.
- `'held'` — The state transition table maintains the most recent values of the states.
- `'reset'` — The state transition table reverts to the initial conditions of the states.

For more information, see “Control States in Charts Enabled by Function-Call Input Events”.

**`InitializeOutput` — Whether to initialize output data**
`false` or 0 (default) | `true` or 1

Whether to initialize the output data every time the state transition table wakes up, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Initialize outputs every time chart wakes up”.

**Active State Output**

**`HasOutputData` — Whether to create active state data output**
`false` or 0 (default) | `true` or 1

Whether to create an active state data output port for the state transition table, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Monitor State Activity Through Active State Data”.

**`OutputData` — Active state data object**
`Stateflow.Data` object

This property is read-only.

Active state data object for the state transition table, specified as a `Stateflow.Data` object. This property applies only when the `HasOutputData` property for the state transition table is `true`.

**`OutputPortName` — Name of active state data object**
character vector

Name of the active state data object for the state transition table, specified as a character vector. This property applies only when the `HasOutputData` property for the state transition table is `true`.

**`OutputMonitoringMode` — Monitoring mode for active state output**
`'ChildActivity'` (default) | `'LeafStateActivity'`

Monitoring mode for the active state output data, specified as `'ChildActivity'` or `'LeafStateActivity'`.

**`EnumTypeName` — Name of enumerated data type for active state data object**
character vector

Name of the enumerated data type for the active state data object for the state transition table, specified as a character vector. For more information, see “Enum Name”.

**`DoNotAutogenerateEnum` — Whether to define enumerated data type manually**
`false` or 0 (default) | `true` or 1

Whether to define the enumerated data type for the active state data output manually, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Define State Activity Enumeration Type”.

**Discrete and Continuous-Time Semantics**

**`ChartUpdate` — Activation method for state transition table**
`'INHERITED'` (default) | `'CONTINUOUS'` | `'DISCRETE'`

Activation method for the state transition table, specified as `'CONTINUOUS'`, `'DISCRETE'`, or `'INHERITED'`. For more information, see “Update Method”.

**`SampleTime` — Sample time for activating state transition table**
`'-1'` (default) | character vector

Sample time for activating the state transition table, specified as a character vector. This property applies only when the `ChartUpdate` property for the state transition table is `'DISCRETE'`.

**`EnableZeroCrossings` — Whether to enable zero-crossing detection**
`true` or 1 (default) | `false` or 0

Whether to enable zero-crossing detection on state transitions in the state transition table, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `ChartUpdate` property for the state transition table is set to `'CONTINUOUS'`. For more information, see “Disable Zero-Crossing Detection”.

**Super Step Semantics**

**`EnableNonTerminalStates` — Whether to enable super step semantics**
`false` or 0 (default) | `true` or 1

Whether to enable super step semantics for the state transition table, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Super Step Semantics”.

**`NonTerminalMaxCounts` — Maximum number of transitions in one super step**
`1000` (default) | scalar

Maximum number of transitions the state transition table can take in one super step, specified as an integer scalar. This property applies only when the `EnableNonTerminalStates` property for the state transition table is `true`.

**`NonTerminalUnstableBehavior` — Behavior if super step exceeds maximum number of transitions**
`'Proceed'` (default) | `'Throw Error'`

Behavior if a super step for the state transition table exceeds the maximum number of transitions specified in the `NonTerminalMaxCounts` property before reaching a stable state, specified as one of these values:

- `'Proceed'` — The state transition table goes to sleep with the last active state configuration.
- `'Throw Error'` — The state transition table generates an error.

This property applies only when the `EnableNonTerminalStates` property for the state transition table is `true`.

**Integer and Fixed-Point Data**

**`SaturateOnIntegerOverflow` — Whether data saturates on integer overflow**
`true` or 1 (default) | `false` or 0

Whether the data in the state transition table saturates on integer overflow, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is disabled, the data in the state transition table wraps on integer overflow. For more information, see “Handle Integer Overflow for Chart Data”.

**`TreatAsFi` — Inherited Simulink signals to treat as `fi` objects**
`'Fixed-point'` (default) | `'Fixed-point & Integer'`

Inherited Simulink signals to treat as Fixed-Point Designer `fi` objects, specified as one of these values:

- `'Fixed-point'` — The state transition table treats all fixed-point inputs as `fi` objects.
- `'Fixed-point & Integer'` — The state transition table treats all fixed-point and integer inputs as `fi` objects.

This property applies only when the `ActionLanguage` property of the state transition table is `'MATLAB'`.

**`EmlDefaultFimath` — Default `fimath` properties**
`'Same as MATLAB Default'` (default) | `'Other:UserSpecified'`

Default `fimath` properties for the state transition table, specified as one of these values:

- `'Same as MATLAB Default'` — Use the same `fimath` properties as the current default `fimath` object.
- `'Other:UserSpecified'` — Use the `InputFimath` property to specify the default `fimath` object.

This property applies only when the `ActionLanguage` property of the state transition table is `'MATLAB'`.

**`InputFimath` — Default `fimath` object**
character vector

Default `fimath` object, specified as a character vector. When the `EmlDefaultFimath` property for the state transition table is `'Other:UserSpecified'`, you can use this property to:

- Enter an expression that constructs a `fimath` object.
- Enter the variable name for a `fimath` object in the MATLAB or model workspace.

This property applies only when the `ActionLanguage` property of the state transition table is `'MATLAB'`.

**C Action Language**

**`StrongDataTypingWithSimulink` — Whether to use strong data typing**
`true` or 1 (default) | `false` or 0

Whether to use strong data typing when the state transition table interfaces with Simulink input and output signals, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only to state transition tables that use C as the action language. For more information, see “Use strong data typing with Simulink I/O”.

**`EnableBitOps` — Whether to use bit operations**
`false` or 0 (default) | `true` or 1

Whether to use bit operations in state and transition actions in the state transition table, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only to state transition tables that use C as the action language. For more information, see “Enable C-bit operations”.

**Debugging**

**`Debug.Breakpoints.OnEntry` — Whether to set `On Chart Entry` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `On Chart Entry` breakpoint for the state transition table, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `table.Debug.Breakpoints.OnEntry = true;`

**Graphical Appearance**

**`Editor` — Editor**
`Stateflow.Editor` object

This property is read-only.

Editor for the state transition table, specified as a `Stateflow.Editor` object. You can use this object to control the position, size, and magnification level of the Stateflow Editor window.

**`Visible` — Whether editor is displaying state transition table**
`true` or 1 | `false` or 0

Whether the Stateflow Editor window is displaying the state transition table, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`ChartColor` — Background color**
`[1 0.9608 0.8824]` (default) | `[red green blue]`

Background color for the chart that is automatically generated for the state transition table, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.

**`StateColor` — Color for states**
`[0 0 0]` (default) | `[red green blue]`

Color for the boxes, functions, and states in the chart that is automatically generated for the state transition table, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.

**`TransitionColor` — Color for transitions**
`[0.2902 0.3294 0.6039]` (default) | `[red green blue]`

Color for transitions in the chart that is automatically generated for the state transition table, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.

**`JunctionColor` — Color for junctions**
`[0.6824 0.3294 0]` (default) | `[red green blue]`

Color for junctions in the chart that is automatically generated for the state transition table, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.

**`StateLabelColor` — Color for state labels**
`[0 0 0]` (default) | `[red green blue]`

Color for the box, function, and state labels in the chart that is automatically generated for the state transition table, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.

**`StateFont.Angle` — Font angle for state labels**
`'NORMAL'` (default) | `'ITALIC'`

Font angle for the box, function, and state labels in the chart that is automatically generated for the state transition table, specified as `'NORMAL'` or `'ITALIC'`.

Example: `table.StateFont.Angle = 'ITALIC';`

**`StateFont.Weight` — Font weight for state labels**
`'NORMAL'` (default) | `'BOLD'`

Font weight for the box, function, and state labels in the chart that is automatically generated for the state transition table, specified as `'NORMAL'` or `'BOLD'`.

Example: `table.StateFont.Weight = 'BOLD;`

**`StateFont.Size` — Initial font size for state labels**
12 (default) | scalar

Initial font size for the annotation, box, function, and state labels in the chart that is automatically generated for the state transition table, specified as a scalar.

Example: `table.StateFont.Size = 8;`

**`StateFont.Name` — Font name for state labels**
`'Helvetica'` (default) | character vector

Font name for the annotation, box, function, and state labels in the chart that is automatically generated for the state transition table, specified as a character vector.

Example: `table.StateFont.Name = 'Arial';`

**`TransitionLabelColor` — Color for transition labels**
`[0.2902 0.3294 0.6039]` (default) | `[red green blue]`

Color for the transition labels in the chart that is automatically generated for the state transition table, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.

**`TransitionFont.Angle` — Font angle for transition labels**
`'NORMAL'` (default) | `'ITALIC'`

Font angle for the transition labels in the chart that is automatically generated for the state transition table, specified as `'NORMAL'` or `'ITALIC'`.

Example: `table.TransitionFont.Angle = 'ITALIC';`

**`TransitionFont.Weight` — Font weight for the transition labels in this chart**
`'NORMAL'` (default) | `'BOLD'`

Font weight for the transition labels in the chart that is automatically generated for the state transition table, specified as `'NORMAL'` or `'BOLD'`.

Example: `table.TransitionFont.Weight = 'BOLD';`

**`TransitionFont.Size` — Initial font size for transition labels**
`12` (default) | scalar

Initial font size for the transition labels in the chart that is automatically generated for the state transition table, specified as a scalar.

Example: `table.TransitionFont.Size = 8;`

**`TransitionFont.Name` — Font name for transition labels**
`'Helvetica'` (default) | character vector

Font name for the transition labels in the chart that is automatically generated for the state transition table, specified as a character vector.

Example: `table.TransitionFont.Name = 'Arial';`

**Hierarchy**

**`Machine` — Machine that contains state transition table**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the state transition table, specified as a `Stateflow.Machine` object.

**`Path` — Location of state transition table in model hierarchy**
character vector

This property is read-only.

Location of the state transition table in the model hierarchy, specified as a character vector.

**`Dirty` — Whether state transition table has changed**
`true` or 1 | `false` or 0

Whether the state transition table has changed after being opened or saved, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`Locked` — Whether state transition table is locked**
`false` or 0 (default) | `true` or 1

Whether the state transition table is locked, specified as a numeric or logical 1 (`true`) or 0 (`false`). Enable this property to prevent changes in the state transition table.

**`Iced` — Whether state transition table is locked**
`false` or 0 (default) | `true` or 1

This property is read-only.

Whether the state transition table is locked, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property is equivalent to the property `Locked`, but is used internally to prevent changes in the state transition table during simulation.

**Identification**

**`Description` — Description**
`''` (default) | character vector

Description for the state transition table, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the state transition table, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the state transition table, specified as data of any type.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Use this property to distinguish the state transition table from other objects in the model. The value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| find | Specified objects in hierarchy |
| get | Return MATLAB structure containing property settings of object or array of objects |
| parse | Parse single chart or all charts in model |
| set | Set properties with specified values |
| view | Display object in editing environment |

## Examples

### Create Empty State Transition Table

Call the function `sfnew` with the `-STT` argument to open a new Simulink model that contains an empty State Transition Table block.

```
sfnew -STT
```

Access the `Simulink.Root` object by calling the `sfroot` function.

```
rt = sfroot;
```

Access the `Stateflow.StateTransitionTableChart` object by calling the `find` function for the `Simulink.Root` object.

```
table = find(rt,'-isa','Stateflow.StateTransitionTableChart');
```

## See Also

**Blocks**
State Transition Table

**Functions**
`add_block` | `find` | `sfnew` | `sfroot`

**Topics**
“Overview of the Stateflow API” on page 1-2
“Finite State Machine Concepts”
“State Transition Tables in Stateflow”
“List of Stateflow API Properties” on page 4-2

**Introduced in R2012b**

# Stateflow.Transition

Transition in chart, state, box, or function

## Description

Use `Stateflow.Transition` objects to create transitions from one operating mode to another. For more information, see “Transition Between Operating Modes”.

## Creation

### Syntax

```
transition = Stateflow.Transition(parent)
```

#### Description

`transition = Stateflow.Transition(parent)` creates a `Stateflow.Transition` object in a parent chart, state, box, or graphical function.

#### Input Arguments

**`parent` — Parent for new transition**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

Parent for the new transition, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.Function`
- `Stateflow.State`

### Properties

#### Content

**`LabelString` — Full label for transition**
`''` (default) | character vector

Full label for the transition, specified as a character vector. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16.

**`Condition` — Transition condition**
character vector

This property is read-only.

Transition condition, specified as a character vector. The value of this property depends on the `LabelString` property for the transition. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16.

**`ConditionAction` — Transition condition action**
character vector

This property is read-only.

Transition condition action, specified as a character vector. The value of this property depends on the `LabelString` property for the transition. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16.

**`TransitionAction` — Transition action**
character vector

This property is read-only.

Transition action, specified as a character vector. The value of this property depends on the `LabelString` property for the transition. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16.

**`Trigger` — Transition trigger**
character vector

This property is read-only.

Transition trigger, specified as a character vector. The value of this property depends on the `LabelString` property for the transition. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16.

**`ExecutionOrder` — Execution order for transition**
scalar

Execution order for the transition when its source is active, specified as an integer scalar. This property applies only when the `UserSpecifiedStateTransitionExecutionOrder` property of the chart that contains the transition is `true`. For more information, see “Transition Evaluation Order”.

**`IsExplicitlyCommented` — Whether to comment out transition**
`false` or 0 (default) | `true` or 1

Whether to comment out the transition, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the transition and selecting **Comment Out**. For more information, see “Commenting Stateflow Objects in a Chart”.

**`IsImplicitlyCommented` — Whether transition is implicitly commented out**
`true` or 1 | `false` or 0

This property is read-only.

Whether the transition is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The transition is implicitly commented out when you comment out a superstate in its hierarchy.

### `CommentText` — Comment text

`''` (default) | character vector

Comment text added to the transition, specified as a character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge  on the transition, the text appears as a tooltip. When you set the `IsExplicitlyCommented` property to `false`, the value of `CommentText` reverts to `''`.

### Graphical Appearance

### `Source` — Source of transition

`[]` (default) | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Junction` object | ...

Source of the transition, specified as an empty array or a Stateflow API object of one of these types:

- `Stateflow.AtomicBox`
- `Stateflow.AtomicSubchart`
- `Stateflow.Box`
- `Stateflow.Junction`
- `Stateflow.SimulinkBasedState`
- `Stateflow.State`

### `SourceEndPoint` — Position of transition endpoint at source

`[2 2]` (default) | `[x y]`

Position of the transition endpoint at its source, specified as a two-element numeric vector `[x y]` of coordinates relative to the upper left corner of the chart.

### `SourceOClock` — Location of transition endpoint at source

`0` (default) | scalar between 0 and 12

Location of the transition endpoint at its source, specified as a scalar between 0 and 12 that describes a clock position.

### `Destination` — Destination of transition

`[]` (default) | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Junction` object | ...

Destination of the transition, specified as an empty array or a Stateflow API object of one of these types:

- `Stateflow.AtomicBox`
- `Stateflow.AtomicSubchart`
- `Stateflow.Box`
- `Stateflow.Junction`
- `Stateflow.SimulinkBasedState`
- `Stateflow.State`

### `DestinationEndPoint` — Position of transition endpoint at destination

`[40 40]` (default) | `[x y]`

Position of the transition endpoint at its destination, specified as a two-element numeric vector `[x y]` of coordinates relative to the upper left corner of the chart.

**`DestinationOClock` — Location of transition endpoint at destination**
`0` (default) | scalar between 0 and 12

Location of the transition endpoint at its destination, specified as a scalar between 0 and 12 that describes a clock position.

**`MidPoint` — Position of midpoint of transition**
`[21 21]` (default) | `[x y]`

Position of the midpoint of the transition, specified as a two-element numeric vector `[x y]` of coordinates relative to the upper left corner of the chart.

**`LabelPosition` — Position and size of transition label**
`[0 0 8 14]` (default) | `[left top width height]`

Position and size of the transition label, specified as a four-element numeric vector of the form `[left top width height]`.

**`ArrowSize` — Size of transition arrow**
scalar

Size of the transition arrow at the destination, specified as a scalar. When you change the destination of the transition, this property resets to the value of the `ArrowSize` property of the new destination.

**`FontSize` — Font size for transition label**
scalar

Font size for the transition label, specified as a scalar. The `TransitionFont.Size` property of the chart that contains the transition sets the initial value of this property.

**Debugging**

**`Debug.Breakpoints.WhenTested` — Whether to set `When Transition is Tested` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `When Transition is Tested` breakpoint for the transition, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `transition.Debug.Breakpoints.WhenTested = true;`

**`Debug.Breakpoints.WhenValid` — Whether to set `When Transition is Valid` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `When Transition is Valid` breakpoint for the transition, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `transition.Debug.Breakpoints.WhenValid = true;`

**Code Generation**

**`IsVariant` — Whether transition is a variant transition**
`false` or 0 (default) | `true` or 1

Whether the transition is a variant transition, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Code Generation Using Variant Transitions”.

