# Stateflow®

API

# MATLAB&SIMULINK®

**R**2022**a**

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

# Contents

**API Object Reference**

**2**

**API Object Function Reference**

**3**

**1**

# Using the Stateflow API

# Overview of the Stateflow API

| **In this section...** |
| --- |
| "Hierarchy of Stateflow API Objects" on page 1-2 |
| "Access Stateflow API Objects" on page 1-4 |
| "Modify Properties of API Objects" on page 1-4 |
| "Call API Object Functions" on page 1-5 |

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.
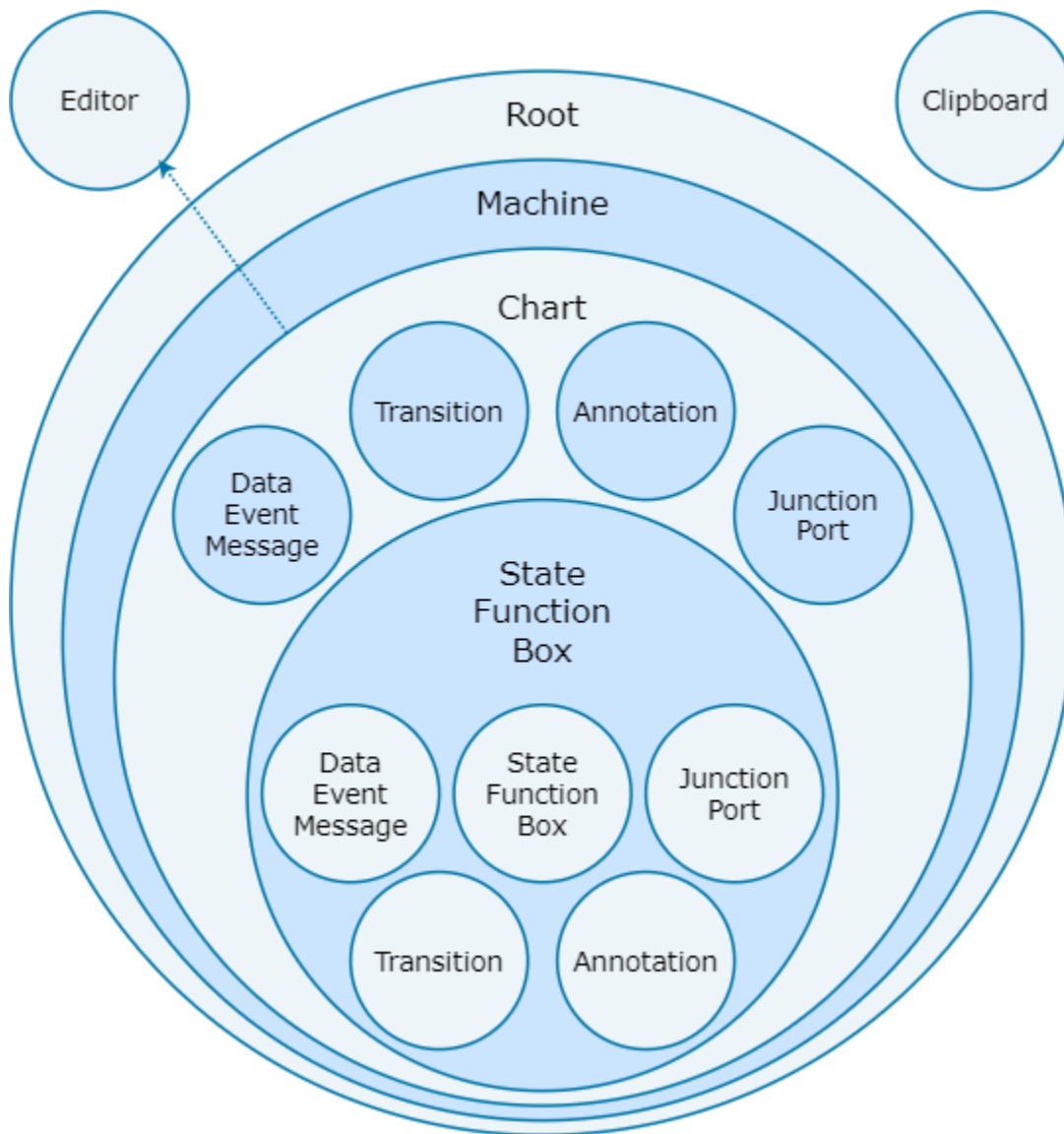
The hierarchy consists of four levels of containment:

- **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, entry and exit ports, 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, entry and exit ports, 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 2-58, "Zoom out on Stateflow Chart" on page 2-59, and "Set Zoom Factor" on page 2-59.
- **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 1-27.

## Modify Properties of API Objects

API objects have properties that correspond to the values you set in the Stateflow Editor. For example, to use the editor to change the position of a state, you 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 1-11.

## Call API Object Functions

API objects have functions that correspond to actions in the Stateflow Editor. For example, to use the editor to open the Properties dialog box for a transition, you 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 1-11.

## See Also

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

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

## More About

# Access Objects in Your Stateflow Chart

The objects in the Stateflow API 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. For more information, see "Overview of the Stateflow API" on page 1-2.

## Find Objects in a Chart

With the `find` function, you can locate an API object by specifying search criteria. 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

By default, the `find` function finds objects at all depths of containment within an object. For example, suppose that `ch` is a `Stateflow.Chart` object that corresponds to this chart. The chart contains a parent state `A` with two child states, `A1` and `A2`. For more information on this example, see "Create Charts by Using a MATLAB Script" on page 1-32.
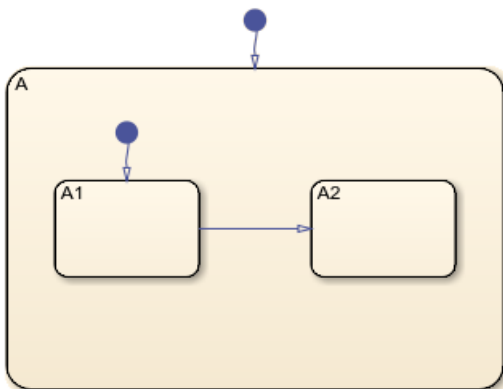


Calling the `find` function to find all the states in this chart returns an array with three `Stateflow.State` objects:

```
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, you can call the `find` function to search for states in the first level of containment in 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","-depth",1);
get(states,"Name")

ans =

  3×1 cell array

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

To exclude 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'}
```

## Navigate the Stateflow Hierarchy

After you access an API object, you can use the `getChildren` and `getParent` functions to navigate through the Stateflow hierarchy and identify the children that the object contains or the parent that contains the object.

### Find Child Objects

To find the children of an API object, call the `getChildren` function. For instance, suppose that `ch` is the `Stateflow.Chart` object that corresponds to the chart in the previous example. Calling the `getChildren` function on `ch` returns an array that contains a `Stateflow.State` object and a `Stateflow.Transition` object.

```
children = getChildren(ch);
arrayfun(@class,children,UniformOutput=false)

ans =

  2×1 cell array

    {'Stateflow.State'     }
    {'Stateflow.Transition'}
```

The first element in the array is a `Stateflow.State` object that corresponds to state A.

```
state = children(1);
state.Name

ans =

    'A'
```

The second element in the array is a `Stateflow.Transition` object that corresponds to the default transition into state A.

```
children(2).Source

ans =

    []
```

```
children(2).Destination.Name

ans =

    'A'
```

Similarly, calling the `getChildren` function on the state returns an array that contains two `Stateflow.State` objects and two `Stateflow.Transition` objects.

```
grandchildren = getChildren(state);
arrayfun(@class,grandchildren,UniformOutput=false)

ans =

  4×1 cell array

    {'Stateflow.State'     }
    {'Stateflow.State'     }
    {'Stateflow.Transition'}
    {'Stateflow.Transition'}
```

The first and second elements in this array are `Stateflow.State` objects that correspond to the states A1 and A2.

```
grandchildren(1).Name

ans =

    'A1'
```

```
grandchildren(2).Name
```

```
ans =

    'A2'
```

The third and fourth elements in `grandchildren` are `Stateflow.Transition` objects that correspond to the transitions into states A1 and between state A1 and A2, respectively.

```
grandchildren(3).Source
```

```
ans =

     []
```

```
grandchildren(3).Destination.Name
```

```
ans =

    'A1'
```

```
grandchildren(4).Source.Name
```

```
ans =

    'A1'
```

```
grandchildren(4).Destination.Name
```

```
ans =

    'A2'
```

**Find Parent Object**

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

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

```
ans =

    'A'
```

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

```
grandparent = getParent(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, suppose that you select the transition from state `A1` to state `A2` in the previous example. Calling `sfgco` returns the corresponding `Stateflow.Transition` object:

```
tr = sfgco;
str = 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 | getChildren | getParent | setdiff | sfgco | arrayfun | class

**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 1-11
- "Create and Delete Stateflow Objects" on page 1-21
- "Create Charts by Using the Stateflow API" on page 1-27

# Modify Properties and Call Functions of Stateflow Objects

| In this section... |
| --- |
| "Call Object Functions" on page 1-11 |
| "Access Properties by Using Dot Notation" on page 1-11 |
| "Get and Set the Values of Multiple Properties" on page 1-12 |

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. For example, to use the editor to change the position of a state, you 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 correspond to actions in the Stateflow Editor. For example, to use the editor to open the Properties dialog box for a transition, you 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);
```

## Call Object Functions

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)
```

## Access Properties by Using 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";
```

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 in a single command 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"});
```

You can also use the `get` to access properties of multiple API objects. For example, this command returns a cell array that contains the names and descriptions of the `Stateflow.Chart` objects in the array `chartArray`:

```
chartInfo = get(chartArray,{"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."})
```

To set the names and descriptions of the `Stateflow.Chart` objects in the array `chartArray`, enter:

```
set(chartArray,{"Name","Description"},chartInfo)
```

In this command, `chartInfo` must be an *N*-by-2 cell array, where *N* equals the number of charts in `chartArray`. The first column in `chartInfo` contains the new chart names, and the second column contains the new descriptions.

## See Also

**Functions**
defaultTransitions | dialog | fitToView

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

## More About

*   "Overview of the Stateflow API" on page 1-2
*   "Summary of Stateflow API Objects and Properties" on page 1-34

# Refactor Charts Programmatically

You can use the Stateflow API to programmatically refactor common issues that affect the legibility of Stateflow charts. You can programmatically analyze a chart to identify issues with confusingly nested states, inconsistent state names, or extraneous transitions, and then fix these issues by using the Stateflow API.

## Example Model

In this example, you use a poorly designed model taht emulates the behavior of an insect. The Stateflow chart describes this behavior:

- The insect is confined to a box.
- The insect moves in a straight line by default.
- When the insect hits a wall, the insect bounces off the wall and continues in the opposite direction.
- When the insect sees prey, the insect redirects its trajectory to head towards the prey at an increased speed.
- When the insect sees a predator, the insect redirects its trajectory to run away from the predator at an increased speed.
- The insect can see in a directional cone of finite length, which means the insect must be close to an object and facing it in order to detect its presence.
- After 12 hours, the insect is tired and has to rest.

This Stateflow chart has an inconsistent naming scheme in which some state names start with a capital letter and some do not. Additionally, the chart contains states that are nested more than three levels deep, as well as extraneous transitions.
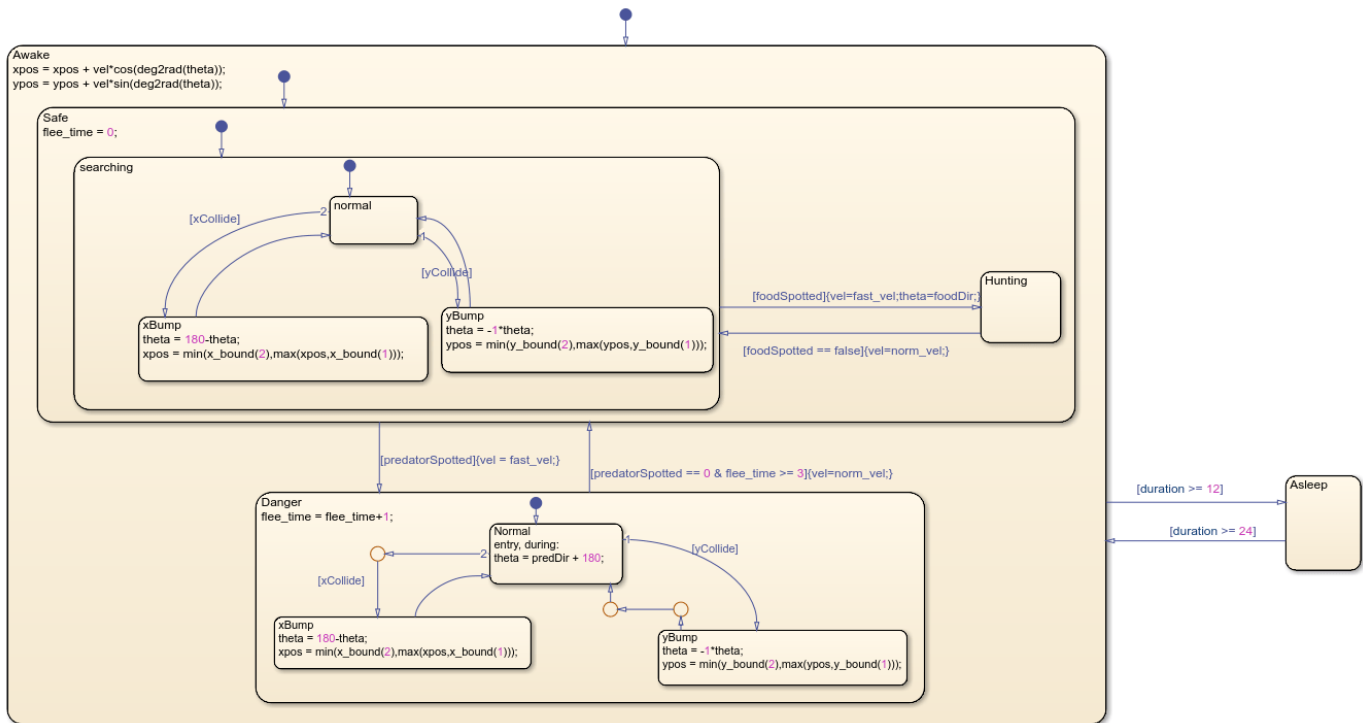
## Open the Model

This Stateflow chart has an inconsistent naming scheme in which some state names start with a capital letter and some do not. Additionally, the chart contains states that are nested more than three levels deep, as well as extraneous transitions.