### Hierarchy

**`Chart` — Chart that contains transition**
`Stateflow.Chart` object

This property is read-only.

Chart that contains the transition, specified as a `Stateflow.Chart` object.

**`Subviewer` — Subviewer for transition**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

This property is read-only.

Subviewer for the transition, specified as a `Stateflow.Chart`, `Stateflow.State`, `Stateflow.Box`, or `Stateflow.Function` object. The subviewer is the chart or subchart where you can graphically view the transition.

**`Machine` — Machine that contains transition**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the transition, specified as a `Stateflow.Machine` object.

**`Path` — Location of parent in model hierarchy**
character vector

This property is read-only.

Location of the parent of the transition in the model hierarchy, specified as a character vector.

### Identification

**`Description` — Description**
`''` (default) | character vector

Description for the transition, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the transition, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the transition, specified as data of any type.

**`SSIdNumber` — Session-independent identifier**
scalar

This property is read-only.

Session-independent identifier, specified as an integer scalar. Use this property to distinguish the transition from other objects in the model.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Unlike `SSIdNumber`, the value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| fitToView | Zoom in on graphical object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| highlight | Highlight graphical object |
| isCommented | Determine if object is commented out |
| set | Set properties with specified values |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add Transition to Chart

Add a transition that connects state `s1` to state `s2` in the chart `ch`.

```
transition = Stateflow.Transition(ch);
transition.Source = s1;
transition.Destination = s2;
```

### Label Transitions

Add a label that specifies a trigger, condition, and condition action on the transition `transition`.

```
transition.LabelString = 'trigger[guard]{action();}';
```


To extract the trigger, condition, and condition action specified by the transition label, enter:

```
trigger = transition.Trigger
```

```
trigger =

    'trigger'

condition = transition.Condition

condition =

    'guard'

action = transition.ConditionAction

action =

    'action();'
```

### Add a Default Transition

Create a `Stateflow.Transition` object in the `Stateflow.Chart` object `ch`.

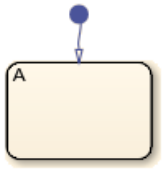
```
dt = Stateflow.Transition(ch);
```

Set the destination of the transition to the `Stateflow.State` object `st`.

```
dt.Destination = st;
dt.DestinationOClock = 0;
```

Place the source endpoint for the transition 30 pixels above the destination endpoint. Place the midpoint for the transition 15 pixels above the destination endpoint.

```
dt.SourceEndPoint = dt.DestinationEndPoint-[0 30];
dt.MidPoint = dt.DestinationEndPoint-[0 15];
```




### Add Supertransition from Subchart

Create a supertransition that connects junction `j1`, which is inside a subchart, to junction `j2`, which is outside the subchart.


Save the original position of subchart `st` to a temporary workspace variable `subchartPosition`.

```
subchartPosition = st.Position;
```

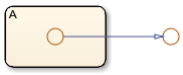
Convert the subchart to a normal state by setting its `IsSubchart` and `IsGrouped` properties to `false`.

```
st.IsSubchart = false;
st.IsGrouped = false;
```

When you convert a subchart to a normal state, it may change size to display its contents.

Add a transition that connects junction `j1` to junction `j2` in the chart `ch`.

```
tr = Stateflow.Transition(ch);
tr.Source = j1;
tr.Destination = j2;
```



Revert the state to a subchart by setting its `IsSubchart` property to `true`. Restore the subchart to its original position.

```
st.IsSubchart = true;
st.Position = subchartPosition;
```

The state A is now a subchart and the transition between the junctions is now a supertransition.

For more information, see “Move Between Levels of Hierarchy by Using Supertransitions”.

## See Also

`Stateflow.AtomicBox` | `Stateflow.AtomicSubchart` | `Stateflow.Box` | `Stateflow.Chart` | `Stateflow.Function` | `Stateflow.Junction` | `Stateflow.SimulinkBasedState` | `Stateflow.State`

### Topics

“Overview of the Stateflow API” on page 1-2
“Transition Between Operating Modes”
“List of Stateflow API Properties” on page 4-2

### Introduced before R2006a

# Stateflow.TruthTable

Truth table function in chart, state, box, or function

## Description

Use `Stateflow.TruthTable` objects to create truth table functions that implement combinatorial logic design in a concise, tabular format. Typical applications include decision making for:

- Fault detection and management
- Mode switching

You can call a truth table function in the actions of states and transitions. For more information, see "Use Truth Tables to Model Combinatorial Logic".

## Creation

### Syntax

```
function = Stateflow.TruthTable(parent)
```

#### Description

`function = Stateflow.TruthTable(parent)` creates a `Stateflow.TruthTable` object in a parent chart, state, box, or function.

#### Input Arguments

**`parent` — Parent for new truth table**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

Parent for the new truth table, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.Function`
- `Stateflow.State`

### Properties

#### Content

**`Name` — Name of truth table**
`''` (default) | character vector

Name of the truth table, specified as a character vector.

**`LabelString` — Full label for truth table**
`'?'` (default) | character vector

Full label for the truth table, specified as a character vector.

**`ActionTable` — Action table**
cell array of character vectors

Action table for the truth table, specified as a cell array of character vectors.

**`ConditionTable` — Condition table**
cell array of character vectors

Condition table for the truth table, specified as a cell array of character vectors.

**`Language` — Action language**
`'MATLAB'` (default) | `'C'`

Action language used to program the truth table, specified as `'MATLAB'` or `'C'`. The option `'C'` is supported only in truth tables in charts that use C as the action language. For more information, see "Differences Between MATLAB and C as Action Language Syntax".

**`IsExplicitlyCommented` — Whether to comment out truth table**
`false` or 0 (default) | `true` or 1

Whether to comment out the truth table, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the truth table and selecting **Comment Out**. For more information, see "Commenting Stateflow Objects in a Chart".

**`IsImplicitlyCommented` — Whether truth table is implicitly commented out**
`true` or 1 | `false` or 0

This property is read-only.

Whether the truth table is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The truth table is implicitly commented out when you comment out a superstate in its hierarchy.

**`CommentText` — Comment text**
`''` (default) | character vector

Comment text added to the truth table, specified as a character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge on the truth table, the text appears as a tooltip. When you set the `IsExplicitlyCommented` property to `false`, the value of `CommentText` reverts to `''`.

**Graphical Appearance**

**`Position` — Position and size of truth table**
`[0 0 90 60]` (default) | `[left top width height]`

Position and size of the truth table, specified as a four-element numeric vector of the form `[left top width height]`.

**`BadIntersection` — Whether function intersects a box, state, or function**
`true` or 1 | `false` or 0

This property is read-only.

Whether the truth table graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`FontSize` — Font size for truth table label**
scalar

Font size for the truth table label, specified as a scalar. The `StateFont.Size` property of the chart that contains the truth table sets the initial value of this property.

**Debugging**

**`OverSpecDiagnostic` — Level of diagnostic when truth table is overspecified**
`'Error'` (default) | `'Warning'` | `'None'`

Level of diagnostic action when the truth table is overspecified, specified as `'Error'`, `'Warning'`, or `'None'`. For more information, see “Correct Overspecified and Underspecified Truth Tables”.

**`UnderSpecDiagnostic` — Level of diagnostic when truth table is underspecified**
`'Error'` (default) | `'Warning'` | `'None'`

Level of diagnostic action when the truth table is underspecified, specified as `'Error'`, `'Warning'`, or `'None'`. For more information, see “Correct Overspecified and Underspecified Truth Tables”.

**`Debug.Breakpoints.OnDuring` — Whether to set `During Function Call` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `During Function Call` breakpoint for the truth table, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when both the `Language` property of the truth table and the `ActionLanguage` of the chart that contains the truth table are `'C'`.

Example: `function.Debug.Breakpoints.OnDuring = true;`

**Integer and Fixed-Point Data**

**`SaturateOnIntegerOverflow` — Whether data saturates on integer overflow**
`true` or 1 (default) | `false` or 0

Whether the data in the truth table saturates on integer overflow, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is disabled, the data in the truth table wraps on integer overflow. This property applies only when the `Language` property of the truth table is `'MATLAB'` and the `ActionLanguage` of the chart that contains the truth table is `'C'`. For more information, see “Handle Integer Overflow for Chart Data”.

**`EmlDefaultFimath` — Default `fimath` properties**
`'Same as MATLAB Default'` (default) | `'Other:UserSpecified'`

Default `fimath` properties for the truth table, specified as one of these values:

- `'Same as MATLAB Default'` — Use the same `fimath` properties as the current default `fimath` object.

- `'Other:UserSpecified'` — Use the `InputFimath` property to specify the default `fimath` object.

This property applies only when the `Language` property of the truth table is `'MATLAB'` and the `ActionLanguage` of the chart that contains the truth table is `'C'`.

**`InputFimath` — Default `fimath` object**
character vector

Default `fimath` object, specified as a character vector. When the `EmlDefaultFimath` property for the MATLAB function is `'Other:UserSpecified'`, you can use this property to:

- Enter an expression that constructs a `fimath` object.
- Enter the variable name for a `fimath` object in the MATLAB or model workspace.

This property applies only when the `Language` property of the truth table is `'MATLAB'` and the `ActionLanguage` of the chart that contains the truth table is `'C'`.

**Code Generation**

**`InlineOption` — Appearance in generated code**
`'Auto'` (default) | `'Function'` | `'Inline'`

Appearance of the truth table in generated code, specified as one of these values:

- `'Auto'` — An internal calculation determines the appearance of the function in generated code.
- `'Function'` — The function is implemented as a separate C function.
- `'Inline'` — Calls to the function are replaced by code.

For more information, see “Inline State Functions in Generated Code” (Simulink Coder).

**Hierarchy**

**`Chart` — Chart that contains truth table**
`Stateflow.Chart` object

This property is read-only.

Chart that contains the truth table, specified as a `Stateflow.Chart` object.

**`Subviewer` — Subviewer for truth table**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object

This property is read-only.

Subviewer for the truth table, specified as a `Stateflow.Chart`, `Stateflow.State`, `Stateflow.Box`, or `Stateflow.Function` object. The subviewer is the chart or subchart where you can graphically view the truth table.

**`Machine` — Machine that contains truth table**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the truth table, specified as a `Stateflow.Machine` object.

**`Path` — Location of parent in model hierarchy**
character vector

This property is read-only.

Location of the parent of the truth table in the model hierarchy, specified as a character vector.

**Identification**

**`Description` — Description**
`''` (default) | character vector

Description for the truth table, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the truth table, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the truth table, specified as data of any type.

**`SSIdNumber` — Session-independent identifier**
scalar

This property is read-only.

Session-independent identifier, specified as an integer scalar. Use this property to distinguish the truth table from other objects in the model.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Unlike `SSIdNumber`, the value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| find | Specified objects in hierarchy |
| fitToView | Zoom in on graphical object |
| get | Return MATLAB structure containing property settings of object or array of objects |
| highlight | Highlight graphical object |
| isCommented | Determine if object is commented out |
| set | Set properties with specified values |
| up | Return parent of object |
| view | Display object in editing environment |

## Examples

### Add Truth Table Function to Chart

Add a truth table function in the chart `ch`. Set its label to `'[y1,y2] = f(x1,x2,x3)'`.

```
function = Stateflow.TruthTable(ch);
function.LabelString = '[y1,y2] = f(x1,x2,x3)';
```

## See Also

`Stateflow.Box` | `Stateflow.Chart` | `Stateflow.Function` | `Stateflow.State`

### Topics

“Overview of the Stateflow API” on page 1-2
“Use Truth Tables to Model Combinatorial Logic”
“List of Stateflow API Properties” on page 4-2

### Introduced before R2006a

# Stateflow.TruthTableChart

Tabular representation of state machine for decision logic

## Description

Use `Stateflow.TruthTableChart` objects to create truth table blocks that implement combinatorial logic design in a concise, tabular format. Typical applications include decision making for:

- Fault detection and management
- Mode switching

Truth table blocks execute as Simulink blocks and provide a more direct implementation of decision logic than using truth table functions in Stateflow charts. For more information, see “Use Truth Tables to Model Combinatorial Logic”.

## Creation

To create a `Stateflow.TruthTableChart` object, call the function `sfnew` with the `-TT` argument. For example, to create a Truth Table block in a new Simulink model called `myModel`, enter:

```
sfnew -TT myModel
```

Alternatively, you can add a new Truth Table block to an existing model by using the function `add_block`:

```
add_block('sflib/Truth Table','myModel/Truth Table')
```

Then, to access the `Stateflow.TruthTableChart` object, call the `find` function for the `Simulink.Root` object:

```
rt = sfroot;
table = find(rt,'-isa','Stateflow.TruthTableChart', ...
    'Path','myModel/Truth Table');
```

### Properties

#### Content

**`Name` — Name of truth table**
`'Truth Table'` (default) | character vector

Name of the truth table, specified as a character vector.

**`ActionTable` — Action table**
cell array of character vectors

Action table for the truth table, specified as a cell array of character vectors.

**`ConditionTable` — Condition table**
cell array of character vectors

Condition table for the truth table, specified as a cell array of character vectors.

**`SupportVariableSizing` — Whether truth table supports variable-size data**
`true` or 1 (default) | `false` or 0

Whether the truth table supports variable-size data, specified as a numeric or logical 1 (`true`) or 0 (`false`). Only variable-size data can change dimension during simulation. For more information, see "Declare Variable-Size Data in Stateflow Charts".

**Discrete and Continuous-Time Semantics**

**`ChartUpdate` — Activation method for truth table**
`'INHERITED'` (default) | `'CONTINUOUS'` | `'DISCRETE'`

Activation method for the truth table, specified as `'CONTINUOUS'`, `'DISCRETE'`, or `'INHERITED'`. For more information, see "Update Method".

**`SampleTime` — Sample time for activating truth table**
`'-1'` (default) | character vector

Sample time for activating the truth table, specified as a character vector. This property applies only when the `ChartUpdate` property for the truth table is `'DISCRETE'`.

**Integer and Fixed-Point Data**

**`SaturateOnIntegerOverflow` — Whether data saturates on integer overflow**
`true` or 1 (default) | `false` or 0

Whether the data in the truth table saturates on integer overflow, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is disabled, the data in the truth table wraps on integer overflow. For more information, see "Handle Integer Overflow for Chart Data".

**`TreatAsFi` — Inherited Simulink signals to treat as `fi` objects**
`'Fixed-point'` (default) | `'Fixed-point & Integer'`

Inherited Simulink signals to treat as Fixed-Point Designer `fi` objects, specified as one of these values:

- `'Fixed-point'` — The truth table treats all fixed-point inputs as `fi` objects.
- `'Fixed-point & Integer'` — The truth table treats all fixed-point and integer inputs as `fi` objects.

**`EmlDefaultFimath` — Default `fimath` properties**
`'Same as MATLAB Default'` (default) | `'Other:UserSpecified'`

Default `fimath` properties for the truth table, specified as one of these values:

- `'Same as MATLAB Default'` — Use the same `fimath` properties as the current default `fimath` object.
- `'Other:UserSpecified'` — Use the `InputFimath` property to specify the default `fimath` object.

**`InputFimath` — Default `fimath` object**
character vector

Default `fimath` object, specified as a character vector. When the `EmlDefaultFimath` property for the truth table is `'Other:UserSpecified'`, you can use this property to:

- Enter an expression that constructs a `fimath` object.
- Enter the variable name for a `fimath` object in the MATLAB or model workspace.

**Debugging**

**`OverSpecDiagnostic` — Level of diagnostic when truth table is overspecified**
`'Error'` (default) | `'Warning'` | `'None'`

Level of diagnostic action when the truth table is overspecified, specified as `'Error'`, `'Warning'`, or `'None'`. For more information, see “Correct Overspecified and Underspecified Truth Tables”.

**`UnderSpecDiagnostic` — Level of diagnostic when truth table is underspecified**
`'Error'` (default) | `'Warning'` | `'None'`

Level of diagnostic action when the truth table is underspecified, specified as `'Error'`, `'Warning'`, or `'None'`. For more information, see “Correct Overspecified and Underspecified Truth Tables”.

**Hierarchy**

**`Machine` — Machine that contains truth table**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the truth table, specified as a `Stateflow.Machine` object.

**`Path` — Location of truth table in model hierarchy**
character vector

This property is read-only.

Location of the truth table in the model hierarchy, specified as a character vector.

**`Dirty` — Whether truth table has changed**
`true` or 1 | `false` or 0

Whether the truth table has changed after being opened or saved, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`Locked` — Whether truth table is locked**
`false` or 0 (default) | `true` or 1

Whether the truth table is locked, specified as a numeric or logical 1 (`true`) or 0 (`false`). Enable this property to prevent changes in the truth table.

**`Iced` — Whether truth table is locked**
`false` or 0 (default) | `true` or 1

This property is read-only.

Whether the truth table is locked, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property is equivalent to the property `Locked`, but is used internally to prevent changes in the truth table during simulation.

#### Identification

**`Description` — Description**
`''` (default) | character vector

Description for the truth table, specified as a character vector.

**`Document` — Document link**
`''` (default) | character vector

Document link for the truth table, specified as a character vector.

**`Tag` — User-defined tag**
`[]` (default) | any data type

User-defined tag for the truth table, specified as data of any type.

**`Id` — Unique identifier**
scalar

This property is read-only.

Unique identifier, specified as an integer scalar. Use this property to distinguish the truth table from other objects in the model. The value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted.

## Object Functions

| | |
|---|---|
| classhandle | Provide class handle for object |
| dialog | Open properties dialog box |
| find | Specified objects in hierarchy |
| get | Return MATLAB structure containing property settings of object or array of objects |
| parse | Parse single chart or all charts in model |
| set | Set properties with specified values |
| view | Display object in editing environment |

## Examples

#### Create Empty Truth Table

Call the function `sfnew` with the `-TT` argument to open a new Simulink model that contains an empty Truth Table block.

```
sfnew -TT
```

Access the `Simulink.Root` object by calling the `sfroot` function.

```
rt = sfroot;
```

Access the `Stateflow.TruthTableChart` object by calling the `find` function for the `Simulink.Root` object.

```
table = find(rt,'-isa','Stateflow.TruthTableChart');
```

## See Also

**Blocks**
Truth Table

**Functions**
`add_block` | `find` | `sfnew` | `sfroot`

**Topics**
“Overview of the Stateflow API” on page 1-2
“Finite State Machine Concepts”
“Use Truth Tables to Model Combinatorial Logic”
“List of Stateflow API Properties” on page 4-2

**Introduced before R2006a**

# API Object Function Reference

# classhandle

Provide class handle for object

## Syntax

```
handle = classhandle(thisObject)
```

## Description

The `classhandle` function returns a read-only class handle for a Stateflow API object. You can use the `classhandle` function to retrieve information about the structure of each object type.

## Arguments

| | |
|---|---|
| `thisObject` | The object for which to return a class handle. Can be any Stateflow object. |

## Returns

| | |
|---|---|
| `handle` | Class handle for this object. |

## Examples

If `j` is a `Stateflow.Junction` object, `classhandle(j)` returns the class handle of a `Stateflow.Junction` object. To display the structure for this class handle, enter:

```
get(classhandle(j))
```

The structure for this class handle includes the fields `Names` and `Properties`. To display the `Properties` substructure, enter:

```
get(classhandle(j).Properties)
```

The `Properties` substructure includes the fields `Name` and `DataType`. You can display a list of property names and data types of a `Stateflow.Junction` object with the following command:

```
get(classhandle(j).Properties,{'Name','DataType'})
```

## See Also

`get`

### Topics

“Overview of the Stateflow API” on page 1-2
“Modify Properties and Call Functions of Stateflow Objects” on page 1-10

### Introduced before R2006a

# copy

**Package:** `Stateflow`

Copy array of objects to clipboard

## Syntax

```
copy(clipboard,objArray)
```

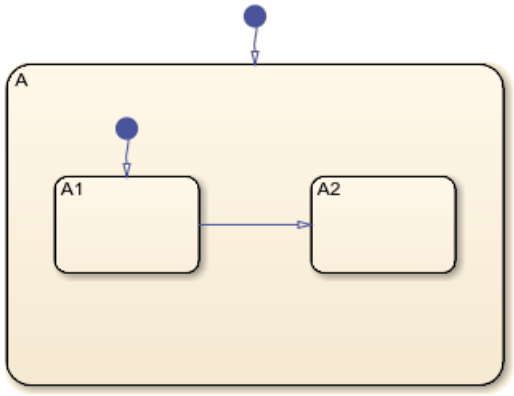
## Description

`copy(clipboard,objArray)` copies the objects in the array `objArray` to the clipboard. To paste the copied objects, use the `pasteTo` function.

## Examples

### Copy and Paste by Grouping

Group state `A` and copy its contents to chart `ch`. When you group a state, box, or graphical function, you can copy and paste all the objects contained in the grouped object, as well as all the relationships among these objects. This method is the simplest way of copying and pasting objects programmatically. If a state is not grouped, copying the state does not copy any of its contents.




**1** Find the `Stateflow.State` object named `A` in chart `ch`.

```
sA = find(ch,'-isa','Stateflow.State','Name','A');
```

**2** Group state `A` and its contents by setting the `IsGrouped` property for `sA` to `true`. Save the previous setting of this property so you can revert to it later.

```
prevGrouping = sA.IsGrouped;
sA.IsGrouped = true;
```

**3** Change the name of the state to `'Copy_of_A'`. Save the previous name so you can revert to it later.

```
prevName = sA.Name;
newName = ['Copy_of_' prevName];
sA.Name = newName;
```

4 Access the clipboard object.

```
cb = sfclipboard;
```

5 Copy the grouped state to the clipboard.

```
copy(cb,sA);
```

6 Restore the state properties to their original settings.

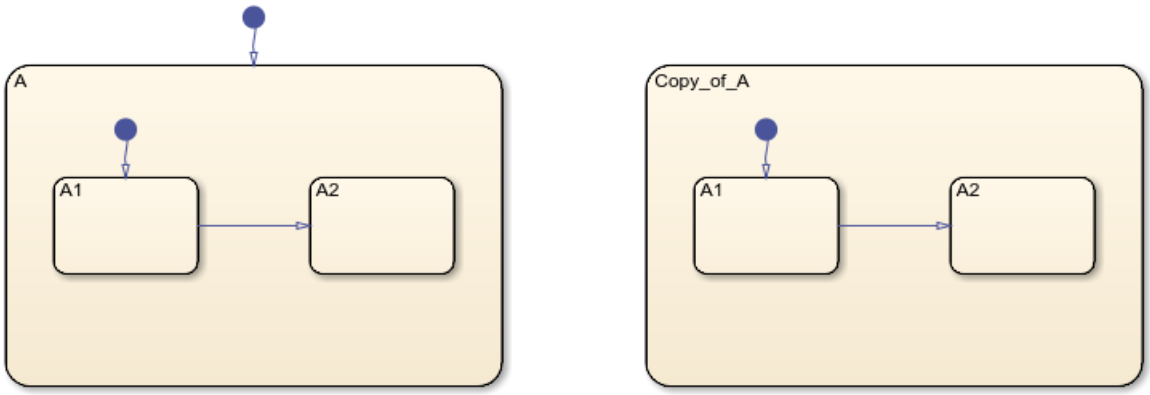
```
sA.IsGrouped = prevGrouping;
sA.Name = prevName;
```

7 Paste a copy of the objects from the clipboard to the chart.

```
pasteTo(cb,ch);
```

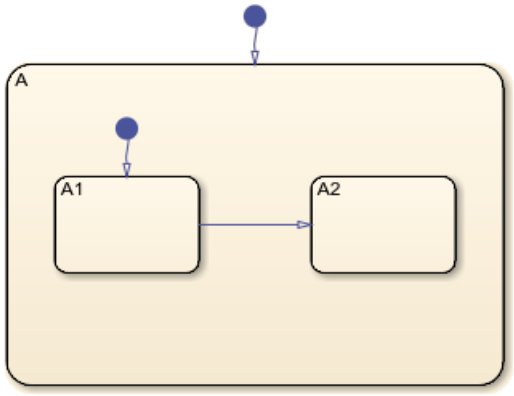
8 Adjust the state properties of the new state.

```
sNew = find(ch,'-isa','Stateflow.State','Name',newName);
sNew.Position = sA.Position + [400 0 0 0];
sNew.IsGrouped = prevGrouping;
```




### Copy and Paste Array of Objects

Copy states `A1` and `A2`, along with the transition between them, to a new state in chart `ch`. To preserve transition connections and containment relationships between objects, copy all the connected objects at once.




1. Find the `Stateflow.State` object named `A` in chart `ch`.

```
sA = find(ch,'-isa','Stateflow.State','Name','A');
```

2. Add a new state called `B`. To enable pasting of other objects inside `B`, convert the new state to a subchart.

```
sB = Stateflow.State(ch);
sB.Name = 'B';
sB.Position = sA.Position + [400 0 0 0];
sB.IsSubchart = true;
```

3. Create an array called `objArray` that contains the states and transitions in state `A`. Use the function `setdiff` to remove state `A` from the array of objects to copy.

```
objArrayS = find(sA,'-isa','Stateflow.State');
objArrayS = setdiff(objArrayS,sA);
objArrayT = find(sA,'-isa','Stateflow.Transition');
objArray = [objArrayS objArrayT];
```

4. Access the clipboard object.

```
cb = sfclipboard;
```

5. Copy the objects in `objArray` and paste them in subchart `B`.

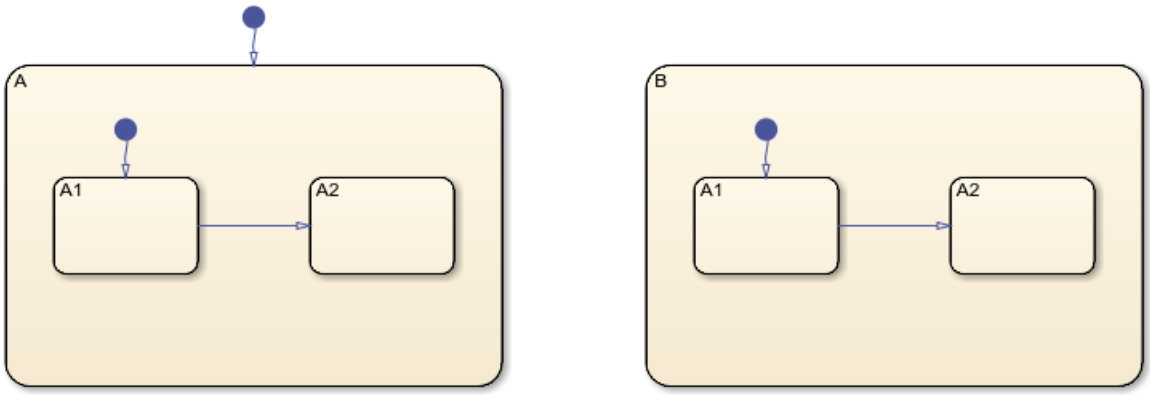
```
copy(cb,objArray);
pasteTo(cb,sB);
```

6. Revert `B` to a state.

```
sB.IsSubchart = false;
sB.IsGrouped = false;
```

7. Reposition the states and transitions in `B`.

```
newStates = find(sB,'-isa','Stateflow.State');
newStates = setdiff(newStates,sB);
newTransitions = find(sB,'-isa','Stateflow.Transition');
newOClocks = get(newTransitions,{'SourceOClock','DestinationOClock'});
for i = 1:numel(newStates)
newStates(i).Position = newStates(i).Position + [25 35 0 0];
end
set(newTransitions,{'SourceOClock','DestinationOClock'},newOClocks);
```




## Input Arguments

### `clipboard` — Clipboard

`Stateflow.Clipboard` object

Clipboard, specified as a `Stateflow.Clipboard` object.

### `objArray` — Objects to copy

array of Stateflow objects

Objects to copy, specified as an array of Stateflow API objects. The array must contain only graphical objects or only nongraphical objects.

Graphical objects include:

- `Stateflow.Annotation`
- `Stateflow.AtomicBox`
- `Stateflow.AtomicSubchart`
- `Stateflow.Box`
- `Stateflow.EMFunction`
- `Stateflow.Function`
- `Stateflow.Junction`
- `Stateflow.SimulinkBasedState`
- `Stateflow.SLFunction`
- `Stateflow.State`
- `Stateflow.Transition`
- `Stateflow.TruthTable`

Nongraphical objects include:

- `Stateflow.Data`
- `Stateflow.Event`
- `Stateflow.Message`

Copying graphical objects also copies the `Stateflow.Data`, `Stateflow.Event`, and `Stateflow.Message` objects that the graphical objects contain. When you copy multiple graphical objects, the value of their `Subviewer` property must be the same.

## Tips

To maintain the transition connections and containment relationships between copied objects, you must:

- Copy a grouped object to the clipboard. When you group a state, box, or graphical function, you can copy and paste all the objects contained in the grouped object, as well as all the relationships among these objects. For more information, see “Copy and Paste by Grouping” on page 3-3.
- Copy all the related objects. For example, to copy two states connected by a transition to another container, create an array that contains both the states and the transition. Then you can copy the array to the clipboard. For more information, see “Copy and Paste Array of Objects” on page 3-4.

## See Also

**Functions**
`find` | `get` | `pasteTo` | `set` | `setdiff` | `sfclipboard`

**Objects**
`Stateflow.Clipboard` | `Stateflow.State`

**Topics**
“Overview of the Stateflow API” on page 1-2

**Introduced before R2006a**

# defaultTransitions

Return default transitions in object at top level of containment

## Syntax

```
defaultTransitions = defaultTransitions(thisObject)
```

## Description

The `defaultTransitions` function returns the default transitions in this object at the top level of containment.

## Arguments

| | |
|---|---|
| `thisObject` | The object for which to return default transitions. Can be an object of type Chart, State, Box, or Function. |

## Returns

| | |
|---|---|
| `defaultTransitions` | Array of default transitions in this object at the top level of containment. |

## Examples

If state `A` contains state `A1`, and state `A1` contains state `A11`, and states `A1` and `A11` have default transitions attached to them, the `defaultTransitions` function of state `A` returns the default transition attached to state `A1`.

#### Introduced before R2006a

# dialog

**Package:** `Stateflow`

Open properties dialog box

## Syntax

```
dialog(object)
```

## Description

`dialog(object)` opens the properties dialog box of an object.

## Examples

### Open Chart Properties Dialog Box

Open a Simulink model called `myModel`. Suppose that the model contains a Stateflow chart named `My Chart`.

```
open_system('myModel')
```

Access the `Simulink.Root` object at the top level of the Stateflow hierarchy.

```
rt = sfroot;
```

Find the chart named `My Chart`.

```
ch = find(rt,'-isa','Stateflow.Chart','Name','My Chart');
```

Open the properties dialog box for the chart.

```
dialog(ch);
```

## Input Arguments

### `object` — Object to inspect

`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object | ...

Object to inspect, specified as a Stateflow API object of one of these types:

- `Stateflow.Annotation`
- `Stateflow.AtomicBox`
- `Stateflow.AtomicSubchart`
- `Stateflow.Box`
- `Stateflow.Chart`

- `Stateflow.Data`
- `Stateflow.EMChart`
- `Stateflow.EMFunction`
- `Stateflow.Event`
- `Stateflow.Function`
- `Stateflow.Junction`
- `Stateflow.Machine`
- `Stateflow.Message`
- `Stateflow.SimulinkBasedState`
- `Stateflow.SLFunction`
- `Stateflow.State`
- `Stateflow.StateTransitionTableChart`
- `Stateflow.Transition`
- `Stateflow.TruthTable`
- `Stateflow.TruthTableChart`

## See Also

`get` | `set`

**Topics**

“Overview of the Stateflow API” on page 1-2

**Introduced before R2006a**

# find

**Package:** `Stateflow`

Specified objects in hierarchy

## Syntax

```
objArray = find(location,Name,Value)
objArray = find(location,'-not',Name,Value)
objArray = find(location,'-regexp',Name,Value)
objArray = find(location,___,logicalOp,___)
```

## Description

`objArray = find(location,Name,Value)` returns an array of objects in the hierarchy of `location` that match the criteria specified by one or more `Name,Value` pair arguments.

`objArray = find(location,'-not',Name,Value)` returns objects that do not match the criteria specified by the subsequent `Name,Value` pair argument.

`objArray = find(location,'-regexp',Name,Value)` indicates that the subsequent `Name,Value` pair argument contains a regular expression. For more information, see “Regular Expressions”.

`objArray = find(location,___,logicalOp,___)` combines search criteria by using one of these logical operations:

- `'-and'` — Results must match both search criteria.
- `'-or'` — Results must match at least one criterion.
- `'-xor'` — Results must match exactly one criterion.

When using various logical operators, `-and` has the highest precedence, while `-or` and `-xor` are right-associative. If no logical operator is specified, then `-and` is assumed.

## Examples

### Find States in a Chart

Find all states in the chart `ch`.

```
states = find(ch,'-isa','Stateflow.State')
```

### Find States Named A

Find all states in the chart `ch` whose `Name` property is `'A'`.

```
statesNamedA = find(ch,'-isa','Stateflow.State','-and','Name','A')
```

#### Find Objects with Name Starting with A

Find all objects in the chart `ch` whose `Name` property starts with the letter `A`.

```
startsWithA = find(ch,'-regexp','Name','^A')
```

#### Find Nongraphical Objects

Find all objects in the chart `ch` that do not have an object function called `fitToView`.

```
nongraphical = find(ch,'-not','-method','fitToView')
```

#### Use Function to Specify Search Criteria

Find all charts in a Simulink model called `myModel`.

```
f = @(h) (strcmp(h.Machine.Name,'myModel'));                    % define function handle
ch = find(rt,'-isa','Stateflow.Chart','-and','-function',f);    % find charts for which f returns
```

## Input Arguments

### `location` — Location to search

`Simulink.Root` object | `Stateflow.Chart` object | `Stateflow.State` object | ...