To interact with the Stateflow chart using the API, open the model, then locate the chart. For more information on this workflow, see "Access Stateflow API Objects" on page 1-4.

Start by clearing the workspace and closing all previously open models. Then, open the model that contains the chart you wish to inspect.

```
openExample('stateflow/InsectInABoxExample')
```

Because the root object is the parent of all Stateflow API Objects, you can use `find` to find the chart object. For more information, see "Hierarchy of Stateflow API Objects" on page 1-2.

## Chart Depth Issues

### Flagging Chart Depth Issues

To improve chart legibility, avoid nesting charts deeper than three levels deep, where possible. Instead, use subcharts. You can search for states that are nested too deeply by using `find` and its optional arguments to create an array:

```
all_states = find(ch,"-isa","Stateflow.State");
shallow_states = find(ch,"-isa","Stateflow.State","-depth",3);
deep_states = setdiff(all_states,shallow_states);
```

You can inspect the contents of that array in MATLAB through several means. For example, you can write a script to print the contents of the array to the Command Window. This script includes information regarding the hierarchy, which can help you find specific states easily in a large or complex Stateflow chart.

```
num_deep_states = length(deep_states);
deep_state_names = "";
for i=1:num_deep_states
    deep_path_text = getHierarchy(deep_states(i));
    deep_state_names = deep_state_names + newline + " * " + deep_path_text;
end

warning("The following states have been flagged for having a depth greater than 3." + ...
    newline + "Consider creating subcharts instead." + deep_state_names)
```

The `for` loop in this script iterates through each problem state. The `getHierarchy` helper function, shown below, traces the parentage of the state to the original model, which shows you how to navigate to the flagged state. The script calls `getParent` in a loop to trace the parents of the state as far back as the model. The `warning` function prints the paths to all the problem states in the Command Window.

```
function path_text = getHierarchy(state)
    path_text = state.Path + "." + state.Name;

    indices = strfind(path_text,"/");

    for i = 3:numel(indices)
        path_text = replaceBetween(path_text,indices(i),indices(i),".");
    end
end
```

For the insect behavior example, this script flags the substates under searching. These substates are four levels deep.

```
Warning: The following states have been flagged for having a depth greater than 3.
Consider creating subcharts instead.
 * insect_behavior_example/Behavioral Logic/Awake.Safe.searching.normal
 * insect_behavior_example/Behavioral Logic/Awake.Safe.searching.yBump
 * insect_behavior_example/Behavioral Logic/Awake.Safe.searching.xBump
```

**Fixing Chart Depth Issues**

After you identify issues with the depth of the states in your chart, you can fix these issues programmatically.

To convert a `Stateflow.State` to a subchart, access the IsSubchart property and set it to true. To automate this process for a large-scale chart, the helper function `createSubchart` looks for states at a depth of three which contain other nested states, then converts the states at a depth of three to subcharts. Then, it repeats the process by calling itself recursively with the new subcharts as parents.

```
function createSubchart(parent,depth)

    states = setdiff( ...
        find(parent,"-isa","Stateflow.State","-depth",depth), ...
        find(parent,"-isa","Stateflow.State","-depth",depth-1));

    for i=1:length(states)
```

```
            children = setdiff(find(states(i),'-isa','Stateflow.State', ...
                '-depth',1),states(i));
            for j=1:length(children)
                states(i).isSubchart = true;
                createSubchart(states(i),depth);
                break
            end
        end
    end
end
```

## State Name Issues

### Flagging State Name Issues

To improve readability, use the same naming conventions throughout the chart. You can use the `find` function to identify the states that do not start with capital letters.

```
correct_states = find(ch,"-isa","Stateflow.State", ...
    "-regexp","Name","^[A-Z]\w*");
misnamed_states = setdiff(all_states,correct_states);
```

When using `find` to search, techniques you can use include:

**1** Logical expressions such as `and` to add several conditions for your search.

**2** Regular expressions to search for open-ended combinations of alphanumeric characters. In this example, `regexp` searches for states whose names start with an uppercase letter and are composed entirely of alphanumeric characters or underscores.

You can view the results in the `misnamed_states` variable or use a script to print the results to the Command Window. For this example, use this script:

```
bad_state_names = "";
for j=1:length(misnamed_states)
    misnamed_path_text = getHierarchy(misnamed_states(j));
    bad_state_names = bad_state_names + newline + " * " + misnamed_path_text;
end

warning("The following states have been flagged for an improper naming scheme." + ...
    newline + "Consider renaming these states to start with a capital letter." + ...
    bad_state_names)
```

The script outputs the states that start with lowercase letters:

```
Warning: The following states have been flagged for an improper naming scheme.
Consider renaming these states to start with a capital letter.
 * insect_behavior_example/Behavioral Logic/Awake.Danger.yBump
 * insect_behavior_example/Behavioral Logic/Awake.Safe.searching.normal
 * insect_behavior_example/Behavioral Logic/Awake.Safe.searching
 * insect_behavior_example/Behavioral Logic/Awake.Danger.xBump
 * insect_behavior_example/Behavioral Logic/Awake.Safe.searching.yBump
 * insect_behavior_example/Behavioral Logic/Awake.Safe.searching.xBump
```

### Fixing State Name Issues

After identifying the list of state names, replace the first letter of each identified state in `misnamed_states` with its uppercase equivalent by using `upper`.

```
for k=1:length(misnamed_states)
    state = misnamed_states(k);
    state.Name = [upper(state.Name(1)) state.Name(2:end)];
end
```

---

**Note** Similarly, you can refactor chart data names with the following code:

```
rt = sfroot;
rt.find('-isa', 'Stateflow.Data')
data.refactor('newName')
```

---

## Unnecessary Transitions

### Flagging Unnecessary Transitions

Unnecessary transitions may increase the complexity of your charts. For example, transitions may not be necessary if you have several sequential transitions. In this example, you can define an unnecessary transition as connected to a junction or chain of junctions with only one sourced transition and one sunk transition. This code uses functions `sourcedTransitions` and `sinkedTransitions` to identify these transitions.

```
function is_linear = checkLinearity(junction)
    is_linear = false;
    source_transitions = sourcedTransitions(junction);
    sink_transitions = sinkedTransitions(junction);
    if (length(source_transitions)<=1) && (length(sink_transitions)<=1)
        is_linear = true;
    end
end
```

You can view the results in the `is_linear` variable. Alternatively, you can use a script to print the results to the Command Window. This code uses the names of the parents to which these transitions belong. Because most transitions do not have a unique name, the code prints the name of the parent state, box, or subchart.

```
junctions = find(ch,"-isa","Stateflow.Junction");
bad_transition_parent_names = "";
flagged_junctions = [];

for k=1:length(junctions)
    if checkLinearity(junctions(k))
        flagged_junctions = [flagged_junctions junctions(k)];
        bad_transition_parent_names = bad_transition_parent_names + ...
            newline + " * " + getHierarchy(getParent(junctions(k)));
    end
end

warning("The following states contain transitions that have been flagged for" + ...
    newline + "superfluousness. You may be able to combine transitions within these states into" + ...
    newline + "one transition with several actions." + ...
    bad_transition_parent_names)
```

The script prints the names of the transitions with only one sourced transition and one sunk transition:

```
Warning: The following states contain transitions that have been flagged for
superfluousness. You may be able to combine transitions within these states into
one transition with several actions.
 * insect_behavior_example/Behavioral Logic/Awake.Danger
 * insect_behavior_example/Behavioral Logic/Awake.Danger
 * insect_behavior_example/Behavioral Logic/Awake.Danger
```

**Fixing Unnecessary Transitions**

In this example, the variable `flagged_junctions` contains the list of transitions with only one sourced transition and one sunk transition. This image shows some of the identified transitions.

This script deletes unnecessary transitions in the chart and replaces them with a single transition. The script:

1  Locates the start transition and end transition of each chain of one or more unnecessary transitions using the helper function `findEnds`, which uses the `Stateflow.Transition` API object properties to navigate through the chart programmatically. It uses Source and Destination to move in opposite directions over the chain. Its sole input is the `flagged_junctions` array.

2  Uses the outputs of `findEnds` to navigate the chain from the start transition until it reaches the end transition.

   a  Stores any guards or actions from the intermediate transitions in variables `cat_guard` and `cat_action` using the helper function `buildConcatLabelStr`. To see more about the formatting involved in constructing a label string, see "Specify Labels in States and Transitions Programmatically" on page 1-24.

   b  Deletes the intermediate transitions and junctions.

3  Builds a new, concatenated label from `cat_guard` and `cat_action` and applies it to the start transition.

4  Redirects the destination of the start transition to the terminus of the end transition. The start transition now encompasses the length of the original chain.

```
[starts,ends] = findEnds(flagged_junctions);

for n=1:length(starts)

    current = starts(n);
    cat_guard = "";
    cat_action = "";

    while current ~= ends(n)
        [current,cat_guard,cat_action] = buildConcatLabelStr(current,...
            cat_guard,cat_action);

        previous = current;
        current = sourcedTransitions(current.Destination);
        delete(previous.Destination);

        if ~ismember(previous,starts)
            delete(previous);
        end

    end
    [~,cat_guard,cat_action] = buildConcatLabelStr(current,cat_guard,...
```

```matlab
            cat_action);

        temp_string = "";

        if strlength(cat_guard)
            temp_string = "[" + cat_guard + "]";
        end
        if strlength(cat_action)
            temp_string = temp_string + "{" + cat_action + "}";
        end

        starts(n).LabelString = temp_string;
        starts(n).Destination = ends(n).Destination;
        delete(ends(n));
    end

    function [starts,ends]=findEnds(flagged_junctions)

        starts = [];
        ends = [];

        while ~isempty(flagged_junctions)
            junction = flagged_junctions(1);
            flagged_junctions = flagged_junctions(2:end);

            [startPoint,flagged_junctions_1] = findEndPoint(...
                sinkedTransitions(junction),flagged_junctions,-1);

            [endPoint,flagged_junctions_2] = findEndPoint(...
                sourcedTransitions(junction),flagged_junctions,1);

            starts = [starts startPoint];
            ends = [ends endPoint];
            flagged_junctions = intersect(flagged_junctions_1,flagged_junctions_2);
        end

        function [start_trans,flagged_junc] = findEndPoint(transition,flagged_junc,dir)
            if dir == 1
                next = transition.Destination;
                nextTrans = sourcedTransitions(next);
            else
                next = transition.Source;
                nextTrans = sinkedTransitions(next);
            end
            if isa(next,'Stateflow.Junction') && checkLinearity(next)
                flagged_junc = flagged_junc(~ismember(flagged_junc,next));
                [start_trans,flagged_junc] = findEndPoint(nextTrans,flagged_junc,dir);
            else
                start_trans = transition;
            end
        end
    end

    function [current,cat_guard,cat_action] = buildConcatLabelStr( ...
        current,cat_guard,cat_action)

guard = current.Condition;
```

```
action = current.ConditionAction;

if strlength(guard)
    if ~strlength(cat_guard)
        cat_guard = guard;
    else
        cat_guard = cat_guard + "&&" + guard;
    end
end

if strlength(action)
    if ~strlength(cat_action)
        cat_action = action;
    else
        cat_action = cat_action + ";" + action;
    end
end
end
```

After the code completes, the resulting chart contains no junctions with only one sourced transition and one sunk transition.

## See Also

**Functions**
defaultTransitions | dialog | fitToView

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

## More About

- "Overview of the Stateflow API" on page 1-2
- "Create and Delete Stateflow Objects" on page 1-21
- "Access Objects in Your Stateflow Chart" on page 1-6
- "Modify Properties and Call Functions of Stateflow Objects" on page 1-11
- "Specify Labels in States and Transitions Programmatically" on page 1-24
- "Summary of Stateflow API Objects and Properties" on page 1-34

# Create and Delete Stateflow Objects

The objects in the Stateflow API 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. For more information, see "Overview of the Stateflow API" on page 1-2.

## 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 the 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 1-27.

**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 1-27

# 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 Guidelines 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><br>`{'trigger1','action1',...,'triggerN','actionN'}`<br><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 "Define Actions in a State" and "Define Actions in a Transition".

## 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 function `newline` to create newline characters:

  ```
  str = "A" + newline + ...
      "en: action1();" + newline + ...
      "du: action2();" + newline + ...
      "en,du: action3();"
  sA.LabelString = str;
  ```
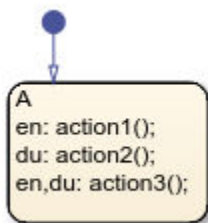


To extract the state name, `entry` action, and `during` action specified by the state label, enter:

```
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 1-27
- "Represent Operating Modes by Using States"
- "Define Actions in a Transition"

# 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.
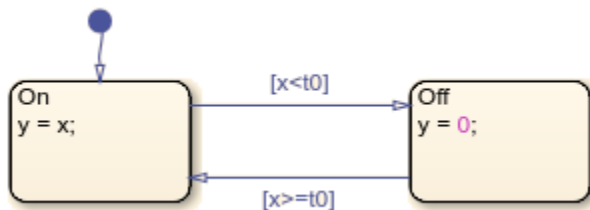


For more information on simulating this chart, see "Construct and Run a Stateflow Chart".

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 1-21.

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 1-24.

```
s1.LabelString = "On"+newline+"y = x;";
```
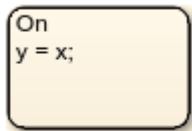


4. Create a second state. Adjust its position and specify its name and label.

```
s2 = Stateflow.State(ch);
s2.Position = [230 30 90 60];
s2.LabelString = "Off"+newline+"y = 0;";
```
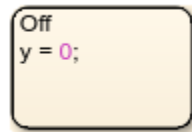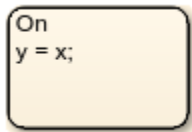


**Add Transitions**

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];
```
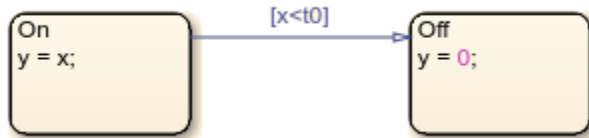


5. Create a second transition. Specify its source, destination, and label.

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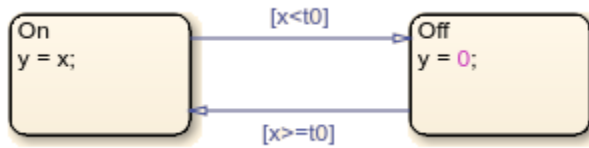


6. Add a default transition to the state `On`. To make a vertical transition, modify the values of the `SourceEndpoint` and `Midpoint` properties. For more information, see "Add a Default Transition" on page 2-180.

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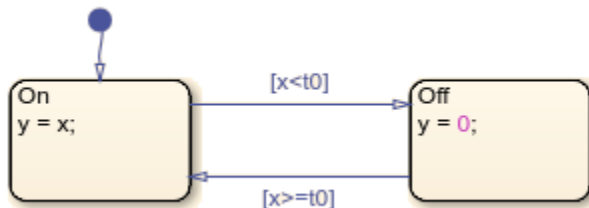


**Add Data**

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.
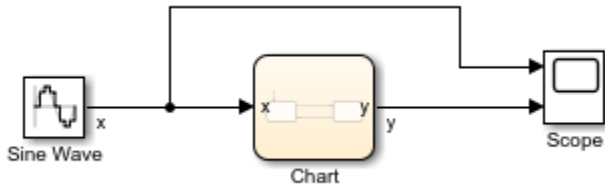


For more information, see "Simulate the Chart as a Simulink Block".

## See Also

**Blocks**
Chart

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

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

## More About

- "Overview of the Stateflow API" on page 1-2
- "Create and Delete Stateflow Objects" on page 1-21
- "Modify Properties and Call Functions of Stateflow Objects" on page 1-11
- "Specify Labels in States and Transitions Programmatically" on page 1-24