Location to search, specified as a Stateflow API object of one of these types:

- `Simulink.Root`
- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.EMChart`
- `Stateflow.EMFunction`
- `Stateflow.Function`
- `Stateflow.Machine`
- `Stateflow.SimulinkBasedState`
- `Stateflow.State`
- `Stateflow.SLFunction`
- `Stateflow.StateTransitionTableChart`
- `Stateflow.TruthTable`
- `Stateflow.TruthTableChart`

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and

value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. In addition to the `Name,Value` arguments listed here, you can use the name of a Stateflow API property and its corresponding value. For more information, see “List of Stateflow API Properties” on page 4-2.

Example: `find(ch,'Name','A')` finds all objects in the chart `ch` whose `Name` property is `'A'`.

**`-isa` — Type of object**
character vector | class handle

Type of object for which to search, specified as the comma-separated pair consisting of `'-isa'` and a character vector or a class handle for an object.

Example: `find(ch,'-isa','Stateflow.State')` finds all states in the chart `ch`.

Example: `find(ch,'-isa',classhandle(object))` finds all objects that have the same class handle as `object`.

**`-depth` — Depth of search**
`inf` (default) | scalar nonnegative integer

Depth of search in the object hierarchy, specified as the comma-separated pair consisting of `'-depth'` and a scalar nonnegative integer or `inf`.

Example: `find(ch,'-depth',2)` finds all objects in the top two levels of the hierarchy of the chart `ch`.

**`-function` — Filtering function**
function handle

Filtering function, specified as the comma-separated pair consisting of `'-function'` and a function handle. The function evaluates each object visited in the search and returns a logical scalar value that indicates whether the object is a match.

Example: `find(ch,'-function',f)` finds all objects for which `f` is `true`.

**`-method` — Object function**
character vector

Object function that belongs to the objects for which to search, specified as the comma-separated pair consisting of `'-method'` and a character vector.

Example: `find(ch,'-method','dialog')` finds all objects in the chart `ch` that have an object function called `dialog`.

**`-property` — Property**
character vector

Property that belongs to the objects for which to search, specified as the comma-separated pair consisting of `'-property'` and a character vector.

Example: `find(ch,'-property','HasOutputData')` finds all objects in the chart `ch` that have a property called `HasOutputData`.

## Output Arguments

**`objArray` — Search results**
array

Search results, returned as an array of Stateflow API objects.

## Tips

- Using the `find` function on `Simulink.Root` or `Stateflow.Machine` objects can return Simulink objects that match the search criteria you specify. For example, this command can return a Simulink subsystem or block named `ABC`:

  ```
  find(rt,'Name','ABC')
  ```

- Opening a main model that refers to a linked Stateflow chart does not guarantee that the Stateflow API can find the linked chart. To access the objects in a linked library chart, first load the library model into the Simulink workspace by performing one of these tasks:

  - Open the library model.
  - View a linked subsystem or block in the main model.
  - Compile or simulate the model.

## See Also

`classhandle` | `get` | `strcmp`

**Topics**

“Access Objects in Your Stateflow Chart” on page 1-6
“List of Stateflow API Properties” on page 4-2
“Regular Expressions”

**Introduced before R2006a**

# fitToView

**Package:** `Stateflow`

Zoom in on graphical object

## Syntax

```
fitToView(graphicalObject)
```

## Description

`fitToView(graphicalObject)` zooms in on a graphical object in the Stateflow Editor.

## Examples

### Zoom in on State in Chart

Open a Simulink model called `myModel`. Suppose that the model contains a Stateflow chart with a state named `A`.

```
open_system('myModel')
```

Access the `Simulink.Root` object at the top level of the Stateflow hierarchy.

```
rt = sfroot;
```

Find the state named `A`.

```
st = find(rt,'-isa','Stateflow.State','Name','A');
```

Zoom in on the state in the Stateflow Editor.

```
fitToView(st);
```

## Input Arguments

### `graphicalObject` — Graphical object

`Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object | ...

Graphical object, specified as a Stateflow API object of one of these types:

- `Stateflow.Annotation`
- `Stateflow.AtomicBox`
- `Stateflow.AtomicSubchart`
- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.EMFunction`

- `Stateflow.Function`
- `Stateflow.Junction`
- `Stateflow.SimulinkBasedState`
- `Stateflow.SLFunction`
- `Stateflow.State`
- `Stateflow.Transition`
- `Stateflow.TruthTable`

## See Also

`highlight` | `view` | `zoomIn` | `zoomOut`

**Topics**

“Overview of the Stateflow API” on page 1-2

**Introduced in R2008a**

# get

Return MATLAB structure containing property settings of object or array of objects

## Syntax

```
propList = get(thisObject,prop)
```

## Description

The `get` function returns and displays a MATLAB structure containing the settings for the specified property of this object. If no property is specified, the settings for all properties are returned.

The `get` function is also vectorized so that it returns an `m`-by-`n` cell array of values for an array of `m` objects and an array of `n` properties.

## Arguments

| | |
|---|---|
| `thisObject` | The object for which to get specified property. |
| `prop` | Name of property (e.g., `'FontSize'`) to get value for. Can also be an array of properties (see return `propList` below). If no property is specified, a list of all properties is returned. |

## Returns

| | |
|---|---|
| `propList` | MATLAB structure listing the properties of this object. Can also be an `m` by `n` cell array of values if `thisObject` is an array of `m` objects and `prop` is an array of `n` properties. |

## Examples

State `A` is represented by the State object `sA`.

The following command lists the properties of state `A`:

```
get(sA)
```

The following command stores a `struct` with the properties of state `A` in the workspace variable `AProperties`:

```
AProperties = get(sA);
```

## See Also

`classhandle`

**Topics**

“Overview of the Stateflow API” on page 1-2
“Modify Properties and Call Functions of Stateflow Objects” on page 1-10

**Introduced before R2006a**

# highlight

**Package:** `Stateflow`

Highlight graphical object

## Syntax

```
highlight(graphicalObject)
```

## Description

`highlight(graphicalObject)` highlights a graphical object in the Stateflow Editor.

## Examples

### Highlight State in Chart

Open a Simulink model called `myModel`. Suppose that the model contains a Stateflow chart with a state named `A`.

```
open_system('myModel')
```

Access the `Simulink.Root` object at the top level of the Stateflow hierarchy.

```
rt = sfroot;
```

Find the state named `A`.

```
st = find(rt,'-isa','Stateflow.State','Name','A');
```

Highlight the state in the Stateflow Editor.

```
highlight(st);
```

## Input Arguments

### `graphicalObject` — Graphical object

`Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object | ...

Graphical object, specified as a Stateflow API object of one of these types:

- `Stateflow.AtomicBox`
- `Stateflow.AtomicSubchart`
- `Stateflow.Box`
- `Stateflow.EMFunction`
- `Stateflow.Function`
- `Stateflow.Junction`

- `Stateflow.SimulinkBasedState`
- `Stateflow.SLFunction`
- `Stateflow.State`
- `Stateflow.Transition`
- `Stateflow.TruthTable`

## Tips

To clear the highlighting, use the `hilite_system` function. For example, to clear the highlighting in chart `ch`, enter:

```
hilite_system(ch.Path,'none')
```

Alternatively, you can use the Stateflow Editor. In the **Debug** tab, under **Animation**, click the Remove animation highlighting button  .

## See Also

`fitToView` | `hilite_system` | `view` | `zoomIn` | `zoomOut`

### Topics

“Overview of the Stateflow API” on page 1-2

### Introduced in R2012a

# innerTransitions

Return inner transitions that originate with chart or state and terminate on contained object

## Syntax

```
transitions = innerTransitions(thisObject)
```

## Description

The `innerTransitions` function returns the inner transitions that originate with this object and terminate on a contained object.

## Arguments

None

## Returns

| | |
|---|---|
| `thisObject` | Object for which to get inner transitions. Can be of type State or Box. |
| `transitions` | Array of inner transitions originating with this object and terminating on a contained state or junction. |

## Examples

State `A` contains state `A1`, and state `A1` contains state `A11`. State `A` has two transitions, each originating from its inside edge and terminating inside it. These are inner transitions. One transition terminates with state `A1` and the other terminates with state `A11`. The `innerTransitions` function of state `A` returns both of these transitions.

#### Introduced before R2006a

# isCommented

Determine if object is commented out

## Syntax

```
isCommented(thisObject)
```

## Description

Returns a Boolean indicating if `thisObject` is explicitly or implicitly commented out.

## Arguments

| | |
|---|---|
| `thisObject` | The object which you determine if it is commented out. |

## Returns

If the object is explicitly or implicitly commented out, returns the Boolean value `true`. Otherwise, returns `false`.

#### Introduced in R2016a

# outerTransitions

Return array of outer transitions for object

## Syntax

```
transitions = outerTransitions(thisObject)
```

## Description

The `outerTransitions` function returns an array of transitions that exit the outer edge of this object and terminate on objects outside the containment of this object.

## Arguments

None

| | |
|---|---|
| `thisObject` | The object for which to find outer transitions. Can be of object type State or Box. |

## Returns

| | |
|---|---|
| `transitions` | An array of transitions exiting the outer edge of this state. |

## Examples

A chart contains three states, `A`, `B`, and `C`. State `A` is connected to state `B` through a transition from state `A` to state `B`. State `B` is connected to state `C` through a transition from state `B` to state `C`. And state `C` is connected to state `A` through a transition from state `C` to state `A`. If state `A` is represented by `Stateflow.State` object `sA`, this command returns the transition from state `A` to state `B`:

```
tr = outerTransitions(sA)
```

**Introduced before R2006a**

# parse

Parse single chart or all charts in model

## Syntax

```
parse(thisChart)
```

```
parse(thisMachine)
```

## Description

For Chart objects, the `parse` function parses this chart.

For Machine objects, the `parse` function parses all the charts in this machine.

## Arguments

| | |
|---|---|
| `thisChart` | The chart to parse. |
| `thisMachine` | The machine containing charts to parse. |

## Returns

None

## Examples

If `ch` is a `Stateflow.Chart` object representing a chart, then this command parses the chart:

```
parse(ch)
```

#### Introduced before R2006a

# pasteTo

**Package:** `Stateflow`

Paste objects in clipboard to specified container object

## Syntax

```
pasteTo(clipboard,parent)
```

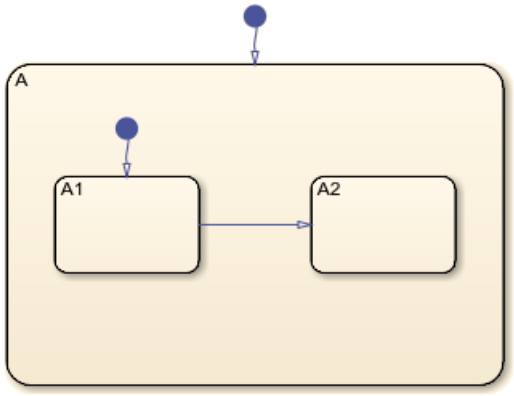
## Description

`pasteTo(clipboard,parent)` pastes the contents of the clipboard to the specified parent. To copy objects to the clipboard, use the `copy` function.

## Examples

### Copy and Paste by Grouping

Group state `A` and copy its contents to chart `ch`. When you group a state, box, or graphical function, you can copy and paste all the objects contained in the grouped object, as well as all the relationships among these objects. This method is the simplest way of copying and pasting objects programmatically. If a state is not grouped, copying the state does not copy any of its contents.




**1** Find the `Stateflow.State` object named `A` in chart `ch`.

```
sA = find(ch,'-isa','Stateflow.State','Name','A');
```

**2** Group state `A` and its contents by setting the `IsGrouped` property for `sA` to `true`. Save the previous setting of this property so you can revert to it later.

```
prevGrouping = sA.IsGrouped;
sA.IsGrouped = true;
```

**3** Change the name of the state to `'Copy_of_A'`. Save the previous name so you can revert to it later.

```
prevName = sA.Name;
newName = ['Copy_of_' prevName];
sA.Name = newName;
```

4 Access the clipboard object.

```
cb = sfclipboard;
```

5 Copy the grouped state to the clipboard.

```
copy(cb,sA);
```

6 Restore the state properties to their original settings.

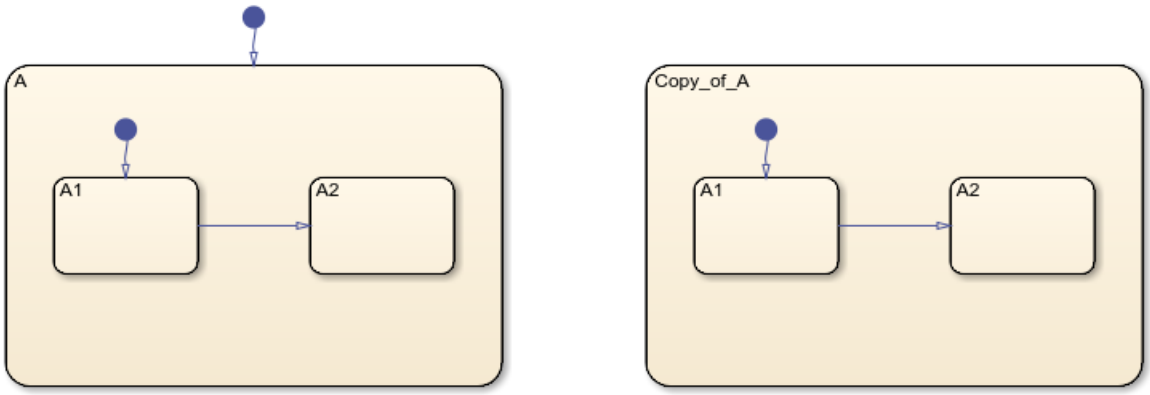
```
sA.IsGrouped = prevGrouping;
sA.Name = prevName;
```

7 Paste a copy of the objects from the clipboard to the chart.

```
pasteTo(cb,ch);
```

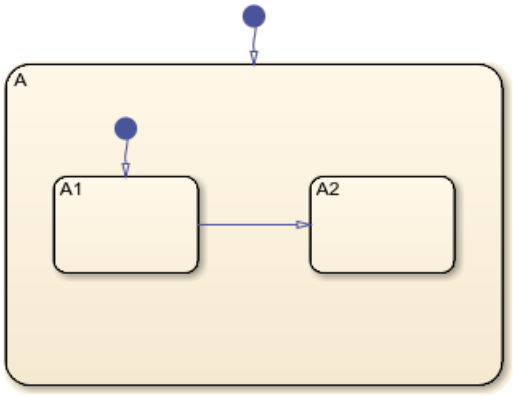
8 Adjust the state properties of the new state.

```
sNew = find(ch,'-isa','Stateflow.State','Name',newName);
sNew.Position = sA.Position + [400 0 0 0];
sNew.IsGrouped = prevGrouping;
```




### Copy and Paste Array of Objects

Copy states `A1` and `A2`, along with the transition between them, to a new state in chart `ch`. To preserve transition connections and containment relationships between objects, copy all the connected objects at once.




1. Find the `Stateflow.State` object named `A` in chart `ch`.

```
sA = find(ch,'-isa','Stateflow.State','Name','A');
```

2. Add a new state called `B`. To enable pasting of other objects inside `B`, convert the new state to a subchart.

```
sB = Stateflow.State(ch);
sB.Name = 'B';
sB.Position = sA.Position + [400 0 0 0];
sB.IsSubchart = true;
```

3. Create an array called `objArray` that contains the states and transitions in state `A`. Use the function `setdiff` to remove state `A` from the array of objects to copy.

```
objArrayS = find(sA,'-isa','Stateflow.State');
objArrayS = setdiff(objArrayS,sA);
objArrayT = find(sA,'-isa','Stateflow.Transition');
objArray = [objArrayS objArrayT];
```

4. Access the clipboard object.

```
cb = sfclipboard;
```

5. Copy the objects in `objArray` and paste them in subchart `B`.

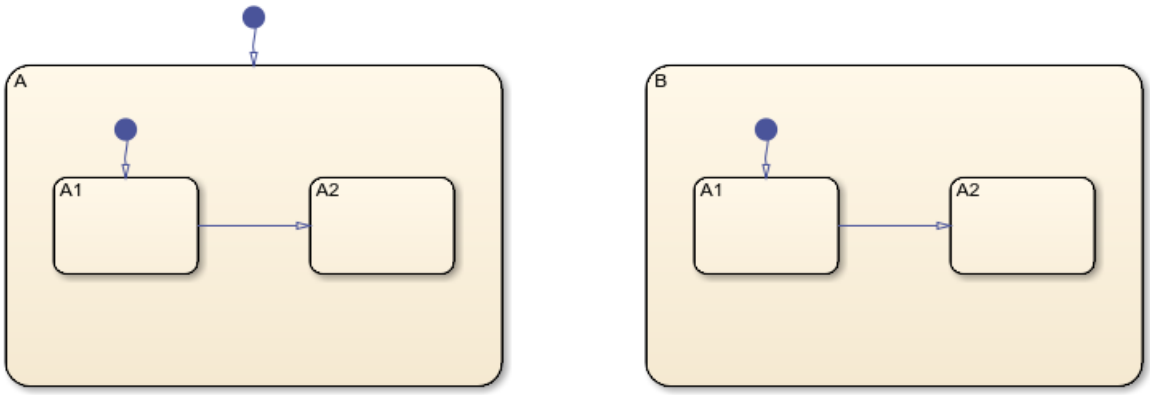
```
copy(cb,objArray);
pasteTo(cb,sB);
```

6. Revert `B` to a state.

```
sB.IsSubchart = false;
sB.IsGrouped = false;
```

7. Reposition the states and transitions in `B`.

```
newStates = find(sB,'-isa','Stateflow.State');
newStates = setdiff(newStates,sB);
newTransitions = find(sB,'-isa','Stateflow.Transition');
newOClocks = get(newTransitions,{'SourceOClock','DestinationOClock'});
for i = 1:numel(newStates)
newStates(i).Position = newStates(i).Position + [25 35 0 0];
end
set(newTransitions,{'SourceOClock','DestinationOClock'},newOClocks);
```




## Input Arguments

**`clipboard` — Clipboard**
`Stateflow.Clipboard` object

Clipboard, specified as a `Stateflow.Clipboard` object.