# 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)
```
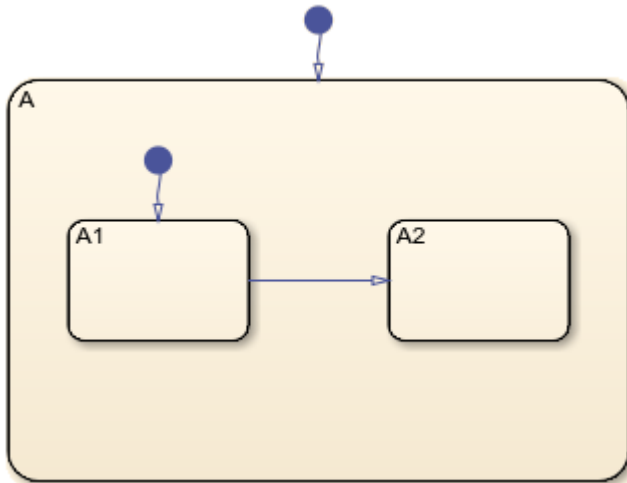


**Create Base Chart Function**

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**
view | sfroot | find | sfnew | setdiff

**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 1-27

# Summary of Stateflow API Objects and Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. For example, to use the editor to change the position of a state, you 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 1-11.

The following reference tables for Stateflow API properties have these columns:

- **Property Name** — The name of the property. To access or set a property value, use its name in dot notation along with a Stateflow object.
- **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.

## Stateflow.Annotation

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".

| Property Name | Access | Description |
|---|---|---|
| Alignment | RW | Alignment of the annotation text, specified as `"LEFT"`, `"CENTER"`, or `"RIGHT"`. |
| AutoBackgroundColor | RW | Whether to use the default background color, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><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. |

| Property Name | Access | Description |
|---|---|---|
| AutoForegroundColor | RW | Whether to use the default foreground color, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><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. |
| 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`. |
| Chart | RO | Chart that contains the annotation, specified as a `Stateflow.Chart` object. |
| ClickFcn | RW | Callback on click, specified as a string scalar or character vector. This callback contains MATLAB code to execute when to execute when you click the annotation. |
| DeleteFcn | RW | Callback at delete, specified as a string scalar or character vector. This callback contains MATLAB code to execute before you delete the annotation. |
| Description | RW | Description for the annotation, specified as a string scalar or character vector. |
| Document | RW | Document link for the annotation, specified as a string scalar or character vector. |
| DropShadow | RW | Whether to display a drop shadow around the annotation box, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| 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. |
| FixedWidth | 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 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. |

| Property Name | Access | Description |
|---|---|---|
| Font | RW | Font for the annotation text, specified as a `Stateflow.NoteFont` object with these properties:<br><br>• **Name** — Font name, specified as a character vector. This property is read-only. The `StateFont.Name` property of the chart that contains the annotation sets the value of this property.<br>• **Angle** — Font angle, specified as `"NORMAL"` or `"ITALIC"`.<br>• **Weight** — Font weight, specified as `"NORMAL"` or `"BOLD"`.<br>• **Size** — Font size, specified as a scalar. |
| 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`. |
| Id | RO | 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. |
| 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]`. |
| Interpretation | RW | Format of the annotation text, specified as `"OFF"`, `"RICH"`, or `"TEX"`. |
| IsImage | RO | Whether the annotation contains an image, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| LoadFcn | RW | Callback at model load, specified as a string scalar or character vector. This callback contains MATLAB code to execute when you load the model that contains the annotation. |
| Machine | RO | Machine that contains the annotation, specified as a `Stateflow.Machine` object. |
| Path | RO | Location of the parent of the annotation in the model hierarchy, specified as a character vector. |
| PlainText | RO | Annotation text without formatting, specified as a character vector. |
| Position | RW | Position and size of annotation box, specified as a four-element numeric vector of the form `[left top width height]`. |
| Subviewer | RO | 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. |
| Tag | RW | User-defined tag for the annotation, specified as data of any type. |
| Text | RW | Text for the annotation, specified as a string scalar or character vector. |
| 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. |

## Stateflow.AtomicBox

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".

| Property Name | Access | Description |
|---|---|---|
| `BadIntersection` | RO | Whether the atomic box graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| `Chart` | RO | Chart that contains the atomic box, specified as a `Stateflow.Chart` object. |
| `CommentText` | RW | Comment text for the atomic box, specified as a string scalar or 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 `""`. |
| `ContentPreviewEnabled` | RW | Whether to display a preview of the atomic box contents, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| `Description` | RW | Description for the atomic box, specified as a string scalar or character vector. |
| `Document` | RW | Document link for the atomic box, specified as a string scalar or character vector. |
| `FontSize` | RW | 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. |
| `Id` | RO | 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. |
| `IsExplicitlyCommented` | RW | 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 "Comment Out Objects in a Stateflow Chart". |
| `IsImplicitlyCommented` | RO | 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 state, box, or function that contains it. |
| `IsLink` | RO | Whether the atomic box is a library link, specified as a numeric or logical 1 (`true`) or 0 (`false`). |

| Property Name | Access | Description |
|---|---|---|
| LabelString | RW | Label for the atomic box, specified as a string scalar or character vector. |
| Machine | RO | Machine that contains the atomic box, specified as a `Stateflow.Machine` object. |
| Name | RW | Name of the atomic box, specified as a string scalar or character vector. |
| Path | RO | Location of the parent of the atomic box in the model hierarchy, specified as a character vector. |
| Position | RW | Position and size of the atomic box, specified as a four-element numeric vector of the form `[left top width height]`. |
| SSIdNumber | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the atomic box from other objects in the model. |
| 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. |
| Subviewer | RO | 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. |
| Tag | RW | User-defined tag for the atomic box, specified as data of any type. |

## Stateflow.AtomicSubchart

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".

| Property Name | Access | Description |
|---|---|---|
| ArrowSize | RW | Size of incoming transition arrows, specified as a scalar. |
| BadIntersection | RO | Whether the atomic subchart graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| Chart | RO | Chart that contains the atomic subchart, specified as a `Stateflow.Chart` object. |
| CommentText | RW | Comment text for the atomic subchart, specified as a string scalar or 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 `""`. |

| Property Name | Access | Description |
|---|---|---|
| ContentPreviewEnabled | RW | Whether to display a preview of the atomic subchart contents, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| Debug | RW | Debugger properties for the state, atomic subchart, or Simulink based state, specified as a `Stateflow.StateDebug` object with these properties:<br><br>• **OnEntry** — Whether to set the `On State Entry` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **OnDuring** — Whether to set the `During State` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **OnExit** — Whether to set the `On State Exit` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>For more information, see "Set Breakpoints to Debug Charts". |
| Description | RW | Description for the atomic subchart, specified as a string scalar or character vector. |
| Document | RW | Document link for the atomic subchart, specified as a string scalar or character vector. |
| ExecutionOrder | RW | 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:<br><br>• The `Type` property of the atomic subchart is `"AND"`.<br>• The `UserSpecifiedStateTransitionExecutionOrder` property of the chart that contains the atomic subchart is `true`. |
| FontSize | RW | 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. |
| HasOutputData | RW | 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". |
| Id | RO | 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. |
| IsExplicitlyCommented | RW | 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 "Comment Out Objects in a Stateflow Chart". |
| IsImplicitlyCommented | RO | 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 state or box that contains it. |
| IsLink | RO | Whether the atomic subchart is a library link, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| LabelString | RW | Label for the atomic subchart, specified as a string scalar or character vector. |

| Property Name | Access | Description |
|---|---|---|
| LoggingInfo | RW | Signal logging properties for the atomic subchart, specified as a `Stateflow.SigLoggingInfo` object with these properties:<br><br>• **DataLogging** — Whether to enable signal logging, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>• **DecimateData** — Whether to limit the amount of logged data, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>• **Decimation** — Decimation interval, specified as an integer scalar. This property applies only when the `DecimateData` property is `true`.<br><br>• **LimitDataPoints** — Whether to limit the number of data points to log, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>• **MaxPoints** — Maximum number of data points to log, specified as an integer scalar. This property applies only when the `LimitDataPoints` property is `true`.<br><br>• **NameMode** — Source of the signal name, specified as `"SignalName"` or `"Custom"`.<br><br>• **LoggingName** — Custom signal name, specified as a string scalar or character vector. This property applies only when the `NameMode` property is `"Custom"`.<br><br>Signal logging saves the self activity of the atomic subchart to the MATLAB workspace during simulation. For more information, see "Log Simulation Output for States and Data". |
| Machine | RO | Machine that contains the atomic subchart, specified as a `Stateflow.Machine` object. |
| Name | RW | Name of the atomic subchart, specified as a string scalar or character vector. |
| OutputData | RO | 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`. |
| OutputMonitoringMode | RW | Monitoring mode for the active state output data, specified as a string scalar or character vector. For atomic subcharts, the only option is `"SelfActivity"`. |
| OutputPortName | RW | Name of the active state data object for the atomic subchart, specified as a string scalar or character vector. This property applies only when the `HasOutputData` property for the atomic subchart is `true`. |
| Path | RO | Location of the parent of the atomic subchart in the model hierarchy, specified as a character vector. |
| Position | RW | Position and size of the atomic subchart, specified as a four-element numeric vector of the form `[left top width height]`. |
| SSIdNumber | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the atomic subchart from other objects in the model. |

| Property Name | Access | Description |
|---|---|---|
| Subchart | RO | 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. For more information, see "Add Exit Port and Junction to Atomic Subchart" on page 2-18. |
| Subviewer | RO | 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. |
| Tag | RW | User-defined tag for the atomic subchart, specified as data of any type. |
| TestPoint | RW | Whether to set the atomic subchart as a test point, specified as a numeric or logical 1 (`true`) or 0 (`false`). You can monitor testpoints with a floating scope during simulation. You can also log test point values to the MATLAB workspace. For more information, see "Monitor Test Points in Stateflow Charts". |
| Type | RO | Decomposition of sibling states, specified as `'AND'` or `'OR'`. The atomic subchart inherits this property from the `Decomposition` property of its parent state or chart. |

## Stateflow.Box

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".

| Property Name | Access | Description |
|---|---|---|
| BadIntersection | RO | Whether the box graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| Chart | RO | Chart that contains the box, specified as a `Stateflow.Chart` object. |
| CommentText | RW | Comment text for the box, specified as a string scalar or 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 `""`. |
| ContentPreviewEnabled | RW | 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`. |
| Description | RW | Description for the box, specified as a string scalar or character vector. |
| Document | RW | Document link for the box, specified as a string scalar or character vector. |
| FontSize | RW | 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. |
| Id | RO | 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. |

| Property Name | Access | Description |
|---|---|---|
| IsExplicitlyCommen ted | RW | 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 "Comment Out Objects in a Stateflow Chart". |
| IsGrouped | RW | 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. |
| IsImplicitlyCommen ted | RO | 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 state, box, or function that contains it. |
| IsSubchart | RW | Whether the box is a subchart, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| LabelString | RW | Label for the box, specified as a string scalar or character vector. |
| Machine | RO | Machine that contains the box, specified as a `Stateflow.Machine` object. |
| Name | RW | Name of the box, specified as a string scalar or character vector. |
| Path | RO | Location of the parent of the box in the model hierarchy, specified as a character vector. |
| Position | RW | Position and size of the box, specified as a four-element numeric vector of the form `[left top width height]`. |
| Subviewer | RO | 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. |
| Tag | RW | User-defined tag for the box, specified as data of any type. |

## Stateflow.Chart

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 "Model Finite State Machines by Using Stateflow Charts" and "Create Charts by Using the Stateflow API" on page 1-27.

| Property Name | Access | Description |
|---|---|---|
| ActionLanguage | RW | 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". |
| AllowGlobalAccessT oExportedFunctions | 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`. |

| Property Name | Access | Description |
|---|---|---|
| `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. |
| `ChartUpdate` | RW | Activation method for the chart, specified as `"CONTINUOUS"`, `"DISCRETE"`, or `"INHERITED"`. For more information, see "Update Method". |
| `Debug` | RW | Debugger properties for the chart, specified as a `Stateflow.ChartDebug` object with this property:<br><br>• **`Breakpoints.OnEntry`** — Whether to set the `On Chart Entry` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>For more information, see "Set Breakpoints to Debug Charts". |
| `Decomposition` | RW | Decomposition of substates at the top level of containment in the chart, specified as `"EXCLUSIVE_OR"` or `"PARALLEL_AND"`. For more information, see "Define Exclusive and Parallel Modes by Using State Decomposition". |
| `Description` | RW | Description for the chart, specified as a string scalar or character vector. |
| `Dirty` | RW | Whether the chart has changed after being opened or saved, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| `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`). For more information, see "Define State Activity Enumeration Type". |
| `Document` | RW | Document link for the chart, specified as a string scalar or character vector. |
| `Editor` | RO | 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. |
| `EmlDefaultFimath` | RW | Default `fimath` properties for the chart, specified as one of these values:<br><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><br>This property applies only when the `ActionLanguage` property of the chart is `"MATLAB"`. |
| `EnableBitOps` | RW | 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". |
| `EnableNonTerminalStates` | RW | 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". |

| Property Name | Access | Description |
|---|---|---|
| EnableZeroCrossings | RW | 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". |
| EnumTypeName | RW | Name of the enumerated data type for the active state data object for the chart, specified as a string scalar or character vector. For more information, see "Enum Name". |
| ExecuteAtInitialization | RW | 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". |
| 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". |
| 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". |
| HasOutputData | RW | 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". |
| Iced | RO | 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. |
| Id | RO | 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. |
| InitializeOutput | RW | 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". |
| InputFimath | RW | Default `fimath` object, specified as a string scalar or character vector. When the `EmlDefaultFimath` property for the chart is `"Other:UserSpecified"`, you can use this property to:<br><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><br>This property applies only to charts that use MATLAB as the action language. |
| 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. |

| Property Name | Access | Description |
| --- | --- | --- |
| `Locked` | RW | 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. |
| `Machine` | RO | Machine that contains the chart, specified as a `Stateflow.Machine` object. |
| `Name` | RW | Name of the chart, specified as a string scalar or character vector. |
| `NonTerminalMaxCounts` | RW | 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` | RW | 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: <br><br> • `"Proceed"` — The chart goes to sleep with the last active state configuration. <br> • `"Throw Error"` — The chart generates an error. <br><br> This property applies only when the `EnableNonTerminalStates` property for the chart is `true`. |
| `OutputData` | RO | 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`. |
| `OutputMonitoringMode` | RW | Monitoring mode for the active state output data, specified as `"ChildActivity"` or `"LeafStateActivity"`. |
| `OutputPortName` | RW | Name of the active state data object for the chart, specified as a string scalar or character vector. This property applies only when the `HasOutputData` property for the chart is `true`. |
| `Path` | RO | Location of the chart in the model hierarchy, specified as a character vector. |
| `SampleTime` | RW | Sample time for activating the chart, specified as a string scalar or character vector. This property applies only when the `ChartUpdate` property for the chart is `"DISCRETE"`. |
| `SaturateOnIntegerOverflow` | RW | 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". |
| `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. |

| Property Name | Access | Description |
|---|---|---|
| `StateFont` | RW | Font for the box, function, and state labels in the chart, specified as a `Stateflow.StateFont` object with these properties:<br><br>• **Name** — Font name, specified as a string scalar or character vector. This property also determines the font for annotations in the chart.<br>• **Angle** — Font angle, specified as `"NORMAL"` or `"ITALIC"`.<br>• **Weight** — Font weight, specified as `"NORMAL"` or `"BOLD"`.<br>• **Size** — Default font size for new boxes, functions, and states, specified as a scalar. This property also determines the default font size for new annotations in the chart. |
| `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. |
| `StateMachineType` | RW | State machine semantics implemented by the chart, specified as `"Classic"`, `"Mealy"`, or `"Moore"`. For more information, see "Overview of Mealy and Moore Machines". |