**`parent` — Parent for copied objects**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object | ...

Parent for the copied objects, specified as a Stateflow API object of one of these types:

- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.EMFunction`
- `Stateflow.Function`
- `Stateflow.SimulinkBasedState`
- `Stateflow.SLFunction`
- `Stateflow.State`
- `Stateflow.TruthTable`

If the objects in the clipboard are all graphical (states, boxes, functions, annotations, transitions, or junctions), this object must be a chart or subchart.

## Tips

When you paste graphical objects, the new parent must be a chart or a subchart. To convert a state, box, or graphical function to a subchart, set its `IsSubchart` property to `true`. After pasting, you can revert the parent by setting its `IsSubchart` and `IsGrouped` properties to `false`.

## See Also

**Functions**
`copy` | `find` | `get` | `set` | `setdiff` | `sfclipboard`

**Objects**
`Stateflow.Clipboard` | `Stateflow.State`

**Topics**
“Overview of the Stateflow API” on page 1-2

**Introduced before R2006a**

# set

Set properties with specified values

## Syntax

```
set(thisObject,propName,value,...)
```

**Note** Arguments can consist of an indefinite number of property (name, value) pairs.

## Description

The `set` function sets the value of a specified property or sets the values of a set of specified properties for this object. You specify properties and values through pairs of property (name, value) arguments.

The `get` function is also vectorized so that it sets an `m`-by-`n` cell array of values for an array of `m` objects and an array of `n` properties.

## Arguments

| | |
|---|---|
| `thisObject` | The object for which the specified property is set. Can be any Stateflow object. |
| `propName` | Name of the property to set (e.g., `'FontSize'`). Can also be a cell array of `m` property names. |
| `value` | New value for the specified property. Can be a cell array of `m`-by-`n` values if `thisObject` is an array of `m` objects and `propName` is an array of `n` property names. |

## Returns

None

## Examples

The following command sets the `Name` and `Description` properties of the `Stateflow.State` object `s`:

```
set(s,'Name', 'Kentucky', 'Description', 'Bluegrass State')
```

The following command sets the `Position` property of the `Stateflow.State` object `s`:

```
set(s,'Position',[200,119,90,60])
```

## See Also

`classhandle` | `get`

**Topics**
“Overview of the Stateflow API” on page 1-2
“Modify Properties and Call Functions of Stateflow Objects” on page 1-10

**Introduced before R2006a**

# setImage

**Package:** `Stateflow`

Insert image into annotation

## Syntax

```
setImage(annotation,source)
```

## Description

`setImage(annotation,source)` inserts an image from the clipboard or an image file into an annotation.

## Examples

### Add Image Annotation to Chart

Add an annotation in the chart `ch`. Use the file `myImageFile.png`, which is located in the folder `myfolder/annotation_images`, as the image for the annotation.

```
annotation = Stateflow.Annotation(ch);
setImage(annotation, ...
    fullfile('myfolder','annotation_images','myImageFile.png');
```

## Input Arguments

### `annotation` — Annotation

`Stateflow.Annotation` object

Annotation, specified as a `Stateflow.Annotation` object.

### `source` — Source of image

character array | `'clipboard'` | `''`

Source of the image, specified as a character array that contains the full path and name of an image file. Alternatively, to insert an image from the clipboard, specify `'clipboard'`.

To convert an image annotation to a text annotation, specify `''`.

## See Also

**Functions**

`fullfile`

**Objects**

`Stateflow.Annotation`

**Topics**
“Overview of the Stateflow API” on page 1-2

**Introduced in R2014a**

# sinkedTransitions

Return transitions that have object as destination

## Syntax

```
transitions = sinkedTransitions(thisObject)
```

## Description

The `sinkedTransitions` function returns all inner and outer transitions that have this object as their destination.

## Arguments

| | |
|---|---|
| `thisObject` | Destination object of the returned transitions. Can be of type State, Box, or Junction. |

## Returns

| | |
|---|---|
| `transitions` | Array of all transitions whose destination is this object. |

## Examples

The following example shows how to find all transitions whose destination is the state named `steady_state`.

```
sf_car;
rt = sfroot;
ss_state = find(rt,'-isa','Stateflow.State','Name','steady_state');
sinked_trans = sinkedTransitions(ss_state);
```

**Introduced in R2012a**

# sourcedTransitions

Return transitions that have object as source

## Syntax

```
transitions = sourcedTransitions(thisObject)
```

## Description

The `sourcedTransitions` function returns all inner and outer transitions that have this object as their source.

## Arguments

| | |
|---|---|
| `thisObject` | Source object of the returned transitions. Can be of type State, Box, or Junction. |

## Returns

| | |
|---|---|
| `transitions` | Array of all transitions whose source is this object |

## Examples

The following example shows how to find all transitions whose source is the state named `steady_state`.

```
sf_car;
rt = sfroot;
ss_state = find(rt,'-isa','Stateflow.State','Name','steady_state');
sourced_trans = sourcedTransitions(ss_state);
```

#### Introduced before R2006a

# up

Return parent of object

## Syntax

```
parentObject = up(thisObject)
```

## Description

The up function returns the parent of this object.

## Arguments

| | |
|---|---|
| thisObject | Object for which to return parent (containing) object |

## Returns

| | |
|---|---|
| parentObject | Object containing thisObject |

## Examples

Assume that a chart has two states, A and B, and state A contains state B. If the object sB represents the state B, then the command returns the parent p of state B, which is state A.

```
p = up(sB)
```

## See Also

**Topics**

“Overview of the Stateflow API” on page 1-2
“Access Objects in Your Stateflow Chart” on page 1-6

**Introduced before R2006a**

# view

**Package:** `Stateflow`

Display object in editing environment

## Syntax

```
view(object)
```

## Description

`view(object)` displays an object in its editing environment, such as the Stateflow, MATLAB, and Simulink Editors.

- The Stateflow Editor displays the contents of these objects:
  - `Stateflow.AtomicBox`
  - `Stateflow.AtomicSubchart`
  - `Stateflow.Box` with `IsSubchart` set to `true`
  - `Stateflow.Chart`
  - `Stateflow.Function` with `IsSubchart` set to `true`
  - `Stateflow.State` with `IsSubchart` set to `true`
- The Stateflow Editor shows these objects in their subviewer:
  - `Stateflow.Annotation`
  - `Stateflow.Box` with `IsSubchart` set to `false`
  - `Stateflow.Function` with `IsSubchart` set to `false`
  - `Stateflow.State` with `IsSubchart` set to `false`
  - `Stateflow.Junction`
  - `Stateflow.Transition`
- The MATLAB Editor displays the code for `Stateflow.EMChart` and `Stateflow.EMFunction` objects.
- The Simulink Editor displays the block diagram for `Stateflow.SimulinkBasedState` and `Stateflow.SLFunction` objects.
- The Truth Table Editor displays the content of `Stateflow.TruthTable` and `Stateflow.TruthTableChart` objects.
- The State Transition Table Editor displays the content of `Stateflow.StateTransitionTableChart` objects.
- The Model Explorer displays the properties of these objects:
  - `Stateflow.Data`
  - `Stateflow.Event`
  - `Stateflow.Message`

## Examples

### Display State in Chart

Open a Simulink model called `myModel`. Suppose that the model contains a Stateflow chart with a state named `A`.

```
open_system('myModel')
```

Access the `Simulink.Root` object at the top level of the Stateflow hierarchy.

```
rt = sfroot;
```

Find the state named `A`.

```
st = find(rt,'-isa','Stateflow.State','Name','A');
```

Display the state in the Stateflow Editor.

```
view(st);
```

## Input Arguments

### `object` — Object to view

`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object | ...

Object to view, specified as a Stateflow API object of one of these types:

- `Stateflow.Annotation`
- `Stateflow.AtomicBox`
- `Stateflow.AtomicSubchart`
- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.Data`
- `Stateflow.EMChart`
- `Stateflow.EMFunction`
- `Stateflow.Event`
- `Stateflow.Function`
- `Stateflow.Junction`
- `Stateflow.Message`
- `Stateflow.SimulinkBasedState`
- `Stateflow.SLFunction`
- `Stateflow.State`
- `Stateflow.StateTransitionTableChart`
- `Stateflow.Transition`
- `Stateflow.TruthTable`

- `Stateflow.TruthTableChart`

## See Also

`fitToView` | `highlight` | `zoomIn` | `zoomOut`

**Topics**

“Overview of the Stateflow API” on page 1-2

**Introduced before R2006a**

# zoomIn

**Package:** `Stateflow`

Zoom in on Stateflow chart

## Syntax

```
zoomIn(editor)
```

## Description

`zoomIn(editor)` increases the magnification level of the `Stateflow.Editor` object `editor` for a chart.

## Examples

### Zoom in on Stateflow Chart

Increase the magnification level of a nonempty chart `ch`.

```
ed = ch.Editor;
zoomIn(ed)
```

If the magnification level for the chart was initially 100%, this command increases it to 130%.

## Input Arguments

### `editor` — Editor for chart

`Stateflow.Editor` object

Editor for a chart, specified as a `Stateflow.Editor` object. The `Stateflow.Editor` object provides access to the graphical aspects of a chart. For example, to access the `Stateflow.Editor` object for a `Stateflow.Chart` object `ch`, enter:

```
ed = ch.Editor;
```

## Algorithms

The `zoomIn` function modifies the `ZoomFactor` property of the `Stateflow.Editor` object. The property is limited to a minimum of `0.5` and a maximum of `10`. `zoomIn` multiplies `ZoomFactor` by a factor of `1.3` as long as the resulting value is in this range. Otherwise, `zoomIn` sets `ZoomFactor` to the maximum value of `10`.

## See Also

`fitToView` | `highlight` | `view` | `zoomOut`

### Topics

“Overview of the Stateflow API” on page 1-2

**Introduced before R2006a**

# zoomOut

**Package:** `Stateflow`

Zoom out on Stateflow chart

## Syntax

```
zoomOut(editor)
```

## Description

`zoomOut(editor)` reduces the magnification level of the `Stateflow.Editor` object `editor` for a chart.

## Examples

### Zoom out on Stateflow Chart

Decrease the magnification level of a nonempty chart `ch`.

```
ed = ch.Editor;
zoomOut(ed)
```

If the magnification level for the chart was initially 100%, this command decreases it to 76.9%.

## Input Arguments

### `editor` — Editor for chart

`Stateflow.Editor` object

Editor for a chart, specified as a `Stateflow.Editor` object. The `Stateflow.Editor` object provides access to the graphical aspects of a chart. For example, to access the `Stateflow.Editor` object for a `Stateflow.Chart` object `ch`, enter:

```
ed = ch.Editor;
```

## Algorithms

The `zoomOut` function modifies the `ZoomFactor` property of the `Stateflow.Editor` object. The property is limited to a minimum of `0.5` and a maximum of `10`. `zoomOut` divides `ZoomFactor` by a factor of `1.3` as long as the resulting value is in this range. Otherwise, `zoomOut` sets `ZoomFactor` to the minimum value of `0.5`.

## See Also

`fitToView` | `highlight` | `view` | `zoomIn`

### Topics

“Overview of the Stateflow API” on page 1-2

**Introduced before R2006a**

# API Property Reference

# List of Stateflow API Properties

The following reference tables for Stateflow API properties have these columns:

- **Property** — The name of the property. To access or set a property value, use its name in dot notation along with a Stateflow object. Properties with multiple levels of hierarchy (such as the `LoggingInfo` and `Props` properties of data objects) must be set individually. For more information, see “Modify Properties and Call Functions of Stateflow Objects” on page 1-10.
- **Access** — An access type for the property.
  - RW (read/write): You can access or set the value of these properties by using the Stateflow API.
  - RO (read-only): These properties are set by the Stateflow software.
- **Description** — A description of the property.
- **Objects** — The types of objects that have this property. The object types are listed as: Annotation (A on page 2-2), Atomic Box (AB on page 2-8), Atomic Subchart (AS on page 2-13), Box (B on page 2-20), Chart (C on page 2-28), Clipboard (CB on page 2-24), Data (D on page 2-38), Event (E on page 2-63), Editor (ED on page 2-48), Graphical Function (GF on page 2-67), Junction (J on page 2-72), Machine (M on page 2-76), MATLAB Function (MF on page 2-57), MATLAB Function Block (MFB on page 2-50), Message (MS on page 2-80), State (S on page 2-98), Simulink Based State (SBS on page 2-87), Simulink Function (SF on page 2-94), State Transition Table (STT on page 2-108), Transition (T on page 2-117), Truth Table Block (TTB on page 2-131), and Truth Table Function (TTF on page 2-125).

## Active State Output

| Property | Access | Description | Objects |
|---|---|---|---|
| `DoNotAutogenerateEnum` | RW | Whether to define the enumerated data type for the active state data output manually, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `OutputMonitoringMode` property is `'ChildActivity'` or `'LeafStateActivity'`. For more information, see “Define State Activity Enumeration Type”. | C on page 2-28 S on page 2-98 STT on page 2-108 |
| `EnumTypeName` | RW | Name of the enumerated data type for the active state data object, specified as a character vector. This property applies only when the `OutputMonitoringMode` property is `'ChildActivity'` or `'LeafStateActivity'`. For more information, see “Enum Name”. | C on page 2-28 S on page 2-98 STT on page 2-108 |
| `HasOutputData` | RW | Whether to create an active state data output port for the object, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Monitor State Activity Through Active State Data”. | AS on page 2-13 C on page 2-28 SBS on page 2-87 S on page 2-98 STT on page 2-108 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `OutputData` | RO | Active state data object, specified as a `Stateflow.Data` object. This property applies only when the `HasOutputData` property is `true`. | AS on page 2-13 C on page 2-28 SBS on page 2-87 S on page 2-98 STT on page 2-108 |
| `OutputMonitoringMode` | RW | Monitoring mode for the active state output data, specified as `'SelfActivity'`, `'ChildActivity'`, or `'LeafStateActivity'`.<br><br>For charts, the options are `'ChildActivity'` or `'LeafStateActivity'`.<br><br>For atomic subcharts and Simulink based states, the only option is `'SelfActivity'`. | AS on page 2-13 C on page 2-28 SBS on page 2-87 S on page 2-98 STT on page 2-108 |
| `OutputPortName` | RW | Name of the active state data object, specified as a character vector. This property applies only when the `HasOutputData` property is `true`. | AS on page 2-13 C on page 2-28 SBS on page 2-87 S on page 2-98 STT on page 2-108 |
| `OutputState` | RO | State or chart monitored by the data object, specified as an empty array or a `Stateflow.AtomicSubchart`, `Stateflow.Chart`, `Stateflow.SimulinkBasedState`, `Stateflow.State`, or `Stateflow.StateTransitionTableChart` object. For more information, see “Monitor State Activity Through Active State Data”. | D on page 2-38 |

## C Action Language

| Property | Access | Description | Objects |
|---|---|---|---|
| `EnableBitOps` | RW | Whether to use bit operations in state and transition actions, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only to charts and state transition tables that use C as the action language. For more information, see “Enable C-bit operations”. | C on page 2-28 STT on page 2-108 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `StrongDataTypingWithSimulink` | RW | Whether to use strong data typing when the chart or state transition table interfaces with Simulink input and output signals, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only to charts and state transition tables that use C as the action language. For more information, see “Use strong data typing with Simulink I/O”. | C on page 2-28 STT on page 2-108 |
| `UserSpecifiedStateTransitionExecutionOrder` | RW | Whether to use explicit ordering of parallel states and transitions, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only to charts that use C as the action language. For more information, see “User-specified state/transition execution order”. | C on page 2-28 |