| `StatesWhenEnabling` | RW | Behavior of the states when a function-call input event reenables the chart, specified as one of these values:<br><br>• `""` — The chart does not contain function-call input events.<br>• `"held"` — The chart maintains the most recent values of the states.<br>• `"reset"` — The chart reverts to the initial conditions of the states.<br><br>For more information, see "Control States in Charts Enabled by Function-Call Input Events". |
| `SupportVariableSizing` | RW | Whether the chart supports variable-size data, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see "Declare Variable-Size Data in Stateflow Charts". |
| `Tag` | RW | User-defined tag for the chart, specified as data of any type. |
| `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. |
| `TransitionFont` | RW | Font for the transition labels in the chart, specified as a `Stateflow.TransFont` object with these properties:<br><br>• **Name** — Font name, specified as a string scalar or character vector.<br>• **Angle** — Font angle, specified as `"NORMAL"` or `"ITALIC"`.<br>• **Weight** — Font weight, specified as `"NORMAL"` or `"BOLD"`.<br>• **Size** — Default font size for new transitions, specified as a scalar. |
| `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. |

| Property Name | Access | Description |
|---|---|---|
| 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 treats all fixed-point inputs as fi objects.<br>• "Fixed-point & Integer" — The chart treats all fixed-point and integer inputs as fi objects.<br><br>This property applies only to charts that use MATLAB as the action language. |
| UserSpecifiedState TransitionExecutio nOrder | 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". |
| Visible | RW | Whether the Stateflow Editor window is displaying the chart, specified as a numeric or logical 1 (true) or 0 (false). |

## Stateflow.Clipboard

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.

Stateflow.Clipboard objects do not have any properties.

## Stateflow.Data

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".

| Property Name | Access | Description |
|---|---|---|
| CompiledSize | RO | Data size as determined by the compiler, specified as a character vector. |
| CompiledType | RO | Data type as determined by the compiler, specified as a character vector. |

| Property Name | Access | Description |
|---|---|---|
| DataType | RW | Type of the data object, specified as a string scalar or character vector that depends on the `Props.Type.Method` property of the data object:<br><br>• 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.<br>• If the `Props.Type.Method` property of the data object is `"Built-in"`, you can specify this property with one of these options:<br><br>   • `"double"`<br>   • `"single"`<br>   • `"int8"`<br>   • `"int16"`<br>   • `"int32"`<br>   • `"int64"`<br>   • `"uint8"`<br>   • `"uint16"`<br>   • `"uint32"`<br>   • `"uint64"`<br>   • `"boolean"`<br>   • `"string"`<br>   • `"ml"` (Supported only in charts that use C as the action language)<br>• Otherwise, the `Props.Type` properties of the data object determine the value of this property.<br><br>For more information, see the section Add Data on page 1-0     in "Create Charts by Using the Stateflow API" on page 1-27. |
| Debug | RW | Debugger properties for the data object, specified as a `Stateflow.DataDebug` object with this property:<br><br>• **Watch** — 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". |
| Description | RW | Description for the data object, specified as a string scalar or character vector. |
| Document | RW | Document link for the data object, specified as a string scalar or character vector. |
| Id | RO | 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. |

| Property Name | Access | Description |
|---|---|---|
| `InitializeMethod` | RW | Method for initializing the value of the data object, specified as a string scalar or character vector that depends on the scope of the data:<br><br>• For local and output data, use `"Expression"` or `"Parameter"`.<br>• For constant data, use `"Expression"`.<br>• For input data, parameters, and data store memory, use `"Not Needed"`.<br><br>To specify the initial value of the data object, use the `Props.InitialValue` property.<br><br>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". |
| `LoggingInfo` | RW | Signal logging properties for the data object, specified as a `Stateflow.SigLoggingInfo` object with these properties:<br><br>• **DataLogging** — Whether to enable signal logging, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **DecimateData** — Whether to limit the amount of logged data, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **Decimation** — Decimation interval, specified as an integer scalar. This property applies only when the `DecimateData` property is `true`.<br>• **LimitDataPoints** — Whether to limit the number of data points to log, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **MaxPoints** — Maximum number of data points to log, specified as an integer scalar. This property applies only when the `LimitDataPoints` property is `true`.<br>• **NameMode** — Source of the signal name, specified as `"SignalName"` or `"Custom"`.<br>• **LoggingName** — Custom signal name, specified as a string scalar or character vector. This property applies only when the `NameMode` property is `"Custom"`.<br><br>Signal logging saves the values of the data object to the MATLAB workspace during simulation. For more information, see "Log Simulation Output for States and Data". |
| `Machine` | RO | Machine that contains the data object, specified as a `Stateflow.Machine` object. |
| `Name` | RW | Name of the data object, specified as a string scalar or character vector. |
| `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". |

| Property Name | Access | Description |
|---|---|---|
| Path | RO | Location of the parent of the data object in the model hierarchy, specified as a character vector. |
| Port | RW | 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". |

| Property Name | Access | Description |
|---|---|---|
| Props | RW | Data specification properties, specified as a `Stateflow.DataProps` object with these properties:<br><br>• **Type.Method** — Method for setting the type of the data object, specified as a string scalar or character vector.<br><br>    • For local, input, output, or parameter data, use `"Inherited"`, `"Built-in"`, `"Bus Object"`, `"Enumerated"`, `"Expression"`, or `"Fixed point"`.<br><br>    • For constant data, use `"Built-in"`, `"Expression"`, or `"Fixed point"`.<br><br>    • For data store memory data, use `"Inherited"`.<br><br>    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".<br><br>• **Type.BusObject** — Name of the `Simulink.Bus` object that defines the data object, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Bus Object"`. For more information, see "Access Bus Signals Through Stateflow Structures".<br><br>• **Type.EnumType** — Name of the enumerated type that defines the data object, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Enumerated"`. For more information, see "Reference Values by Name by Using Enumerated Data".<br><br>• **Type.Expression** — Expression that evaluates to the data type of the data object, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Expression"`. For more information, see "Specify Data Properties by Using MATLAB Expressions".<br><br>• **Type.Signed** — Signedness, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `Type.Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts".<br><br>• **Type.WordLength** — Word length, in bits, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts".<br><br>• **Type.Fixpt.ScalingMode** — Method for scaling the fixed-point data object, specified as `"Binary point"`, `"Slope and bias"`, or `"None"`. This property applies only when the `Type.Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts".<br><br>• **Type.Fixpt.FractionLength** — Fraction length, in bits, specified as a string scalar or character vector. This property applies only when the `Type.Method` property is `"Fixed point"` and the `Type.Fixpt.ScalingMode` property is `"Binary point"`. |

| Property Name | Access | Description |
|---|---|---|
| | | • **Type.Fixpt.Slope** — Slope, specified as a string scalar or character vector. This property applies only when the `Type.Method` property is `"Fixed point"` and the `Type.Fixpt.ScalingMode` property is `"Slope and bias"`. |
| | | • **Type.Fixpt.Bias** — Bias, specified as a string scalar or character vector. This property applies only when the `Type.Method` property is `"Fixed point"` and the `Type.Fixpt.ScalingMode` property is `"Slope and bias"`. |
| | | • **Type.Fixpt.Lock** — Whether to prevent replacement of the fixed-point type with an autoscaled type chosen by the Fixed-Point Tool (Fixed-Point Designer), specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `Type.Method` property of the data object is `"Fixed point"`. |
| | | • **Array.Size** — Size of the data object, specified as a string scalar or character vector. For more information, see "Specify Size of Stateflow Data". |
| | | • **Array.IsDynamic** — Whether the data object has variable size, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property is equivalent to the **Variable Size** check box in the **Property Inspector**, the Model Explorer, or the Data properties dialog box. For more information, see "Declare Variable-Size Data in Stateflow Charts". |
| | | • **Array.FirstIndex** — Index for the first element of the array data object, specified as a string scalar or 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". |
| | | • **Complexity** — Whether the data object accepts complex values, specified as `"On"` or `"Off"`. For more information, see "Complex Data in Stateflow Charts". |
| | | • **InitialValue** — Initial value, specified as a string scalar or character vector. For more information, see "Initial value". |
| | | • **Range.Minimum** — Minimum value, specified as a string scalar or character vector. For more information, see "Limit range". |
| | | • **Range.Maximum** — Maximum value, specified as a string scalar or character vector. For more information, see "Limit range". |
| | | • **ResolveToSignalObject** — 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". |
| | | • **Unit.Name** — Unit of measurement, specified as a string scalar or character vector. This property applies only to input and output data. For more information, see "Specify Units for Stateflow Data". |
| SSIdNumber | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the data object from other objects in the model. |

| Property Name | Access | Description |
|---|---|---|
| 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". |
| Scope | RW | Scope of the data object, specified as one of these values:<br><br>• `"Local"`<br>• `"Input"`<br>• `"Output"`<br>• `"Constant"`<br>• `"Parameter"`<br>• `"Data Store Memory"`<br>• `"Temporary"`<br>• `"Imported"`<br>• `"Exported"`<br><br>For more information, see "Scope". |
| Tag | RW | User-defined tag for the data object, specified as data of any type. |
| TestPoint | RW | Whether to set the data object as a test point, specified as a numeric or logical 1 (`true`) or 0 (`false`). You can monitor testpoints with a floating scope during simulation. You can also log test point values to the MATLAB workspace. For more information, see "Monitor Test Points in Stateflow Charts". |
| 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. |
| 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". |

## Stateflow.EMChart

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 is 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. See "Create and Configure MATLAB S-Functions" (Simulink).

For more information, see "Implement MATLAB Functions in Simulink with MATLAB Function Blocks" (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 Blocks Programmatically" (Simulink).

---

| Property Name | Access | Description |
|---|---|---|
| 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). |
| ChartUpdate | RW | Activation method for the MATLAB Function block, specified as `"CONTINUOUS"`, `"DISCRETE"`, or `"INHERITED"`. For more information, see "Update method" (Simulink). |
| Description | RW | Description for the MATLAB Function block, specified as a string scalar or character vector. |
| Dirty | RW | Whether the MATLAB Function block has changed after being opened or saved, specified as a numeric or logical `1` (`true`) or `0` (`false`). |
| Document | RW | Document link for the MATLAB Function block, specified as a string scalar or character vector. |
| EmlDefaultFimath | RW | Default `fimath` properties for the MATLAB Function block, specified as one of these values:<br><br>• `"Same as MATLAB Default"` — Use the same `fimath` properties as the current default `fimath` object.<br><br>• `"Other:UserSpecified"` — Use the `InputFimath` property to specify the default `fimath` object. |
| Iced | RO | 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. |
| Id | RO | 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. |
| InputFimath | RW | Default `fimath` object, specified as a string scalar or character vector. When the `EmlDefaultFimath` property for the MATLAB Function block is `"Other:UserSpecified"`, you can use this property to:<br><br>• Enter an expression that constructs a `fimath` object.<br><br>• Enter the variable name for a `fimath` object in the MATLAB or model workspace. |

| Property Name | Access | Description |
|---|---|---|
| `Inputs` | RO | 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. |
| `Locked` | RW | 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. |
| `Machine` | RO | Machine that contains the MATLAB Function block, specified as a `Stateflow.Machine` object. |
| `Name` | RW | Name of the MATLAB Function block, specified as a string scalar or character vector. |
| `Outputs` | RO | 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. |
| `Path` | RO | Location of the MATLAB Function block in the model hierarchy, specified as a character vector. |
| `SampleTime` | RW | Sample time for activating the MATLAB Function block, specified as a string scalar or character vector. This property applies only when the `ChartUpdate` property for the MATLAB function is `"DISCRETE"`. |
| `SaturateOnIntegerOverflow` | RW | 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). |
| `Script` | RW | Code for the MATLAB Function block, specified as a string scalar or character vector. To enter multiple lines of code, you can:<br><br>• Call the MATLAB function `sprintf` and use `\n` to insert newline characters:<br><br>`str = sprintf("function y=f(x)\ny=x+1;\nend");`<br>`block.Script = str;`<br><br>• Enter a concatenated text expression that uses the function `newline` to create newline characters:<br><br>`str = "function y=f(x)" + newline + ...`<br>`    "y=x+1;" + newline + ...`<br>`    "end";`<br>`block.Script = str;` |
| `SupportVariableSizing` | RW | Whether the MATLAB Function block supports variable-size data, specified as a numeric or logical `1` (`true`) or `0` (`false`). For more information, see "Declare Variable-Size Outputs" (Simulink). |
| `Tag` | RW | User-defined tag for the MATLAB Function block, specified as data of any type. |

| Property Name | Access | Description |
|---|---|---|
| TreatAsFi | RW | Inherited Simulink signals to treat as Fixed-Point Designer `fi` objects, specified as one of these values:<br><br>• `"Fixed-point"` — The MATLAB Function block treats all fixed-point inputs as `fi` objects.<br>• `"Fixed-point & Integer"` — The MATLAB Function block treats all fixed-point and integer inputs as `fi` objects. |
| VectorOutputs1D | RW | Whether the MATLAB Function block outputs column vectors as one-dimensional data, specified as a numeric or logical `0` (`false`) or `1` (`true`). For more information, see "Interpret output column vectors as one-dimensional data" (Simulink). |

## Stateflow.EMFunction

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".

| Property Name | Access | Description |
|---|---|---|
| BadIntersection | RO | Whether the MATLAB function graphically intersects a box, state, or function, specified as a numeric or logical `1` (`true`) or `0` (`false`). |
| Chart | RO | Chart that contains the MATLAB function, specified as a `Stateflow.Chart` object. |
| CommentText | RW | Comment text for the MATLAB function, specified as a string scalar or 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 `""`. |
| Description | RW | Description for the MATLAB function, specified as a string scalar or character vector. |
| Document | RW | Document link for the MATLAB function, specified as a string scalar or character vector. |

| Property Name | Access | Description |
|---|---|---|
| EmlDefaultFimath | RW | Default `fimath` properties for the MATLAB function, specified as one of these values:<br><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><br>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. |
| FontSize | RW | 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. |
| Id | RO | 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. |
| InlineOption | RW | Appearance of the MATLAB function in generated code, specified as one of these values:<br><br>• `"Auto"` — An internal calculation determines the appearance of the function in generated code.<br>• `"Function"` — The function is implemented as a separate C function.<br>• `"Inline"` — Calls to the function are replaced by code as long as the function is not part of a recursion.<br><br>For more information, see "Inline State Functions in Generated Code" (Simulink Coder). |
| InputFimath | RW | Default `fimath` object, specified as a string scalar or character vector. When the `EmlDefaultFimath` property for the MATLAB function is `"Other:UserSpecified"`, you can use this property to:<br><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><br>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. |
| IsExplicitlyCommented | RW | 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 "Comment Out Objects in a Stateflow Chart". |
| IsImplicitlyCommented | RO | 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 state, box, or function that contains it. |

| Property Name | Access | Description |
|---|---|---|
| `LabelString` | RW | Label for the MATLAB function, specified as a string scalar or character vector. |
| `Machine` | RO | Machine that contains the MATLAB function, specified as a `Stateflow.Machine` object. |
| `Name` | RW | Name of the MATLAB function, specified as a string scalar or character vector. |
| `Path` | RO | Location of the parent of the MATLAB function in the model hierarchy, specified as a character vector. |
| `Position` | RW | Position and size of the MATLAB function, specified as a four-element numeric vector of the form `[left top width height]`. |
| `SSIdNumber` | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the MATLAB function from other objects in the model. |
| `SaturateOnIntegerOverflow` | RW | 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".<br><br>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. |
| `Script` | RW | Code for the MATLAB function, specified as a string scalar or 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>`function.Script = str;`<br><br>• Enter a concatenated text expression that uses the function `newline` to create newline characters:<br><br>`str = "function y=f(x)" + newline + ...`<br>`    "y=x+1;" + newline + ...`<br>`    "end";`<br>`function.Script = str;` |
| `Subviewer` | RO | 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. |
| `Tag` | RW | User-defined tag for the MATLAB function, specified as data of any type. |

## Stateflow.Editor

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.

| Property Name | Access | Description |
|---|---|---|
| `WindowPosition` | RW | Position and size of the Stateflow editor window, specified as a four-element numeric vector of the form `[left top width height]`. |
| `ZoomFactor` | RW | 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%. |

## Stateflow.Event

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".

| Property Name | Access | Description |
|---|---|---|
| `Debug` | RW | Debugger properties for the event, specified as a `Stateflow.EventDebug` object with these properties:<br><br>• **`Breakpoints.StartBroadcast`** — Whether to set the `Start of Broadcast` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **`Breakpoints.EndBroadcast`** — Whether to set the `End of Broadcast` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>For more information, see "Set Breakpoints to Debug Charts". |
| `Description` | RW | Description for the event, specified as a string scalar or character vector. |
| `Document` | RW | Document link for the event, specified as a string scalar or character vector. |