## Callbacks

| Property | Access | Description | Objects |
|---|---|---|---|
| `ClickFcn` | RW | Callback on click, specified as a character vector. This callback contains MATLAB code to execute when to execute when you click the annotation. | A on page 2-2 |
| `DeleteFcn` | RW | Callback at delete, specified as a character vector. This callback contains MATLAB code to execute before you delete the annotation. | A on page 2-2 |
| `LoadFcn` | RW | Callback at model load, specified as a character vector. This callback contains MATLAB code to execute when you load the model that contains the annotation. | A on page 2-2 |
| `UseDisplayTextAsClickCallback` | RW | Whether to use the annotation text as a callback, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is enabled, the contents of the `Text` property is used as the callback when you click the annotation. | A on page 2-2 |

## Chart Initialization

| Property | Access | Description | Objects |
|---|---|---|---|
| `ExecuteAtInitialization` | RW | Whether to initialize the state configuration of the chart or state transition table at time zero instead of at the first input event, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Execution of a Chart at Initialization”. | C on page 2-28 STT on page 2-108 |
| `InitializeOutput` | RW | Whether to initialize the output data every time the chart or state transition table wakes up, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Initialize outputs every time chart wakes up”. | C on page 2-28 STT on page 2-108 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `StatesWhenEnabling` | RW | Behavior of the states when a function-call input event reenables the chart or state transition table, specified as one of these values:<br>• `''` — The chart or state transition table does not contain function-call input events.<br>• `'held'` — The chart or state transition table maintains the most recent values of the states.<br>• `'reset'` — The chart or state transition table reverts to the initial conditions of the states.<br>For more information, see “Control States in Charts Enabled by Function-Call Input Events”. | C on page 2-28 STT on page 2-108 |

## Code Generation

| Property | Access | Description | Objects |
|---|---|---|---|
| `GeneratePreprocessorConditionals` | RW | Whether the generated code includes a preprocessor conditional statement for the variant conditions in the chart, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when generating code with Embedded Coder. For more information, see “Code Generation Using Variant Transitions”. | C on page 2-28 |
| `InlineOption` | RW | Appearance of the state functions, graphical function, MATLAB function, or truth table function in generated code, specified as one of these values:<br>• `'Auto'` — An internal calculation determines the appearance of the functions in generated code.<br>• `'Function'` — The functions are implemented as a separate C functions.<br>• `'Inline'` — Calls to the functions are replaced by code.<br>For more information, see “Inline State Functions in Generated Code” (Simulink Coder). | GF on page 2-67 MF on page 2-57 S on page 2-98 TTF on page 2-125 |
| `IsVariant` | RW | Whether the transition is a variant transition, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Code Generation Using Variant Transitions”. | T on page 2-117 |

## Content

| Property | Access | Description | Objects |
|---|---|---|---|
| `ActionLanguage` | RW | Action language used to program the chart or state transition table, specified as `'MATLAB'` or `'C'`. For more information, see “Differences Between MATLAB and C as Action Language Syntax”. | C on page 2-28 STT on page 2-108 |
| `ActionTable` | RW | Action table for the truth table, specified as a cell array of character vectors. | TTB on page 2-131 TTF on page 2-125 |
| `Alignment` | RW | Alignment of the annotation text, specified as `'LEFT'`, `'CENTER'`, or `'RIGHT'`. | A on page 2-2 |
| `AllowDirectFeedthrough` | RW | Whether the MATLAB Function block supports direct feedthrough semantics, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Allow direct feedthrough” (Simulink). | MFB on page 2-50 |
| `CommentText` | RW | Comment text added to the graphical object, specified as a character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge  on the graphical object, the text appears as a tooltip. When you set the `IsExplicitlyCommented` property to `false`, the value of `CommentText` reverts to `''`. | AB on page 2-8 AS on page 2-13 B on page 2-20 GF on page 2-67 J on page 2-72 MF on page 2-57 SBS on page 2-87 SF on page 2-94 S on page 2-98 T on page 2-117 TTF on page 2-125 |
| `Condition` | RO | Transition condition, specified as a character vector. The value of this property depends on the `LabelString` property for the transition. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16. | T on page 2-117 |
| `ConditionAction` | RO | Transition condition action, specified as a character vector. The value of this property depends on the `LabelString` property for the transition. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16. | T on page 2-117 |
| `ConditionTable` | RW | Condition table for the truth table, specified as a cell array of character vectors. | TTB on page 2-131 TTF on page 2-125 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `DuringAction` | RO | State `during` action, specified as a character vector. The value of this property depends on the `LabelString` property for the state. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16. This property is not supported in Moore charts. | S on page 2-98 |
| `EntryAction` | RO | State `entry` action, specified as a character vector. The value of this property depends on the `LabelString` property for the state. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16. This property is not supported in Moore charts. | S on page 2-98 |
| `ExecutionOrder` | RW | Execution order for the transition when its source is active, specified as an integer scalar. This property applies only when the `UserSpecifiedStateTransitionExecutionOrder` property of the chart that contains the transition is `true`. For more information, see “Transition Evaluation Order”. | T on page 2-117 |
| `ExitAction` | RO | State `exit` action, specified as a character vector. The value of this property depends on the `LabelString` property for the state. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16. This property is not supported in Moore charts. | S on page 2-98 |
| `FullFileName` | RO | Full file path of the Simulink model for the machine, specified as a character vector. | M on page 2-76 |
| `Interpretation` | RW | Format of the annotation text, specified as `'OFF'`, `'RICH'`, or `'TEX'`. | A on page 2-2 |
| `IsExplicitlyCommented` | RW | Whether to comment out the graphical object, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the graphical object and selecting **Comment Out**. For more information, see “Commenting Stateflow Objects in a Chart”. | AB on page 2-8 AS on page 2-13 B on page 2-20 GF on page 2-67 J on page 2-72 MF on page 2-57 SBS on page 2-87 SF on page 2-94 S on page 2-98 T on page 2-117 TTF on page 2-125 |
| `IsImage` | RO | Whether the annotation contains an image, specified as a numeric or logical 1 (`true`) or 0 (`false`). | A on page 2-2 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `IsImplicitlyCommented` | RO | Whether the graphical object is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The graphical object is implicitly commented out when you comment out a superstate in its hierarchy. | AB on page 2-8 AS on page 2-13 B on page 2-20 GF on page 2-67 J on page 2-72 MF on page 2-57 SBS on page 2-87 SF on page 2-94 S on page 2-98 T on page 2-117 TTF on page 2-125 |
| `IsLibrary` | RO | Whether the Simulink model for the machine builds a library and not an application, specified as a numeric or logical 1 (`true`) or 0 (`false`). | M on page 2-76 |
| `IsLink` | RO | Whether the atomic box or subchart is a library link, specified as a numeric or logical 1 (`true`) or 0 (`false`). | AB on page 2-8 AS on page 2-13 |
| `LabelString` | RW | Full label for the graphical object, specified as a character vector. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16. | AB on page 2-8 AS on page 2-13 B on page 2-20 GF on page 2-67 MF on page 2-57 SF on page 2-94 S on page 2-98 T on page 2-117 TTF on page 2-125 |
| `Language` | RW | Action language used to program the truth table, specified as `'MATLAB'` or `'C'`. The option `'C'` is supported only in truth tables in charts that use C as the action language. For more information, see “Differences Between MATLAB and C as Action Language Syntax”. | TTF on page 2-125 |
| `MooreAction` | RO | State action in a Moore chart, specified as a character vector. The value of this property depends on the `LabelString` property for the state. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16. This property is supported only in Moore charts. For more information, see “Design Rules for Moore Charts”. | S on page 2-98 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Name` | RO for all `Stateflow.Machine` objects.<br><br>RW for all other objects. | Name of the object, specified as a character vector. | AB on page 2-8 AS on page 2-13 B on page 2-20 C on page 2-28 GF on page 2-67 M on page 2-76 MF on page 2-57 SBS on page 2-87 SF on page 2-94 S on page 2-98 STT on page 2-108 TTB on page 2-131 TTF on page 2-125 MFB on page 2-50 |
| `OnAction` | RO | State `on` actions, specified as a cell array of character vectors in the form<br><br>`{'trigger1','action1',...,'triggerN','actionN'}`<br><br>The value of this property depends on the `LabelString` property for the state. For more information, see “Specify Labels in States and Transitions Programmatically” on page 1-16. This property is not supported in Moore charts. | S on page 2-98 |
| `PlainText` | RO | Annotation text without formatting, specified as a character vector. | A on page 2-2 |
| `Script` | RW | Code for the MATLAB function or MATLAB Function block, specified as a character vector. To enter multiple lines of code, you can:<br><br>• Call the MATLAB function `sprintf` and use the escape sequence `\n` to insert newline characters:<br><br>`str = sprintf('function y=f(x)\ny=x+1;\nend');`<br>`mfb.Script = str;`<br><br>• Enter a concatenated text expression that uses the integer 10 as the ASCII equivalent of a newline character:<br><br>`str = ['function y=f(x)',10, ...`<br>`'y=x+1;',10, ...`<br>`'end'];`<br>`mfb.Script = str;` | MF on page 2-57 MFB on page 2-50 |

| Property | Access | Description | Objects |
|---|---|---|---|
| StateMachineType | RW | State machine semantics implemented by the chart or state transition table, specified as 'Classic', 'Mealy', or 'Moore'. For more information, see "Overview of Mealy and Moore Machines". | C on page 2-28 STT on page 2-108 |
| SupportVariableSizing | RW | Whether the chart, state transition table, truth table, or MATLAB Function block supports variable-size data, specified as a numeric or logical 1 (true) or 0 (false). Only variable-size data can change dimension during simulation. For more information, see "Declare Variable-Size Inputs and Outputs" (Simulink). | C on page 2-28 STT on page 2-108 TTB on page 2-131 MFB on page 2-50 |
| Text | RW | Text for the annotation, specified as a character vector. | A on page 2-2 |
| TransitionAction | RO | Transition action, specified as a character vector. The value of this property depends on the LabelString property for the transition. For more information, see "Specify Labels in States and Transitions Programmatically" on page 1-16. | T on page 2-117 |
| Trigger | RO | Transition trigger, specified as a character vector. The value of this property depends on the LabelString property for the transition. For more information, see "Specify Labels in States and Transitions Programmatically" on page 1-16. | T on page 2-117 |
| Type | RW | Type of junction, specified as one of these values:<br>• 'CONNECTIVE' — Connective junction that represents a decision point in a transition path<br>• 'HISTORY' — History junction that records the activity of substates inside a superstate | J on page 2-72 |

## Data Size

| Property | Access | Description | Objects |
|---|---|---|---|
| CompiledSize | RO | Data size as determined by the compiler, specified as a character vector. | D on page 2-38 MS on page 2-80 |
| Props.Array.FirstIndex | RW | Index for the first element of the array data object, specified as a character vector. This property applies only to array data in charts that use C as the action language. For more information, see "Save Final Value to Base Workspace". | D on page 2-38 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Props.Array.IsDynamic` | RW | Whether the data object has variable size, specified as a numeric or logical 1 (`true`) or 0 (`false`). Only variable-size data can change size during simulation. This property applies only to input and output data and is equivalent to the **Variable Size** check box in the Data properties dialog box. For more information, see “Declare Variable-Size Data in Stateflow Charts”. | D on page 2-38 |
| `Props.Array.Size` | RW | Size of the data object or message data, specified as a character vector. For more information, see “Specify Size of Stateflow Data”. | D on page 2-38 MS on page 2-80 |

## Data Type

| Property | Access | Description | Objects |
|---|---|---|---|
| `CompiledType` | RO | Data type as determined by the compiler, specified as a character vector. | D on page 2-38 MS on page 2-80 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `DataType` | RW | Type of the data object or message data, specified as a character vector that depends on the `Props.Type.Method` property:<br>• If the `Props.Type.Method` property is `'Inherit'`, the value of this property is `'Inherit: From definition in chart'` for local data and `'Inherit: Same as Simulink'` for input data, output data, parameter data, and messages.<br>• If the `Props.Type.Method` property is `'Built-in'`, you can specify this property with one of these options:<br>  • `'double'`<br>  • `'single'`<br>  • `'int8'`<br>  • `'int16'`<br>  • `'int32'`<br>  • `'int64'`<br>  • `'uint8'`<br>  • `'uint16'`<br>  • `'uint32'`<br>  • `'uint64'`<br>  • `'boolean'`<br>  • `'ml'` (Supported only in charts that use C as the action language)<br>  • `'string'` (Supported only in charts that use C as the action language)<br>• Otherwise, the `Props.Type` properties determine the value of this property.<br>For more information, see the section Add Data on page 1-0 in "Create Charts by Using the Stateflow API" on page 1-19. | D on page 2-38 MS on page 2-80 |
| `Props.Type.BusObject` | RW | Name of the `Simulink.Bus` object that defines the data object or message data, specified as a character vector. This property applies only when the `Props.Type.Method` property is `'Bus Object'`. For more information, see "Access Bus Signals Through Stateflow Structures". | D on page 2-38 MS on page 2-80 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Props.Type.EnumType` | RW | Name of the enumerated type that defines the data object or message data, specified as a character vector. This property applies only when the `Props.Type.Method` property is `'Enumerated'`. For more information, see “Reference Values by Name by Using Enumerated Data”. | D on page 2-38 MS on page 2-80 |
| `Props.Type.Expression` | RW | Expression that evaluates to the data type of the data object or message data, specified as a character vector. This property applies only when the `Props.Type.Method` property is `'Expression'`. For more information, see “Specify Data Properties by Using MATLAB Expressions”. | D on page 2-38 MS on page 2-80 |
| `Props.Type.Fixpt.Bias` | RW | Bias of the fixed-point data object or message data, specified as a character vector. This property applies only to fixed-point data when the `Props.Type.Fixpt.ScalingMode` property is `'Slope and bias'`. For more information, see “Fixed-Point Data in Stateflow Charts”. | D on page 2-38 MS on page 2-80 |
| `Props.Type.Fixpt.FractionLength` | RW | Fraction length, in bits, of the fixed-point data object or message data, specified as a character vector. This property applies only to fixed-point data when the `Props.Type.Fixpt.ScalingMode` property is `'Binary point'`. For more information, see “Fixed-Point Data in Stateflow Charts”. | D on page 2-38 MS on page 2-80 |
| `Props.Type.Fixpt.Lock` | RW | Whether to prevent replacement of the fixed-point type of the data object or message data with an autoscaled type chosen by the Fixed-Point Tool (Fixed-Point Designer), specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Autoscaling Using the Fixed-Point Tool” (Fixed-Point Designer). | D on page 2-38 MS on page 2-80 |
| `Props.Type.Fixpt.ScalingMode` | RW | Method for scaling the fixed-point data object or message data, specified as `'Binary point'`, `'Slope and bias'`, or `'None'`. This property applies only when the `Props.Type.Method` property is `'Fixed point'`. For more information, see “Fixed-Point Data in Stateflow Charts”. | D on page 2-38 MS on page 2-80 |
| `Props.Type.Fixpt.Slope` | RW | Slope of the fixed-point data object or message data, specified as a character vector. This property applies only to fixed-point data when the `Props.Type.Fixpt.ScalingMode` property is `'Slope and bias'`. For more information, see “Fixed-Point Data in Stateflow Charts”. | D on page 2-38 MS on page 2-80 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Props.Type.Method` | RW | Method for setting the type of the data object or message data, specified as a character vector.<br>• For local, input, output, or parameter data, use `'Inherited'`, `'Built-in'`, `'Bus Object'`, `'Enumerated'`, `'Expression'`, or `'Fixed point'`.<br>• For constant data, use `'Built-in'`, `'Expression'`, or `'Fixed point'`.<br>• For data store memory data, use `'Inherited'`.<br>• For messages, use `'Inherited'`, `'Built-in'`, `'Bus Object'`, `'Enumerated'`, `'Expression'`, or `'Fixed point'`.<br>This property is equivalent to the **Mode** field of the Data Type Assistant in the Model Explorer, the Data properties dialog box, or the Message properties dialog box. For more information, see “Specify Type of Stateflow Data”. | D on page 2-38 MS on page 2-80 |
| `Props.Type.Signed` | RW | Signedness of the fixed-point data object or message data, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `Props.Type.Method` property is `'Fixed point'`. For more information, see “Fixed-Point Data in Stateflow Charts”. | D on page 2-38 MS on page 2-80 |
| `Props.Type.WordLength` | RW | Word length, in bits, of the fixed-point data object or message data, specified as a character vector. This property applies only when the `Props.Type.Method` property is `'Fixed point'`. For more information, see “Fixed-Point Data in Stateflow Charts”. | D on page 2-38 MS on page 2-80 |

## Debugging

<table>
<tr><th>Property</th><th>Access</th><th>Description</th><th>Objects</th></tr>
<tr><td>Debug.Animation.Delay</td><td>RW</td><td>Delay that the chart animation uses for highlighting each transition segment in the machine, specified as a scalar. These values correspond to the settings of the Animation Speed drop-down list in the Debug tab:<br><br><table><tr><th>Delay Value</th><th>Animation Speed</th></tr><tr><td>0.5</td><td>Slow</td></tr><tr><td>0.2</td><td>Medium</td></tr><tr><td>0</td><td>Fast</td></tr><tr><td>-1</td><td>Lightning Fast</td></tr></table><br>This property applies only when the Debug.Animation.Enable property of the machine is true.</td><td>M on page 2-76</td></tr>
<tr><td>Debug.Animation.Enabled</td><td>RW</td><td>Whether to animate the charts in the machine during simulation, specified as a numeric or logical 1 (true) or 0 (false). Disabling this property is equivalent to selecting None in the Animation Speed drop-down list in the Debug tab.</td><td>M on page 2-76</td></tr>
<tr><td>Debug.Animation.MaintainHighlighting</td><td>RO</td><td>Whether to maintain the highlighting of active states in the machine after the simulation ends, specified as a numeric or logical 1 (true) or 0 (false).</td><td>M on page 2-76</td></tr>
<tr><td>Debug.Breakpoints.EndBroadcast</td><td>RW</td><td>Whether to set the End of Broadcast breakpoint for the event, specified as a numeric or logical 1 (true) or 0 (false). For more information, see “Debugger Breakpoints”.</td><td>E on page 2-63</td></tr>
<tr><td>Debug.Breakpoints.OnDuring</td><td>RW</td><td>Whether to set the During State or During Function Call breakpoint, specified as a numeric or logical 1 (true) or 0 (false).<br><br>For truth tables, this property applies only when both the Language property of the truth table and the ActionLanguage of the chart that contains the truth table are 'C' .</td><td>AS on page 2-13 GF on page 2-67 SBS on page 2-87 S on page 2-98 TTF on page 2-125</td></tr>
<tr><td>Debug.Breakpoints.OnEntry</td><td>RW</td><td>Whether to set the On Chart Entry or On State Entry breakpoint, specified as a numeric or logical 1 (true) or 0 (false).</td><td>AS on page 2-13 C on page 2-28 SBS on page 2-87 S on page 2-98 STT on page 2-108</td></tr>
</table>

| Property | Access | Description | Objects |
|---|---|---|---|
| `Debug.Breakpoints.OnExit` | RW | Whether to set the `On State Exit` or `On State Exit`breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`). | AS on page 2-13 SBS on page 2-87 S on page 2-98 |
| `Debug.Breakpoints.StartBroadcast` | RW | Whether to set the `Start of Broadcast` breakpoint for the event, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Debugger Breakpoints”. | E on page 2-63 |
| `Debug.Breakpoints.WhenTested` | RW | Whether to set the `When Transition is Tested` breakpoint for the transition, specified as a numeric or logical 1 (`true`) or 0 (`false`). | T on page 2-117 |
| `Debug.Breakpoints.WhenValid` | RW | Whether to set the `When Transition is Valid` breakpoint for the transition, specified as a numeric or logical 1 (`true`) or 0 (`false`). | T on page 2-117 |
| `Debug.Watch` | RW | Whether to track the value of the data object in the Breakpoints and Watch window, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “View Data in the Breakpoints and Watch Window”. | D on page 2-38 |
| `OverSpecDiagnostic` | RW | Level of diagnostic action when the truth table is overspecified, specified as `'Error'`, `'Warning'`, or `'None'`. For more information, see “Correct Overspecified and Underspecified Truth Tables”. | TTB on page 2-131 TTF on page 2-125 |
| `TestPoint` | RW | Whether to set the atomic subchart, state, Simulink based state, or data object as a test point, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Monitor Test Points in Stateflow Charts”. | AS on page 2-13 D on page 2-38 SBS on page 2-87 S on page 2-98 |
| `UnderSpecDiagnostic` | RW | Level of diagnostic action when the truth table is underspecified, specified as `'Error'`, `'Warning'`, or `'None'`. For more information, see “Correct Overspecified and Underspecified Truth Tables”. | TTB on page 2-131 TTF on page 2-125 |

## Discrete and Continuous-Time Semantics

| Property | Access | Description | Objects |
|---|---|---|---|
| `ChartUpdate` | RW | Activation method for the chart, state transition table, truth table, or MATLAB Function block, specified as `'CONTINUOUS'`, `'DISCRETE'`, or `'INHERITED'`. For more information, see “Update Method”. | C on page 2-28 STT on page 2-108 TTB on page 2-131 MFB on page 2-50 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `EnableZeroCrossings` | RW | Whether to enable zero-crossing detection on state transitions in the chart or state transition table, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `ChartUpdate` property for the chart is set to `'CONTINUOUS'`. For more information, see “Disable Zero-Crossing Detection”. | C on page 2-28 STT on page 2-108 |
| `SampleTime` | RW | Sample time for activating the chart, state transition table, truth table, or MATLAB Function block, specified as a character vector. This property applies only when the `ChartUpdate` property for the chart is `'DISCRETE'`. | C on page 2-28 STT on page 2-108 TTB on page 2-131 MFB on page 2-50 |

## Exported Functions

| Property | Access | Description | Objects |
|---|---|---|---|
| `AllowGlobalAccessToExportedFunctions` | RW | Whether exported functions from the chart are globally visible in the Simulink model, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is enabled, blocks throughout the model can call functions exported from the chart without using qualified notation. This property applies only when the `ExportChartFunctions` property for the chart is `true`. | C on page 2-28 |
| `ExportChartFunctions` | RW | Whether to export chart-level functions to other blocks in the Simulink model, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Export Stateflow Functions for Reuse”. | C on page 2-28 |

## Graphical Appearance

| Property | Access | Description | Objects |
|---|---|---|---|
| `ArrowSize` | RW | For states and subcharts, size of incoming transition arrows, specified as a scalar.<br><br>For transitions, size of the transition arrow at the destination. When you change the destination of the transition, this property resets to the value of the `ArrowSize` property of the new destination. | AS on page 2-13 J on page 2-72 SBS on page 2-87 S on page 2-98 T on page 2-117 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `AutoBackgroundColor` | RW | Whether to use the default background color, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• `true` — Use the default color specified by the `ChartColor` property of the chart that contains the annotation.<br>• `false` — Use the color specified by the `BackgroundColor` property of the annotation. | A on page 2-2 |
| `AutoForegroundColor` | RW | Whether to use the default foreground color, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• `true` — Use the default color specified by the `StateLabelColor` property of the chart that contains the annotation.<br>• `false` — Use the color specified by the `ForegroundColor` property of the annotation. | A on page 2-2 |
| `BackgroundColor` | RW | Background color for the annotation, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1. This property applies only when the `AutoBackgroundColor` property is `false`. | A on page 2-2 |
| `BadIntersection` | RO | Whether a box, state, or function graphically intersects another box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`). | AB on page 2-8 AS on page 2-13 B on page 2-20 GF on page 2-67 MF on page 2-57 SBS on page 2-87 SF on page 2-94 S on page 2-98 TTF on page 2-125 |
| `ChartColor` | RW | Background color for the chart, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.<br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `ContentPreviewEnabled` | RW | Whether to display a preview of the contents of the object, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>For boxes, states, or graphical functions, this property applies only when the `IsSubchart` property is `true`. | AB on page 2-8 AS on page 2-13 B on page 2-20 GF on page 2-67 SBS on page 2-87 SF on page 2-94 S on page 2-98 |
| `Destination` | RW | Destination of the transition, specified as an empty array or a Stateflow API object of one of these types:<br><br>• `Stateflow.AtomicBox`<br>• `Stateflow.AtomicSubchart`<br>• `Stateflow.Box`<br>• `Stateflow.Junction`<br>• `Stateflow.SimulinkBasedState`<br>• `Stateflow.State` | T on page 2-117 |
| `DestinationEndPoint` | RW | Position of the transition endpoint at its destination, specified as a two-element numeric vector `[x y]` of coordinates relative to the upper left corner of the chart. | T on page 2-117 |
| `DestinationOClock` | RW | Location of the transition endpoint at its destination, specified as a scalar between 0 and 12 that describes a clock position. | T on page 2-117 |
| `DropShadow` | RW | Whether to display a drop shadow around the annotation box, specified as a numeric or logical 1 (`true`) or 0 (`false`). | A on page 2-2 |
| `Editor` | RO | Editor for the chart or state transition table, specified as a `Stateflow.Editor` object. You can use this object to control the position, size, and magnification level of the Stateflow Editor window. | C on page 2-28 STT on page 2-108 |
| `FixedHeight` | RW | Whether to fix height of the annotation box, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>• `true` — Fixes the height of the annotation box and hides content that is longer than the box.<br>• `false` — Resizes the annotation box vertically as you add content. | A on page 2-2 |

| Property | Access | Description | Objects |
|---|---|---|---|
| FixedWidth | RW | Whether to fix height of the annotation box, specified as a numeric or logical 1 (true) or 0 (false).<br>• true — Fixes the width of the annotation box and wraps text that is longer than the box.<br>• false — Resizes the annotation box horizontally as you add content. | A on page 2-2 |
| Font.Angle | RW | Font angle for the annotation text, specified as 'NORMAL' or 'ITALIC'. | A on page 2-2 |
| Font.Name | RO | Font name for the annotation text, specified as a character vector. The StateFont.Name property of the chart that contains the annotation sets the value of this property. | A on page 2-2 |
| Font.Size | RW | Font size for the annotation text, specified as a scalar. The StateFont.Size property of the chart that contains the annotation sets the initial value of this property. | A on page 2-2 |
| Font.Weight | RW | Font weight for the annotation text, specified as 'NORMAL' or 'BOLD'. | A on page 2-2 |
| FontSize | RW | Font size for the label of a box, state, function, or transition, specified as a scalar. The StateFont.Size property of the chart that contains the graphical function sets the initial value of this property. | AB on page 2-8 AS on page 2-13 B on page 2-20 GF on page 2-67 MF on page 2-57 SBS on page 2-87 SF on page 2-94 S on page 2-98 T on page 2-117 TTF on page 2-125 |
| ForegroundColor | RW | Foreground color for the annotation, specified as a three-element numeric vector of the form [red green blue] that specifies the red, green, and blue values. Each element must be in the range between 0 and 1. This property applies only when the AutoForegroundColor property is false. | A on page 2-2 |
| InternalMargins | RW | Space between the text and the border of the annotation box, specified as a four-element numeric vector of the form [left top right bottom]. | A on page 2-2 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `IsGrouped` | RW | Whether the box, function, or state is grouped, specified as a numeric or logical 1 (`true`) or 0 (`false`). When you copy and paste a grouped object, you copy not only the object but all of its contents. For more information, see "Copy and Paste by Grouping" on page 2-24. | B on page 2-20 GF on page 2-67 S on page 2-98 |
| `IsSubchart` | RW | Whether the box, function, or state is a subchart, specified as a numeric or logical 1 (`true`) or 0 (`false`). | B on page 2-20 GF on page 2-67 S on page 2-98 |
| `JunctionColor` | RW | Color for junctions in the chart, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.<br><br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |
| `LabelPosition` | RW | Position and size of the transition label, specified as a four-element numeric vector of the form `[left top width height]`. | T on page 2-117 |
| `MidPoint` | RW | Position of the midpoint of the transition, specified as a two-element numeric vector `[x y]` of coordinates relative to the upper left corner of the chart. | T on page 2-117 |
| `Position` | RW | Position and size of the graphical object, specified as a four-element numeric vector of the form `[left top width height]`. | A on page 2-2 AB on page 2-8 AS on page 2-13 B on page 2-20 GF on page 2-67 MF on page 2-57 SBS on page 2-87 SF on page 2-94 S on page 2-98 TTF on page 2-125 |
| `Position.Center` | RW | Position of the center of the junction, specified as a two-element numeric vector `[x y]` of coordinates relative to the upper left corner of the chart. | J on page 2-72 |
| `Position.Radius` | RW | Radius of the junction, specified as a scalar. | J on page 2-72 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Source` | RW | Source of the transition, specified as an empty array or a Stateflow API object of one of these types:<br>• `Stateflow.AtomicBox`<br>• `Stateflow.AtomicSubchart`<br>• `Stateflow.Box`<br>• `Stateflow.Junction`<br>• `Stateflow.SimulinkBasedState`<br>• `Stateflow.State` | T on page 2-117 |
| `SourceEndPoint` | RW | Position of the transition endpoint at its source, specified as a two-element numeric vector `[x y]` of coordinates relative to the upper left corner of the chart. | T on page 2-117 |
| `SourceOClock` | RW | Location of the transition endpoint at its source, specified as a scalar between 0 and 12 that describes a clock position. | T on page 2-117 |
| `StateColor` | RW | Color for the boxes, functions, and states in the chart, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.<br><br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |
| `StateFont.Angle` | RW | Font angle for the box, function, and state labels in the chart, specified as `'NORMAL'` or `'ITALIC'`.<br><br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |
| `StateFont.Name` | RW | Font name for the annotation, box, function, and state labels in the chart, specified as a character vector.<br><br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |
| `StateFont.Size` | RW | Initial font size for the annotation, box, function, and state labels in the chart, specified as a scalar.<br><br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `StateFont.Weight` | RW | Font weight for the box, function, and state labels in the chart, specified as `'NORMAL'` or `'BOLD'`.<br><br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |
| `StateLabelColor` | RW | Color for the box, function, and state labels in the chart, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.<br><br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |
| `TransitionColor` | RW | Color for transitions in the chart, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.<br><br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |
| `TransitionFont.Angle` | RW | Font angle for the transition labels in the chart, specified as `'NORMAL'` or `'ITALIC'`.<br><br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |
| `TransitionFont.Name` | RW | Font name for the transition labels in the chart, specified as a character vector.<br><br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |
| `TransitionFont.Size` | RW | Initial font size for the transition labels in the chart, specified as a scalar.<br><br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |
| `TransitionFont.Weight` | RW | Font weight for the transition labels in the chart, specified as `'NORMAL'` or `'BOLD'`.<br><br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `TransitionLabelColor` | RW | Color for the transition labels in the chart, specified as a three-element numeric vector of the form `[red green blue]` that specifies the red, green, and blue values. Each element must be in the range between 0 and 1.<br><br>For state transition tables, this property controls the appearance of the chart that is automatically generated for the state transition table. | C on page 2-28 STT on page 2-108 |
| `Visible` | RW | Whether the Stateflow Editor window is displaying the chart or state transition table, specified as a numeric or logical 1 (`true`) or 0 (`false`). | C on page 2-28 STT on page 2-108 |

## Hierarchy

| Property | Access | Description | Objects |
|---|---|---|---|
| `Chart` | RO | Chart that contains the object, specified as a `Stateflow.Chart` object. | A on page 2-2 AB on page 2-8 AS on page 2-13 B on page 2-20 GF on page 2-67 J on page 2-72 MF on page 2-57 SBS on page 2-87 SF on page 2-94 S on page 2-98 T on page 2-117 TTF on page 2-125 |
| `Dirty` | RW | Whether the chart, state transition table, truth table, MATLAB Function block, or the Simulink model for the Stateflow machine has changed after being opened or saved, specified as a numeric or logical 1 (`true`) or 0 (`false`). | C on page 2-28 M on page 2-76 STT on page 2-108 TTB on page 2-131 MFB on page 2-50 |
| `Iced` | RO | Whether the chart, state transition table, truth table, MATLAB Function block, or the Simulink model for the Stateflow machine is locked, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property is equivalent to the property `Locked`, but is used internally to prevent changes in the chart, block, or model during simulation. | C on page 2-28 M on page 2-76 STT on page 2-108 TTB on page 2-131 MFB on page 2-50 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Locked` | RW | Whether the chart, state transition table, truth table, MATLAB Function block, or the Simulink model for the Stateflow machine is locked, specified as a numeric or logical 1 (`true`) or 0 (`false`). Enable this property to prevent changes in the chart, block, or model. | C on page 2-28 M on page 2-76 STT on page 2-108 TTB on page 2-131 MFB on page 2-50 |
| `Machine` | RO | Machine that contains the object, specified as a `Stateflow.Machine` object. | A on page 2-2 AB on page 2-8 AS on page 2-13 B on page 2-20 C on page 2-28 D on page 2-38 E on page 2-63 GF on page 2-67 J on page 2-72 MF on page 2-57 MS on page 2-80 SBS on page 2-87 SF on page 2-94 S on page 2-98 STT on page 2-108 T on page 2-117 TTB on page 2-131 TTF on page 2-125 MFB on page 2-50 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Path` | RO | Location of the object in the model hierarchy, specified as a character vector. | A on page 2-2 AB on page 2-8 AS on page 2-13 B on page 2-20 C on page 2-28 D on page 2-38 E on page 2-63 GF on page 2-67 J on page 2-72 M on page 2-76 MF on page 2-57 MS on page 2-80 SBS on page 2-87 SF on page 2-94 S on page 2-98 STT on page 2-108 T on page 2-117 TTB on page 2-131 TTF on page 2-125 MFB on page 2-50 |
| `Subchart` | RO | Contents of the atomic box, specified as a `Stateflow.Chart` object. Use this object to add children, such as states and transitions, to the atomic box.<br><br>Contents of the atomic subchart, specified as a `Stateflow.Chart` object. Use this object to add children, such as states and transitions, to the atomic subchart. | AB on page 2-8 AS on page 2-13 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Subviewer` | RO | Subviewer for the graphical object, specified as a `Stateflow.Chart`, `Stateflow.State`, `Stateflow.Box`, or `Stateflow.Function` object. The subviewer is the chart or subchart where you can graphically view the object. | A on page 2-2 AB on page 2-8 AS on page 2-13 B on page 2-20 GF on page 2-67 J on page 2-72 MF on page 2-57 SBS on page 2-87 SF on page 2-94 S on page 2-98 T on page 2-117 TTF on page 2-125 |