| `Id` | RO | 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. |
| `Machine` | RO | Machine that contains the event, specified as a `Stateflow.Machine` object. |
| `Name` | RW | Name of the event, specified as a string scalar or character vector. |
| `Path` | RO | Location of the parent of the event in the model hierarchy, specified as a character vector. |
| `Port` | RW | Port index for the event, specified as an integer scalar. This property applies only to input and output events. For more information, see "Port". |
| `Scope` | RW | Scope of the event, specified as `"Local"`, `"Input"`, or `"Output"`. For more information, see "Scope". |
| `Tag` | RW | User-defined tag for the event, specified as data of any type. |

| Property Name | Access | Description |
|---|---|---|
| Trigger | RW | Type of trigger associated with the event, specified as a string scalar or character vector that depends on the scope of the data:<br><br>• For input events, use `"Function call"`, `"Rising"`, `"Falling"`, or `"Either"`.<br>• For output events, use `"Function call"` or `"Either"`.<br><br>This property does not apply to local events. For more information, see "Trigger". |

## Stateflow.Function

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".

| Property Name | Access | Description |
|---|---|---|
| BadIntersection | RO | Whether the graphical function graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| Chart | RO | Chart that contains the graphical function, specified as a `Stateflow.Chart` object. |
| CommentText | RW | Comment text for the graphical function, specified as a string scalar or 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 `""`. |
| ContentPreviewEnabled | RW | 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`. |
| Debug | RW | Debugger properties for the graphical function, specified as a `Stateflow.FunctionDebug` object with this property:<br><br>• **Breakpoints.OnDuring** — Whether to set the `During Function Call` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>For more information, see "Set Breakpoints to Debug Charts". |
| Description | RW | Description for the graphical function, specified as a string scalar or character vector. |
| Document | RW | Document link for the graphical function, specified as a string scalar or character vector. |
| FontSize | RW | 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. |

| Property Name | Access | Description |
|---|---|---|
| Id | RO | 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. |
| InlineOption | RW | Appearance of the graphical function in generated code, specified as one of these values:<br><br>• "Auto" — An internal calculation determines the appearance of the function in generated code.<br>• "Function" — The function is implemented as a separate C function.<br>• "Inline" — Calls to the function are replaced by code as long as the function is not part of a recursion.<br><br>For more information, see "Inline State Functions in Generated Code" (Simulink Coder). |
| IsExplicitlyCommented | RW | 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 "Comment Out Objects in a Stateflow Chart". |
| IsGrouped | RW | 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. |
| IsImplicitlyCommented | RO | 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 state, box, or function that contains it. |
| IsSubchart | RW | Whether the function is a subchart, specified as a numeric or logical 1 (true) or 0 (false). |
| LabelString | RW | Label for the graphical function, specified as a string scalar or character vector. |
| Machine | RO | Machine that contains the graphical function, specified as a Stateflow.Machine object. |
| Name | RW | Name of the graphical function, specified as a string scalar or character vector. |
| Path | RO | Location of the parent of the graphical function in the model hierarchy, specified as a character vector. |
| Position | RW | Position and size of the graphical function, specified as a four-element numeric vector of the form [left top width height]. |
| SSIdNumber | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the graphical function from other objects in the model. |
| Subviewer | RO | 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. |

| Property Name | Access | Description |
|---|---|---|
| Tag | RW | User-defined tag for the graphical function, specified as data of any type. |

## Stateflow.Junction

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 "Resume Prior Substate Activity by Using History Junctions".

| Property Name | Access | Description |
|---|---|---|
| ArrowSize | RW | Size of incoming transition arrows, specified as a scalar. |
| Chart | RO | Chart that contains the junction, specified as a `Stateflow.Chart` object. |
| CommentText | RW | Comment text added to the junction, specified as a string scalar or 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 `""`. |
| Description | RW | Description for the junction, specified as a string scalar or character vector. |
| Document | RW | Document link for the junction, specified as a string scalar or character vector. |
| Id | RO | 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. |
| IsExplicitlyCommented | RW | 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 "Comment Out Objects in a Stateflow Chart". |
| IsImplicitlyCommented | RO | 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 state, box, or function that contains it. |
| Machine | RO | Machine that contains the junction, specified as a `Stateflow.Machine` object. |
| Path | RO | Location of the parent of the junction in the model hierarchy, specified as a character vector. |

| Property Name | Access | Description |
|---|---|---|
| Position | RW | Position and size of the junction, specified as a `Stateflow.JunctionPosition` object with these properties:<br><br>• **Center** — 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.<br>• **Radius** — Radius of the junction, specified as a scalar. |
| SSIdNumber | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the junction from other objects in the model. |
| Subviewer | RO | 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. |
| Tag | RW | User-defined tag for the junction, specified as data of any type. |
| Type | RW | Type of junction, specified as one of these values:<br><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 |

## Stateflow.Machine

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.

| Property Name | Access | Description |
|---|---|---|
| Created | RO | Date of the creation of the machine, specified as a character vector. |
| Creator | RW | Creator of the machine, specified as a string scalar or character vector. |

| Property Name | Access | Description |
|---|---|---|
| Debug | RW | Debugger properties for charts in the machine, specified as a `Stateflow.MachineDebug` object with these properties:<br><br>• **Animation.Enabled** — 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.<br>• **Animation.Delay** — 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>• **Animation.MaintainHighlighting** — 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`). |
| Description | RW | Description for the machine, specified as a string scalar or character vector. |
| Dirty | RW | 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`). |
| Document | RW | Document link for the machine, specified as a string scalar or character vector. |
| FullFileName | RO | Full file path of the Simulink model for the machine, specified as a character vector. |
| Iced | RO | 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. |
| Id | RO | 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. |
| 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`). |
| Locked | RW | 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. |
| Modified | RW | Record of modifications to the machine, specified as a string scalar or character vector. |
| Name | RO | Name of the Simulink model for the machine, specified as a character vector. |

| Property Name | Access | Description |
|---|---|---|
| Path | RO | Location of the machine in the model hierarchy, specified as a character vector. |
| Tag | RW | User-defined tag for the machine, specified as data of any type. |
| Version | RW | Version of the machine, specified as a string scalar or character vector. |

## Stateflow.Message

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".

| Property Name | Access | Description |
|---|---|---|
| CompiledSize | RO | Message data size as determined by the compiler, specified as a character vector. |
| CompiledType | RO | Data type as determined by the compiler, specified as a character vector. |
| DataType | RW | Data type of the message, specified as a string scalar or character vector that depends on the `Props.Type.Method` property of the message:<br><br>• If the `Props.Type.Method` property of the message is `"Inherit"`, the value of this property is `"Inherit: Same as Simulink"`.<br>• If the `Props.Type.Method` property of the message is `"Built-in"`, you can specify this property with one of these options:<br><br>  • `"double"`<br>  • `"single"`<br>  • `"int8"`<br>  • `"int16"`<br>  • `"int32"`<br>  • `"int64"`<br>  • `"uint8"`<br>  • `"uint16"`<br>  • `"uint32"`<br>  • `"uint64"`<br>  • `"boolean"`<br>  • `"string"`<br>  • `"ml"` (Supported only in charts that use C as the action language)<br>• Otherwise, the `Props.Type` properties of the message determine the value of this property.<br><br>For more information, see the section Add Data on page 1-0     in "Create Charts by Using the Stateflow API" on page 1-27. |
| Description | RW | Description for the message, specified as a string scalar or character vector. |

| Property Name | Access | Description |
|---|---|---|
| `Document` | RW | Document link for the message, specified as a string scalar or character vector. |
| `Id` | RO | 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. |
| `InitializeMethod` | RW | Method for initializing the value of the message data, specified as a string scalar or character vector that depends on the scope of the message:<br><br>• For local and output messages, use `"Expression"` or `"Parameter"`.<br>• For input messages, use `"Not Needed"`.<br><br>To specify the initial value of the message data, use the `Props.InitialValue` property.<br><br>For more information, see "Initial Value". |
| `Machine` | RO | Machine that contains the message, specified as a `Stateflow.Machine` object. |
| `MessagePriorityOrder` | RW | Type of priority queue for the message, specified as one of these values:<br><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><br>This property applies only when the `QueueType` property of the message is `"Priority"`. For more information, see "Queue Type". |
| `Name` | RW | Name of the message, specified as a string scalar or character vector. |
| `Path` | RO | Location of the parent of the message in the model hierarchy, specified as a character vector. |
| `Port` | RW | Port index for the message, specified as an integer scalar. This property applies only to input and output messages. For more information, see "Port". |
| `Priority` | RW | Priority for the message, specified as a string scalar or 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). |

| Property Name | Access | Description |
|---|---|---|
| Props | RW | Data specification properties, specified as a `Stateflow.DataProps` object with these properties: <br><br> • **Type.Method** — 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 Model Explorer and the Data properties dialog box. For more information, see "Specify Type of Stateflow Data". <br><br> • **Type.BusObject** — Name of the `Simulink.Bus` object that defines the message data, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Bus Object"`. For more information, see "Access Bus Signals Through Stateflow Structures". <br><br> • **Type.EnumType** — Name of the enumerated type that defines the message data, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Enumerated"`. For more information, see "Reference Values by Name by Using Enumerated Data". <br><br> • **Type.Expression** — Expression that evaluates to the data type of the message data, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Expression"`. For more information, see "Specify Data Properties by Using MATLAB Expressions". <br><br> • **Type.Signed** — Signedness, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `Type.Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts". <br><br> • **Type.WordLength** — Word length, in bits, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts". <br><br> • **Type.Fixpt.ScalingMode** — Method for scaling the fixed-point message data, specified as `"Binary point"`, `"Slope and bias"`, or `"None"`. This property applies only when the `Type.Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts". <br><br> • **Type.Fixpt.FractionLength** — Fraction length, in bits, specified as a string scalar or character vector. This property applies only when the `Type.Method` property is `"Fixed point"` and the `Type.Fixpt.ScalingMode` property is `"Binary point"`. <br><br> • **Type.Fixpt.Slope** — Slope, specified as a string scalar or character vector. This property applies only when the `Type.Method` property is `"Fixed point"` and the `Type.Fixpt.ScalingMode` property is `"Slope and bias"`. <br><br> • **Type.Fixpt.Bias** — Bias, specified as a string scalar or character vector. This property applies only to when the `Type.Method` property |

| Property Name | Access | Description |
|---|---|---|
| | | is "Fixed point" and the `Type.Fixpt.ScalingMode` property is "Slope and bias".<br><br>• **`Type.Fixpt.Lock`** — Whether to prevent replacement of the fixed-point type with an autoscaled type chosen by the Fixed-Point Tool (Fixed-Point Designer), specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `Type.Method` property of the data object is "Fixed point".<br><br>• **`Array.Size`** — Size of the message data, specified as a string scalar or character vector. For more information, see "Specify Size of Stateflow Data".<br><br>• **`Complexity`** — Whether the message accepts complex values, specified as "On" or "Off". For more information, see "Complex Data in Stateflow Charts".<br><br>• **`InitialValue`** — Initial value, specified as a string scalar or character vector. |
| 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". |
| 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". |
| QueueType | RW | Order in which messages are removed from the receiving queue, specified as one of these values:<br><br>• "FIFO" — First in, first out.<br><br>• "LIFO" — Last in, first out.<br><br>• "Priority" — Remove messages according to the value in the data field. To specify the order, use the `MessagePriorityOrder` property for the message.<br><br>This property applies only to local messages and to input messages that have `UseInternalQueue` set to `true`. For more information, see "Queue Type". |
| SSIdNumber | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the message from other objects in the model. |
| Scope | RW | Scope of the message, specified as "Local", "Input", or "Output". For more information, see "Scope". |
| Tag | RW | User-defined tag for the message, specified as data of any type. |
| 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". |

## Stateflow.Port

Use `Stateflow.Port` objects to create ports and junctions that provide entry and exit connections across boundaries in the Stateflow hierarchy. Entry and exit ports improve componentization by isolating the transition logic for entering and exiting states. Unlike supertransitions, they can be used in atomic subcharts. For more information, see "Create Entry and Exit Connections Across State Boundaries".

Entry and exit ports are located on the boundary of a state or atomic subchart. Each port has a matching junction that marks the entry or exit point inside the state or atomic subchart. The port and junction are represented by separate `Stateflow.Port` objects.

| Property Name | Access | Description |
| --- | --- | --- |
| `ArrowSize` | RW | Size of incoming transition arrows, specified as a scalar. |
| `Chart` | RO | Chart that contains the port or junction, specified as a `Stateflow.Chart` object. |
| `CommentText` | RW | Comment text added to the entry or exit junction, specified as a string scalar or character vector. This property applies only to entry and exit junctions 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 `""`. |
| `Description` | RW | Description for the port or junction, specified as a string scalar or character vector. |
| `Document` | RW | Document link for the port or junction, specified as a string scalar or character vector. |
| `Home` | RO | Home state or subchart, specified as a `Stateflow.State` or `Stateflow.AtomicSubchart` object. The home of an entry or exit port is the state or subchart whose boundary contains the port. This property applies only to entry and exit ports. |
| `Id` | RO | 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. |
| `IsExplicitlyCommented` | RW | Whether to comment out the junction and port pair, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the entry or exit junction and selecting **Comment Out**. This property applies only to entry and exit junctions. Attempting to set this property on an entry or exit port results in an error. For more information, see "Comment Out Objects in a Stateflow Chart". |
| `IsImplicitlyCommented` | RO | Whether the port or junction is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The port or junction is implicitly commented out when you comment out a state, box, or function that contains it. Additionally, entry and exit ports are implicitly commented out when you comment out their matching entry or exit junction. |
| `LabelPosition` | RW | Position and size of the port or junction label, specified as a four-element numeric vector of the form `[left top width height]`. |

| Property Name | Access | Description |
|---|---|---|
| LabelString | RW | Label for the port or junction, specified as a string scalar or character vector. Changing this property automatically sets the LabelString property for the matching Stateflow.Port object to the same value. |
| Linked | RO | Whether the port or junction has a matching junction or port, specified as a numeric or logical 1 (true) or 0 (false). This property is used to detect internal inconsistencies in the chart. |
| Machine | RO | Machine that contains the port or junction, specified as a Stateflow.Machine object. |
| Path | RO | Location of the parent of the port or junction in the model hierarchy, specified as a character vector. |
| PortType | RO | Type of port or junction, specified as one of these values:<br><br>• 'EntryJunction' — Entry junction inside a state or atomic subchart<br>• 'EntryPort' — Entry port on the boundary of a state or atomic subchart<br>• 'ExitJunction' — Exit junction inside a state or atomic subchart<br>• 'ExitPort' — Exit port on the boundary of a state or atomic subchart |
| Position | RW | Position and size of the port or junction, specified as a Stateflow.PortPosition object with these properties:<br><br>• **Center** — Position of the center of the port or junction, specified as a two-element numeric vector [x  y] of coordinates relative to the upper left corner of the chart.<br>• **Radius** — Radius of the port or junction, specified as a scalar. |
| SSIdNumber | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the port or junction from other objects in the model. |
| Subviewer | RO | Subviewer for the port or junction, specified as a Stateflow.Chart, Stateflow.State, or Stateflow.Box object. The subviewer is the chart or subchart where you can graphically view the port. |
| Tag | RW | User-defined tag for the port or junction, specified as data of any type. |

## Stateflow.SLFunction

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 Functions in Stateflow Charts".

| Property Name | Access | Description |
|---|---|---|
| `BadIntersection` | RO | Whether the Simulink function graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| `Chart` | RO | Chart that contains the Simulink function, specified as a `Stateflow.Chart` object. |
| `CommentText` | RW | Comment text added to the Simulink function, specified as a string scalar or 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 `""`. |
| `ContentPreviewEnabled` | RW | Whether to display a preview of the Simulink function contents, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| `Description` | RW | Description for the Simulink function, specified as a string scalar or character vector. |
| `Document` | RW | Document link for the Simulink function, specified as a string scalar or character vector. |
| `FontSize` | RW | Font size for the Simulink function label, specified as a scalar. The `StateFont.Size` property of the chart that contains the Simulink function sets the initial value of this property. |
| `Id` | RO | 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. |
| `IsExplicitlyCommented` | RW | 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 "Comment Out Objects in a Stateflow Chart". |
| `IsImplicitlyCommented` | RO | 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 state, box, or function that contains it. |
| `LabelString` | RW | Label for the Simulink function, specified as a string scalar or character vector. |
| `Machine` | RO | Machine that contains the Simulink function, specified as a `Stateflow.Machine` object. |
| `Name` | RW | Name of the Simulink function, specified as a string scalar or character vector. |
| `Path` | RO | Location of the parent of the Simulink function in the model hierarchy, specified as a character vector. |
| `Position` | RW | Position and size of the Simulink function, specified as a four-element numeric vector of the form `[left top width height]`. |
| `SSIdNumber` | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the Simulink function from other objects in the model. |

| Property Name | Access | Description |
|---|---|---|
| Subviewer | RO | 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. |
| Tag | RW | User-defined tag for the Simulink function, specified as data of any type. |

## Stateflow.SimulinkBasedState

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".

| Property Name | Access | Description |
|---|---|---|
| ArrowSize | RW | Size of incoming transition arrows, specified as a scalar. |
| BadIntersection | RO | Whether the Simulink based state graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| Chart | RO | Chart that contains the Simulink based state, specified as a `Stateflow.Chart` object. |
| CommentText | RW | Comment text added to the Simulink based state, specified as a string scalar or character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge ![](image) 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 `""`. |
| ContentPreviewEnabled | RW | Whether to display a preview of the Simulink based state contents, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| Debug | RW | Debugger properties for the Simulink based state, specified as a `Stateflow.StateDebug` object with these properties:<br><br>• **OnEntry** — Whether to set the `On State Entry` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>• **OnDuring** — Whether to set the `During State` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>• **OnExit** — Whether to set the `On State Exit` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>For more information, see "Set Breakpoints to Debug Charts". |
| Description | RW | Description for the Simulink based state, specified as a string scalar or character vector. |
| Document | RW | Document link for the Simulink based state, specified as a string scalar or character vector. |

| Property Name | Access | Description |
|---|---|---|
| `ExecutionOrder` | RW | 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:<br><br>• The `Type` property of the Simulink based state is `"AND"`.<br>• The `UserSpecifiedStateTransitionExecutionOrder` property of the chart that contains the Simulink based state is `true`. |
| `FontSize` | RW | 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. |
| `HasOutputData` | RW | 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". |
| `Id` | RO | 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. |
| `IsExplicitlyCommented` | RW | 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 "Comment Out Objects in a Stateflow Chart". |
| `IsImplicitlyCommented` | RO | 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 state or box that contains it. |

| Property Name | Access | Description |
|---|---|---|
| LoggingInfo | RW | Signal logging properties for the Simulink based state, specified as a `Stateflow.SigLoggingInfo` object with these properties:<br><br>• **DataLogging** — Whether to enable signal logging, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **DecimateData** — Whether to limit the amount of logged data, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **Decimation** — Decimation interval, specified as an integer scalar. This property applies only when the `DecimateData` property is `true`.<br>• **LimitDataPoints** — Whether to limit the number of data points to log, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **MaxPoints** — Maximum number of data points to log, specified as an integer scalar. This property applies only when the `LimitDataPoints` property is `true`.<br>• **NameMode** — Source of the signal name, specified as `"SignalName"` or `"Custom"`.<br>• **LoggingName** — Custom signal name, specified as a string scalar or character vector. This property applies only when the `NameMode` property is `"Custom"`.<br><br>Signal logging saves the self activity of the Simulink based state to the MATLAB workspace during simulation. For more information, see "Log Simulation Output for States and Data". |
| Machine | RO | Machine that contains the Simulink based state, specified as a `Stateflow.Machine` object. |
| Name | RW | Name of the Simulink based state, specified as a string scalar or character vector. |
| OutputData | RO | 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`. |
| OutputMonitoringMode | RW | Monitoring mode for the active state output data, specified as a string scalar or character vector. For Simulink based states, the only option is `"SelfActivity"`. |
| OutputPortName | RW | Name of the active state data object for the Simulink based state, specified as a string scalar or character vector. This property applies only when the `HasOutputData` property for the Simulink based state is `true`. |
| Path | RO | Location of the parent of the Simulink based state in the model hierarchy, specified as a character vector. |
| Position | RW | Position and size of the Simulink based state, specified as a four-element numeric vector of the form `[left top width height]`. |
| SSIdNumber | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the Simulink based state from other objects in the model. |

| Property Name | Access | Description |
|---|---|---|
| Subviewer | RO | 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. |
| Tag | RW | User-defined tag for the Simulink based state, specified as data of any type. |
| TestPoint | RW | Whether to set the Simulink based state as a test point, specified as a numeric or logical 1 (`true`) or 0 (`false`). You can monitor testpoints with a floating scope during simulation. You can also log test point values to the MATLAB workspace. For more information, see "Monitor Test Points in Stateflow Charts". |
| Type | RO | Decomposition of sibling states, specified as `'AND'` or `'OR'`. The Simulink based state inherits this property from the `Decomposition` property of its parent state or chart. |

## Stateflow.State

Use `Stateflow.State` objects to describe an operating mode of a reactive system. For more information, see "Represent Operating Modes by Using States".

| Property Name | Access | Description |
|---|---|---|
| ArrowSize | RW | Size of incoming transition arrows, specified as a scalar. |
| BadIntersection | RO | Whether the state graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| Chart | RO | Chart that contains the state, specified as a `Stateflow.Chart` object. |
| CommentText | RW | Comment text added to the state, specified as a string scalar or 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 `""`. |
| ContentPreviewEnabled | RW | 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`. |
| Debug | RW | Debugger properties for the state, specified as a `Stateflow.StateDebug` object with these properties:<br><br>• **OnEntry** — Whether to set the `On State Entry` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **OnDuring** — Whether to set the `During State` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **OnExit** — Whether to set the `On State Exit` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>For more information, see "Set Breakpoints to Debug Charts". |

| Property Name | Access | Description |
|---|---|---|
| `Decomposition` | RW | 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". |
| `Description` | RW | Description for the state, specified as a string scalar or character vector. |
| `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 for the state is `"ChildActivity"` or `"LeafStateActivity"`. For more information, see "Define State Activity Enumeration Type". |
| `Document` | RW | Document link for the state, specified as a string scalar or character vector. |
| `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 1-24. This property is not supported in Moore charts. |
| `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 1-24. This property is not supported in Moore charts. |
| `EnumTypeName` | RW | Name of the enumerated data type for the active state data object for the state, specified as a string scalar or character vector. This property applies only when the `OutputMonitoringMode` property for the state is `"ChildActivity"` or `"LeafStateActivity"`. For more information, see "Enum Name". |
| `ExecutionOrder` | RW | 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:<br><br>• The `Type` property of the state is `"AND"`.<br>• The `UserSpecifiedStateTransitionExecutionOrder` property of the chart that contains the state is `true`. |
| `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 1-24. This property is not supported in Moore charts. |
| `FontSize` | RW | 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. |
| `HasOutputData` | RW | 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". |

| Property Name | Access | Description |
|---|---|---|
| `Id` | RO | 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. |
| `InlineOption` | RW | Appearance of the state functions in generated code, specified as one of these values:<br><br>• `"Auto"` — An internal calculation determines the appearance of state functions in generated code.<br>• `"Function"` — State functions are implemented as separate static functions.<br>• `"Inline"` — Calls to state functions are replaced by code as long as the function is not part of a recursion.<br><br>For more information, see "Inline State Functions in Generated Code" (Simulink Coder). |
| `IsExplicitlyCommented` | RW | 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 "Comment Out Objects in a Stateflow Chart". |
| `IsGrouped` | RW | 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. |
| `IsImplicitlyCommented` | RO | 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 state or box that contains it. |
| `IsSubchart` | RW | Whether the state is a subchart, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| `LabelString` | RW | Label for the state, specified as a string scalar or character vector. For more information, see "Specify Labels in States and Transitions Programmatically" on page 1-24. |

| Property Name | Access | Description |
|---|---|---|
| LoggingInfo | RW | Signal logging properties for the state, specified as a `Stateflow.SigLoggingInfo` object with these properties:<br><br>• **DataLogging** — Whether to enable signal logging, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **DecimateData** — Whether to limit the amount of logged data, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **Decimation** — Decimation interval, specified as an integer scalar. This property applies only when the `DecimateData` property is `true`.<br>• **LimitDataPoints** — Whether to limit the number of data points to log, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **MaxPoints** — Maximum number of data points to log, specified as an integer scalar. This property applies only when the `LimitDataPoints` property is `true`.<br>• **NameMode** — Source of the signal name, specified as `"SignalName"` or `"Custom"`.<br>• **LoggingName** — Custom signal name, specified as a string scalar or character vector. This property applies only when the `NameMode` property is `"Custom"`.<br><br>Signal logging saves the self activity of the state to the MATLAB workspace during simulation. For more information, see "Log Simulation Output for States and Data". |
| Machine | RO | Machine that contains the state, specified as a `Stateflow.Machine` object. |
| 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 1-24. This property is supported only in Moore charts. For more information, see "Design Guidelines for Moore Charts". |
| Name | RW | Name of the state, specified as a string scalar or character vector. |
| 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 1-24. This property is not supported in Moore charts. |
| OutputData | RO | 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`. |
| OutputMonitoringMode | RW | Monitoring mode for the active state output data, specified as `"SelfActivity"`, `"ChildActivity"`, or `"LeafStateActivity"`. |

| Property Name | Access | Description |
|---|---|---|
| `OutputPortName` | RW | Name of the active state data object for the state, specified as a string scalar or character vector. This property applies only when the `HasOutputData` property for the state is `true`. |
| `Path` | RO | Location of the parent of the state in the model hierarchy, specified as a character vector. |
| `Position` | RW | Position and size of the state, specified as a four-element numeric vector of the form `[left top width height]`. |
| `SSIdNumber` | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the state from other objects in the model. |
| `Subviewer` | RO | 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. |
| `Tag` | RW | User-defined tag for the state, specified as data of any type. |
| `TestPoint` | RW | Whether to set the state as a test point, specified as a numeric or logical 1 (`true`) or 0 (`false`). You can monitor testpoints with a floating scope during simulation. You can also log test point values to the MATLAB workspace. For more information, see "Monitor Test Points in Stateflow Charts". |
| `Type` | RO | Decomposition of sibling states, specified as `'AND'` or `'OR'`. The state inherits this property from the `Decomposition` property of its parent state or chart. |

## Stateflow.StateTransitionTableChart

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".

| Property Name | Access | Description |
|---|---|---|
| `ActionLanguage` | RW | 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". |
| `ChartColor` | RW | 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. |
| `ChartUpdate` | RW | Activation method for the state transition table, specified as `"CONTINUOUS"`, `"DISCRETE"`, or `"INHERITED"`. For more information, see "Update Method". |

| Property Name | Access | Description |
|---|---|---|
| Debug | RW | Debugger properties for the state transition table, specified as a `Stateflow.ChartDebug` object with this property:<br><br>• **Breakpoints.OnEntry** — Whether to set the `On Chart Entry` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>For more information, see "Set Breakpoints to Debug Charts". |
| Description | RW | Description for the state transition table, specified as a string scalar or character vector. |
| Dirty | RW | Whether the state transition table has changed after being opened or saved, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| 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`). For more information, see "Define State Activity Enumeration Type". |
| Document | RW | Document link for the state transition table, specified as a string scalar or character vector. |
| Editor | RO | 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. |
| EmlDefaultFimath | RW | Default `fimath` properties for the state transition table, specified as one of these values:<br><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><br>This property applies only when the `ActionLanguage` property of the state transition table is `"MATLAB"`. |
| EnableBitOps | RW | 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". |
| EnableNonTerminalStates | RW | 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". |
| EnableZeroCrossings | RW | 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". |
| EnumTypeName | RW | Name of the enumerated data type for the active state data object for the state transition table, specified as a string scalar or character vector. For more information, see "Enum Name". |

| Property Name | Access | Description |
|---|---|---|
| `ExecuteAtInitialization` | RW | 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". |
| `HasOutputData` | RW | 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". |
| `Iced` | RO | 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. |
| `Id` | RO | 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. |
| `InitializeOutput` | RW | 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". |
| `InputFimath` | RW | Default `fimath` object, specified as a string scalar or character vector. When the `EmlDefaultFimath` property for the state transition table is `"Other:UserSpecified"`, you can use this property to:<br><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><br>This property applies only when the `ActionLanguage` property of the state transition table is `"MATLAB"`. |
| `JunctionColor` | RW | 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. |
| `Locked` | RW | 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. |
| `Machine` | RO | Machine that contains the state transition table, specified as a `Stateflow.Machine` object. |
| `Name` | RW | Name of the state transition table, specified as a string scalar or character vector. |
| `NonTerminalMaxCounts` | RW | 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`. |

| Property Name | Access | Description |
|---|---|---|
| NonTerminalUnstabl eBehavior | RW | 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:<br><br>• `"Proceed"` — The state transition table goes to sleep with the last active state configuration.<br>• `"Throw Error"` — The state transition table generates an error.<br><br>This property applies only when the `EnableNonTerminalStates` property for the state transition table is `true`. |
| OutputData | RO | 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`. |
| OutputMonitoringMo de | RW | Monitoring mode for the active state output data, specified as `"ChildActivity"` or `"LeafStateActivity"`. |
| OutputPortName | RW | Name of the active state data object for the state transition table, specified as a string scalar or character vector. This property applies only when the `HasOutputData` property for the state transition table is `true`. |
| Path | RO | Location of the state transition table in the model hierarchy, specified as a character vector. |
| SampleTime | RW | Sample time for activating the state transition table, specified as a string scalar or character vector. This property applies only when the `ChartUpdate` property for the state transition table is `"DISCRETE"`. |
| SaturateOnIntegerO verflow | RW | 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". |
| StateColor | RW | Color for the 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. |
| StateFont | RW | Font for the state labels in the chart that is automatically generated for the state transition table, specified as a `Stateflow.STTStateFont` object with these properties:<br><br>• **Name** — Font name, specified as a string scalar or character vector.<br>• **Angle** — Font angle, specified as `"NORMAL"` or `"ITALIC"`.<br>• **Weight** — Font weight, specified as `"NORMAL"` or `"BOLD"`.<br>• **Size** — Default font size for new states, specified as a scalar. |
| StateLabelColor | RW | Color for the 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. |

| Property Name | Access | Description |
|---|---|---|
| StateMachineType | RW | 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". |
| StatesWhenEnabling | RW | Behavior of the states when a function-call input event reenables the state transition table, specified as one of these values:<br><br>• "" — The state transition table does not contain function-call input events.<br>• "held" — The state transition table maintains the most recent values of the states.<br>• "reset" — The state transition table reverts to the initial conditions of the states.<br><br>For more information, see "Control States in Charts Enabled by Function-Call Input Events". |