## Identification

| Property | Access | Description | Objects |
|---|---|---|---|
| `Created` | RO | Date of the creation of the machine, specified as a character vector. | M on page 2-76 |
| `Creator` | RW | Creator of the machine, specified as a character vector. | M on page 2-76 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Description` | RW | Description for the object, specified as a character vector. | A on page 2-2 AB on page 2-8 AS on page 2-13 B on page 2-20 C on page 2-28 D on page 2-38 E on page 2-63 GF on page 2-67 J on page 2-72 M on page 2-76 MF on page 2-57 MS on page 2-80 SBS on page 2-87 SF on page 2-94 S on page 2-98 STT on page 2-108 T on page 2-117 TTB on page 2-131 TTF on page 2-125 MFB on page 2-50 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Document` | RW | Document link for the object, specified as a character vector. | A on page 2-2 AB on page 2-8 AS on page 2-13 B on page 2-20 C on page 2-28 D on page 2-38 E on page 2-63 GF on page 2-67 J on page 2-72 M on page 2-76 MF on page 2-57 MS on page 2-80 SBS on page 2-87 SF on page 2-94 S on page 2-98 STT on page 2-108 T on page 2-117 TTB on page 2-131 TTF on page 2-125 MFB on page 2-50 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Id` | RO | Unique identifier, specified as an integer scalar. Use this property to distinguish the object from other objects in the model. The value of this property is reassigned every time you start a new MATLAB session and may be recycled after an object is deleted. | A on page 2-2 AB on page 2-8 AS on page 2-13 B on page 2-20 C on page 2-28 D on page 2-38 E on page 2-63 GF on page 2-67 J on page 2-72 M on page 2-76 MF on page 2-57 MS on page 2-80 SBS on page 2-87 SF on page 2-94 S on page 2-98 STT on page 2-108 T on page 2-117 TTB on page 2-131 TTF on page 2-125 MFB on page 2-50 |
| `Modified` | RW | Record of modifications to the machine, specified as a character vector. | M on page 2-76 |
| `SSIdNumber` | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the object from other objects in the model. | AB on page 2-8 AS on page 2-13 D on page 2-38 GF on page 2-67 J on page 2-72 MF on page 2-57 MS on page 2-80 SBS on page 2-87 SF on page 2-94 S on page 2-98 T on page 2-117 TTF on page 2-125 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Tag` | RW | User-defined tag for the object, specified as data of any type. | A on page 2-2 AB on page 2-8 AS on page 2-13 B on page 2-20 C on page 2-28 D on page 2-38 E on page 2-63 GF on page 2-67 J on page 2-72 M on page 2-76 MF on page 2-57 MS on page 2-80 SBS on page 2-87 SF on page 2-94 S on page 2-98 STT on page 2-108 T on page 2-117 TTB on page 2-131 TTF on page 2-125 MFB on page 2-50 |
| `Version` | RW | Version of the machine, specified as a character vector. | M on page 2-76 |

## Integer and Fixed-Point Data

| Property | Access | Description | Objects |
|---|---|---|---|
| `EmlDefaultFimath` | RW | Default `fimath` properties for the chart, state transition table, truth table, or MATLAB function, specified as one of these values:<br>• `'Same as MATLAB Default'` — Use the same `fimath` properties as the current default `fimath` object.<br>• `'Other:UserSpecified'` — Use the `InputFimath` property to specify the default `fimath` object.<br>For charts and state transition tables, this property applies only when the `ActionLanguage` property is `'MATLAB'`.<br>For MATLAB functions, this property applies only when the `ActionLanguage` of the chart that contains the function is `'C'`.<br>For truth table functions, this property applies only when the `Language` property of the truth table is `'MATLAB'` and the `ActionLanguage` of the chart that contains the truth table is `'C'`. | C on page 2-28 MF on page 2-57 STT on page 2-108 TTB on page 2-131 TTF on page 2-125 MFB on page 2-50 |
| `InputFimath` | RW | Default `fimath` object, specified as a character vector. When the `EmlDefaultFimath` property of the chart, state transition table, truth table, or MATLAB function is `'Other:UserSpecified'`, you can use this property to:<br>• Enter an expression that constructs a `fimath` object.<br>• Enter the variable name for a `fimath` object in the MATLAB or model workspace.<br>For charts and state transition tables, this property applies only when the `ActionLanguage` property is `'MATLAB'`.<br>For MATLAB functions, this property applies only when the `ActionLanguage` of the chart that contains the function is `'C'`.<br>For truth table functions, this property applies only when the `Language` property of the truth table is `'MATLAB'` and the `ActionLanguage` of the chart that contains the truth table is `'C'`. | C on page 2-28 MF on page 2-57 STT on page 2-108 TTB on page 2-131 TTF on page 2-125 MFB on page 2-50 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `SaturateOnIntegerOverflow` | RW | Whether the data in the chart, state transition table, truth table, or MATLAB function saturates on integer overflow, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is disabled, the data wraps on integer overflow. For more information, see "Saturate on integer overflow" (Simulink).<br><br>For MATLAB functions, this property applies only when the `ActionLanguage` of the chart that contains the function is `'C'`.<br><br>For truth table functions, this property applies only when the `Language` property of the truth table is `'MATLAB'` and the `ActionLanguage` of the chart that contains the truth table is `'C'`. | C on page 2-28 MF on page 2-57 STT on page 2-108 TTB on page 2-131 TTF on page 2-125 MFB on page 2-50 |
| `TreatAsFi` | RW | Inherited Simulink signals to treat as Fixed-Point Designer `fi` objects, specified as one of these values:<br><br>• `'Fixed-point'` — The chart, state transition table, truth table, or MATLAB Function block treats all fixed-point inputs as `fi` objects.<br>• `'Fixed-point & Integer'` — The chart, state transition table, truth table, or MATLAB Function block treats all fixed-point and integer inputs as `fi` objects.<br><br>For charts and state transition tables, this property applies only when the `ActionLanguage` property is `'MATLAB'`. | C on page 2-28 STT on page 2-108 TTB on page 2-131 MFB on page 2-50 |

## Interface

| Property | Access | Description | Objects |
|---|---|---|---|
| `InitializeMethod` | RW | Method for initializing the value of the data object or message data, specified as a character vector that depends on the scope of the data or message:<br><br>• For local and output data and messages, use `'Expression'` or `'Parameter'`.<br>• For constant data, use `'Expression'`.<br>• For input data and messages, parameters, and data store memory, use `'Not Needed'`. | D on page 2-38 MS on page 2-80 |
| `Inputs` | RO | Input arguments of the MATLAB Function block, specified as an array of `Stateflow.Data` objects. | MFB on page 2-50 |
| `Name` | RW | Name of the data object, event, or message, specified as a character vector. | D on page 2-38 E on page 2-63 MS on page 2-80 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Outputs` | RO | Output arguments of the MATLAB Function block, specified as an array of `Stateflow.Data` objects. | MFB on page 2-50 |
| `Port` | RW | Port index for the data object, event, or message specified as an integer scalar. This property applies only to input and output data, events, and messages. | D on page 2-38 E on page 2-63 MS on page 2-80 |
| `Priority` | RW | Priority for the message, specified as a character vector. If two distinct messages occur at the same time, this property determines which message is processed first. A smaller numeric value indicates a higher priority. This property applies only to local and output messages in discrete-event charts. For more information, see “Create Custom Queuing Systems Using Discrete-Event Stateflow Charts” (SimEvents). | MS on page 2-80 |
| `Props.Complexity` | RW | Whether the data object or message data accepts complex values, specified as `'On'` or `'Off'`. For more information, see “Complex Data in Stateflow Charts”. | D on page 2-38 MS on page 2-80 |
| `Props.InitialValue` | RW | Initial value of the data object or message data, specified as a character vector. | D on page 2-38 MS on page 2-80 |
| `Props.Range.Maximum` | RW | Maximum value for the data object, specified as a character vector. For more information, see “Limit Range”. | D on page 2-38 |
| `Props.Range.Minimum` | RW | Minimum value for the data object, specified as a character vector. For more information, see “Limit Range”. | D on page 2-38 |
| `Props.ResolveToSignalObject` | RW | Whether the data object resolves to a `Simulink.Signal` object that you define in the model or base workspace, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Resolve Data Properties from Simulink Signal Objects”. | D on page 2-38 |
| `Props.Unit.Name` | RW | Unit of measurement for the data object, specified as a character vector. This property applies only to data in charts that use C as the action language. For more information, see “Specify Units for Stateflow Data”. | D on page 2-38 |
| `SaveToWorkspace` | RW | Whether to save the value of the data object to a variable of the same name in the MATLAB base workspace at the end of the simulation, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only to data in charts that use C as the action language. For more information, see “Save Final Value to Base Workspace”. | D on page 2-38 |

| Property | Access | Description | Objects |
|---|---|---|---|
| `Scope` | RW | Scope of the data object, event, or message, specified as one of these values:<br>• `'Local'`<br>• `'Input'`<br>• `'Output'`<br>• `'Constant'`<br>• `'Parameter'`<br>• `'Data Store Memory'`<br>• `'Temporary'`<br>• `'Imported'`<br>• `'Exported'` | D on page 2-38 E on page 2-63 MS on page 2-80 |
| `Trigger` | RW | Type of trigger associated with the event, specified as a character vector that depends on the scope of the data:<br>• For input events, use `'Function call'`, `'Rising'`, `'Falling'`, or `'Either'`.<br>• For output events, use `'Function call'` or `'Either'`.<br>This property does not apply to local events. For more information, see “Trigger”. | E on page 2-63 |
| `Tunable` | RW | Whether the data object is a tunable parameter, specified as a numeric or logical 1 (`true`) or 0 (`false`). Only tunable parameters can be modified during simulation. This property applies only to parameter data. | D on page 2-38 |
| `UpdateMethod` | RW | Method for updating data object, specified as `'Discrete'` or `'Continuous'`. This property applies only when the `ChartUpdate` property of the chart that contains the data is `'CONTINUOUS'`. For more information, see “Continuous-Time Modeling in Stateflow”. | D on page 2-38 |

## Queue

| Property | Access | Description | Objects |
|---|---|---|---|
| `MessagePriorityOrder` | RW | Type of priority queue for the message, specified as one of these values:<br>• `'Ascending'` — Messages are received in ascending order of the message data value.<br>• `'Descending'` — Messages are received in descending order of the message data value.<br>This property applies only when the `QueueType` property of the message is `'Priority'`. For more information, see “Queue Type”. | MS on page 2-80 |
| `QueueCapacity` | RW | Length of the internal queue for the message, specified as an integer scalar. This property applies only to local messages and to input messages that have `UseInternalQueue` set to `true`. For more information, see “Queue Capacity”. | MS on page 2-80 |
| `QueueOverflowDiagnostic` | RW | Level of diagnostic action when the number of incoming messages exceeds the queue capacity for the message, specified as `'Error'`, `'Warning'`, or `'None'`. This property applies only to local messages and to input messages that have `UseInternalQueue` set to `true`. For more information, see “Queue Overflow Diagnostic”. | MS on page 2-80 |
| `QueueType` | RW | Order in which messages are removed from the receiving queue, specified as one of these values:<br>• `'FIFO'` — First in, first out.<br>• `'LIFO'` — Last in, first out.<br>• `'Priority'` — Remove messages according to the value in the data field. To specify the order, use the `MessagePriorityOrder` property for the message.<br>This property applies only to local messages and to input messages that have `UseInternalQueue` set to `true`. For more information, see “Queue Type”. | MS on page 2-80 |
| `UseInternalQueue` | RW | Whether the Stateflow chart maintains an internal receiving queue for the input message, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only to input messages. For more information, see “Use Internal Queue”. | MS on page 2-80 |

## Signal Logging

| Property | Access | Description | Objects |
|---|---|---|---|
| `LoggingInfo.DataLogging` | RW | Whether to enable signal logging for the atomic subchart, state, Simulink based state, or data object, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Log Simulation Output for States and Data”. | AS on page 2-13 D on page 2-38 SBS on page 2-87 S on page 2-98 |
| `LoggingInfo.DecimateData` | RW | Whether to limit the amount of logged data, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is `true`, signal logging skips samples by using the interval size specified by the `LoggingInfo.Decimation` property. | AS on page 2-13 D on page 2-38 SBS on page 2-87 S on page 2-98 |
| `LoggingInfo.Decimation` | RW | Decimation interval, specified as an integer scalar. The default value of 2 means that the chart logs every other sample. | AS on page 2-13 D on page 2-38 SBS on page 2-87 S on page 2-98 |
| `LoggingInfo.LimitDataPoints` | RW | Whether to limit the number of data points to log, specified as a numeric or logical 1 (`true`) or 0 (`false`). When this property is `true`, signal logging limits the number of data points by using the value specified by the `LoggingInfo.MaxPoints` property. | AS on page 2-13 D on page 2-38 SBS on page 2-87 S on page 2-98 |
| `LoggingInfo.LoggingName` | RW | Custom signal name used for logging the atomic subchart, state, Simulink based state, or data object, specified as a character vector. This property applies only when the `LoggingInfo.NameMode` property is `'Custom'`. | AS on page 2-13 D on page 2-38 SBS on page 2-87 S on page 2-98 |
| `LoggingInfo.MaxPoints` | RW | Maximum number of data points to log, specified as an integer scalar. The default value of `5000` means the chart logs the last 5000 data points generated by the simulation. | AS on page 2-13 D on page 2-38 SBS on page 2-87 S on page 2-98 |
| `LoggingInfo.NameMode` | RW | Source of the signal name used to log the atomic subchart, state, Simulink based state, or data object, specified as one of these values:<br>• `'SignalName'` — Use the name of the atomic subchart, state, Simulink based state, or data object.<br>• `'Custom'` — Use the custom signal name specified by the `LoggingInfo.LoggingName` property. | AS on page 2-13 D on page 2-38 SBS on page 2-87 S on page 2-98 |

## State Decomposition

| Property | Access | Description | Objects |
|---|---|---|---|
| `Decomposition` | RW | Decomposition of substates at the top level of containment in the chart or state, specified as `'EXCLUSIVE_OR'` or `'PARALLEL_AND'`. For more information, see “Specify Substate Decomposition”. | C on page 2-28 S on page 2-98 |
| `ExecutionOrder` | RW | Execution order for the atomic subchart, state, or Simulink based state in parallel (AND) decomposition, specified as an integer scalar. This property applies only when both of these conditions are satisfied:<br>• The `Type` property is `'AND'`.<br>• The `UserSpecifiedStateTransitionExecutionOrder` property of the chart that contains the atomic subchart, state, or Simulink based state is `true`. | AS on page 2-13 SBS on page 2-87 S on page 2-98 |
| `Type` | RO | Decomposition of sibling states, specified as `'OR'` or `'AND'`. The atomic subchart, state, or Simulink based state inherits this property from the `Decomposition` property of its parent state or chart. | AS on page 2-13 SBS on page 2-87 S on page 2-98 |

## Super Step Semantics

| Property | Access | Description | Objects |
|---|---|---|---|
| `EnableNonTerminalStates` | RW | Whether to enable super step semantics for the chart or state transition table, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see “Super Step Semantics”. | C on page 2-28 STT on page 2-108 |
| `NonTerminalMaxCounts` | RW | Maximum number of transitions the chart or state transition table can take in one super step, specified as an integer scalar. This property applies only when the `EnableNonTerminalStates` property is `true`. | C on page 2-28 STT on page 2-108 |
| `NonTerminalUnstableBehavior` | RW | Behavior if a super step for the chart or state transition table exceeds the maximum number of transitions specified in the `NonTerminalMaxCounts` property before reaching a stable state, specified as one of these values:<br>• `'Proceed'` — The chart or state transition table goes to sleep with the last active state configuration.<br>• `'Throw Error'` — The chart or state transition table generates an error.<br>This property applies only when the `EnableNonTerminalStates` property is `true`. | C on page 2-28 STT on page 2-108 |

## See Also

`sfclipboard` | `sfnew` | `sfroot`

## More About

- “Create Charts by Using the Stateflow API” on page 1-19
- “Create and Delete Stateflow Objects” on page 1-13
- “Modify Properties and Call Functions of Stateflow Objects” on page 1-10