| SupportVariableSizing | RW | Whether the state transition table supports variable-size data, specified as a numeric or logical 1 (true) or 0 (false). For more information, see "Declare Variable-Size Data in Stateflow Charts". |
| Tag | RW | User-defined tag for the state transition table, specified as data of any type. |
| TransitionColor | RW | 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. |
| TransitionFont | RW | Font for the transition labels in the chart that is automatically generated for the state transition table, specified as a Stateflow.STTTransFont object with these properties:<br><br>• **Name** — Font name, specified as a string scalar or character vector.<br>• **Angle** — Font angle, specified as "NORMAL" or "ITALIC".<br>• **Weight** — Font weight, specified as "NORMAL" or "BOLD".<br>• **Size** — Default font size for new transitions, specified as a scalar. |
| TransitionLabelColor | RW | 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. |
| TreatAsFi | RW | Inherited Simulink signals to treat as Fixed-Point Designer fi objects, specified as one of these values:<br><br>• "Fixed-point" — The state transition table treats all fixed-point inputs as fi objects.<br>• "Fixed-point & Integer" — The state transition table treats all fixed-point and integer inputs as fi objects.<br><br>This property applies only when the ActionLanguage property of the state transition table is "MATLAB". |

| Property Name | Access | Description |
|---|---|---|
| `Visible` | RW | Whether the Stateflow Editor window is displaying the state transition table, specified as a numeric or logical 1 (`true`) or 0 (`false`). |

## Stateflow.Transition

Use `Stateflow.Transition` objects to create transitions from one operating mode to another. For more information, see "Transition Between Operating Modes".

| Property Name | Access | Description |
|---|---|---|
| `ArrowSize` | RW | 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. |
| `Chart` | RO | Chart that contains the transition, specified as a `Stateflow.Chart` object. |
| `CommentText` | RW | Comment text added to the transition, specified as a string scalar or 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 `""`. |
| `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 1-24. |
| `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 1-24. |
| `Debug` | RW | Debugger properties for the transition, specified as a `Stateflow.TransDebug` object with these properties:<br><br>• **`Breakpoints.WhenTested`** — Whether to set the `When Transition is Tested` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br>• **`Breakpoints.WhenValid`** — Whether to set the `When Transition is Valid` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).<br><br>For more information, see "Set Breakpoints to Debug Charts". |
| `Description` | RW | Description for the transition, specified as a string scalar or character vector. |

| Property Name | Access | Description |
|---|---|---|
| `Destination` | RW | Destination of the transition, specified as an empty array or a Stateflow API object of one of these types:<br><br>• `Stateflow.AtomicSubchart`<br>• `Stateflow.Junction`<br>• `Stateflow.SimulinkBasedState`<br>• `Stateflow.State` |
| `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. |
| `DestinationOClock` | RW | Location of the transition endpoint at its destination, specified as a scalar between 0 and 12 that describes a clock position. |
| `Document` | RW | Document link for the transition, specified as a string scalar or character vector. |
| `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". |
| `FontSize` | RW | 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. |
| `Id` | RO | 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. |
| `IsExplicitlyCommented` | RW | 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 "Comment Out Objects in a Stateflow Chart". |
| `IsImplicitlyCommented` | RO | 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 its source, its destination, or a state, box, or function that contains it. |
| `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". |
| `LabelPosition` | RW | Position and size of the transition label, specified as a four-element numeric vector of the form `[left top width height]`. |
| `LabelString` | RW | Label for the transition, specified as a string scalar or character vector. For more information, see "Specify Labels in States and Transitions Programmatically" on page 1-24. |
| `Machine` | RO | Machine that contains the transition, specified as a `Stateflow.Machine` object. |

| Property Name | Access | Description |
|---|---|---|
| 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. |
| Path | RO | Location of the parent of the transition in the model hierarchy, specified as a character vector. |
| SSIdNumber | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the transition from other objects in the model. |
| Source | RW | Source of the transition, specified as an empty array or a Stateflow API object of one of these types:<br><br>• Stateflow.AtomicSubchart<br>• Stateflow.Junction<br>• Stateflow.SimulinkBasedState<br>• Stateflow.State |
| 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. |
| SourceOClock | RW | Location of the transition endpoint at its source, specified as a scalar between 0 and 12 that describes a clock position. |
| Subviewer | RO | 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. |
| Tag | RW | User-defined tag for the transition, specified as data of any type. |
| 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 1-24. |
| 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 1-24. |

## Stateflow.TruthTable

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".

| Property Name | Access | Description |
|---|---|---|
| `ActionTable` | RW | Action table for the truth table, specified as a cell array of character vectors. |
| `BadIntersection` | RO | Whether the truth table graphically intersects a box, state, or function, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| `Chart` | RO | Chart that contains the truth table, specified as a `Stateflow.Chart` object. |
| `CommentText` | RW | Comment text added to the truth table, specified as a string scalar or 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 `""`. |
| `ConditionTable` | RW | Condition table for the truth table, specified as a cell array of character vectors. |
| `Debug` | RW | Debugger properties for the truth table, specified as a `Stateflow.FunctionDebug` object with this property: <br><br> • **Breakpoints.OnDuring** — Whether to set the `During Function Call` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`). <br><br> 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"`. For more information, see "Set Breakpoints to Debug Charts". |
| `Description` | RW | Description for the truth table, specified as a string scalar or character vector. |
| `Document` | RW | Document link for the truth table, specified as a string scalar or character vector. |
| `EmlDefaultFimath` | RW | Default `fimath` properties for the truth table, specified as one of these values: <br><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><br> 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"`. |
| `FontSize` | RW | 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. |
| `Id` | RO | 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. |

| Property Name | Access | Description |
|---|---|---|
| `InlineOption` | RW | Appearance of the truth table in generated code, specified as one of these values:<br><br>• `"Auto"` — An internal calculation determines the appearance of the truth table in generated code.<br>• `"Function"` — The truth table is implemented as a separate C function.<br>• `"Inline"` — Calls to the truth table are replaced by code as long as the truth table is not part of a recursion.<br><br>For more information, see "Inline State Functions in Generated Code" (Simulink Coder). |
| `InputFimath` | RW | Default `fimath` object, specified as a string scalar or character vector. When the `EmlDefaultFimath` property for the truth table is `"Other:UserSpecified"`, you can use this property to:<br><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><br>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"`. |
| `IsExplicitlyCommented` | RW | 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 "Comment Out Objects in a Stateflow Chart". |
| `IsImplicitlyCommented` | RO | 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 state, box, or function that contains it. |
| `LabelString` | RW | Label for the truth table, specified as a string scalar or character vector. |
| `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". |
| `Machine` | RO | Machine that contains the truth table, specified as a `Stateflow.Machine` object. |
| `Name` | RW | Name of the truth table, specified as a string scalar or character vector. |
| `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". |
| `Path` | RO | Location of the parent of the truth table in the model hierarchy, specified as a character vector. |
| `Position` | RW | Position and size of the truth table, specified as a four-element numeric vector of the form `[left top width height]`. |

| Property Name | Access | Description |
|---|---|---|
| `SSIdNumber` | RO | Session-independent identifier, specified as an integer scalar. Use this property to distinguish the truth table from other objects in the model. |
| `SaturateOnIntegerOverflow` | RW | 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". |
| `Subviewer` | RO | 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. |
| `Tag` | RW | User-defined tag for the truth table, specified as data of any type. |
| `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". |

## Stateflow.TruthTableChart

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".

| Property Name | Access | Description |
|---|---|---|
| `ActionTable` | RW | Action table for the truth table, specified as a cell array of character vectors. |
| `ChartUpdate` | RW | Activation method for the truth table, specified as `"CONTINUOUS"`, `"DISCRETE"`, or `"INHERITED"`. For more information, see "Update Method". |
| `ConditionTable` | RW | Condition table for the truth table, specified as a cell array of character vectors. |
| `Description` | RW | Description for the truth table, specified as a string scalar or character vector. |
| `Dirty` | RW | Whether the truth table has changed after being opened or saved, specified as a numeric or logical 1 (`true`) or 0 (`false`). |
| `Document` | RW | Document link for the truth table, specified as a string scalar or character vector. |

| Property Name | Access | Description |
|---|---|---|
| EmlDefaultFimath | RW | Default `fimath` properties for the truth table, specified as one of these values:<br><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. |
| Iced | RO | 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. |
| Id | RO | 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. |
| InputFimath | RW | Default `fimath` object, specified as a string scalar or character vector. When the `EmlDefaultFimath` property for the truth table is `"Other:UserSpecified"`, you can use this property to:<br><br>• Enter an expression that constructs a `fimath` object.<br>• Enter the variable name for a `fimath` object in the MATLAB or model workspace. |
| Locked | RW | 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. |
| Machine | RO | Machine that contains the truth table, specified as a `Stateflow.Machine` object. |
| Name | RW | Name of the truth table, specified as a string scalar or character vector. |
| 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". |
| Path | RO | Location of the truth table in the model hierarchy, specified as a character vector. |
| SampleTime | RW | Sample time for activating the truth table, specified as a string scalar or character vector. This property applies only when the `ChartUpdate` property for the truth table is `"DISCRETE"`. |
| SaturateOnIntegerOverflow | RW | 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". |
| SupportVariableSizing | RW | Whether the truth table supports variable-size data, specified as a numeric or logical 1 (`true`) or 0 (`false`). For more information, see "Declare Variable-Size Data in Stateflow Charts". |
| Tag | RW | User-defined tag for the truth table, specified as data of any type. |

| Property Name | Access | Description |
|---|---|---|
| `TreatAsFi` | RW | Inherited Simulink signals to treat as Fixed-Point Designer `fi` objects, specified as one of these values:<br><br>• `"Fixed-point"` — The truth table treats all fixed-point inputs as `fi` objects.<br>• `"Fixed-point & Integer"` — The truth table treats all fixed-point and integer inputs as `fi` objects. |
| `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". |

## See Also

`sfclipboard` | `sfnew` | `sfroot`

## More About

- "Create Charts by Using the Stateflow API" on page 1-27
- "Create and Delete Stateflow Objects" on page 1-21
- "Modify Properties and Call Functions of Stateflow Objects" on page 1-11

# 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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

**Text — Text for annotation**
"?" (default) | string scalar | character vector

Text for the annotation, specified as a string scalar or 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 — Font for annotation text**
`Stateflow.NoteFont` object

Font for the annotation text, specified as a `Stateflow.NoteFont` object with these properties:

- **Name** — Font name, specified as a character vector. This property is read-only. The `StateFont.Name` property of the chart that contains the annotation sets the value of this property.
- **Angle** — Font angle, specified as `"NORMAL"` or `"ITALIC"`.
- **Weight** — Font weight, specified as `"NORMAL"` or `"BOLD"`.

- **Size** — Font size, specified as a scalar.

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

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

Example: `annotation.Font.Size = 8;`

**Callbacks**

**ClickFcn — Callback on click**
`""` (default) | string scalar | character vector

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

**LoadFcn — Callback at model load**
`""` (default) | string scalar | character vector

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

**DeleteFcn — Callback at delete**
`""` (default) | string scalar | character vector

Callback at delete, specified as a string scalar or 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) | string scalar | character vector

Description for the annotation, specified as a string scalar or character vector.

**`Document` — Document link**
`""` (default) | string scalar | character vector

Document link for the annotation, specified as a string scalar or 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

getParent    Identify parent of object
dialog       Open properties dialog box
view         Display object in editing environment
fitToView    Zoom in on graphical object
setImage     Insert image into annotation

## 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"
"Summary of Stateflow API Objects and Properties" on page 1-34

**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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

**Name — Name of atomic box**
"" (default) | string scalar | character vector

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

**LabelString — Label for atomic box**
"?" (default) | string scalar | character vector

Label for the atomic box, specified as a string scalar or 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 "Comment Out Objects in a Stateflow 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 state, box, or function that contains it.

**CommentText — Comment text**
"" (default) | string scalar | character vector

Comment text for the atomic box, specified as a string scalar or 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) | string scalar | character vector

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

`Document` — **Document link**
`""` (default) | string scalar | character vector

Document link for the atomic box, specified as a string scalar or 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

| | |
|---|---|
| getParent | Identify parent of object |
| dialog | Open properties dialog box |
| isCommented | Determine if graphical object is commented out |
| view | Display object in editing environment |
| highlight | Highlight graphical object |
| fitToView | Zoom in on graphical object |

## 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"
"Summary of Stateflow API Objects and Properties" on page 1-34

**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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

**Name — Name of atomic subchart**
"" (default) | string scalar | character vector

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

**LabelString — Label for atomic subchart**
"?" (default) | string scalar | character vector

Label for the atomic subchart, specified as a string scalar or 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 "Comment Out Objects in a Stateflow 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 state or box that contains it.

**CommentText — Comment text**
"" (default) | string scalar | character vector

Comment text for the atomic subchart, specified as a string scalar or 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 `'AND'` or `'OR'`. 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**
string scalar | character vector

Name of the active state data object for the atomic subchart, specified as a string scalar or 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 string scalar or character vector. For atomic subcharts, the only option is `"SelfActivity"`.

**Signal Logging and Test Point Monitoring**

**LoggingInfo — Signal logging properties**
`Stateflow.SigLoggingInfo` object

Signal logging properties for the atomic subchart, specified as a `Stateflow.SigLoggingInfo` object with these properties:

- **DataLogging** — Whether to enable signal logging, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **DecimateData** — Whether to limit the amount of logged data, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **Decimation** — Decimation interval, specified as an integer scalar. This property applies only when the `DecimateData` property is `true`.
- **LimitDataPoints** — Whether to limit the number of data points to log, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **MaxPoints** — Maximum number of data points to log, specified as an integer scalar. This property applies only when the `LimitDataPoints` property is `true`.
- **NameMode** — Source of the signal name, specified as `"SignalName"` or `"Custom"`.
- **LoggingName** — Custom signal name, specified as a string scalar or character vector. This property applies only when the `NameMode` property is `"Custom"`.

Signal logging saves the self activity of the atomic subchart to the MATLAB workspace during simulation. For more information, see "Log Simulation Output for States and Data".

Example: `state.LoggingInfo.DataLogging = 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`). You can monitor testpoints with a floating scope during simulation. You can also log test point values to the MATLAB workspace. For more information, see "Monitor Test Points in Stateflow Charts".

**Debugging**

**Debug — Debugger properties**
`Stateflow.StateDebug` object

Debugger properties for the state, atomic subchart, or Simulink based state, specified as a `Stateflow.StateDebug` object with these properties:

- **OnEntry** — Whether to set the `On State Entry` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **OnDuring** — Whether to set the `During State` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **OnExit** — Whether to set the `On State Exit` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).

For more information, see "Set Breakpoints to Debug Charts".

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

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

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

**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. For more information, see "Add Exit Port and Junction to Atomic Subchart" on page 2-18.

**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) | string scalar | character vector

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

**Document — Document link**
"" (default) | string scalar | character vector

Document link for the atomic subchart, specified as a string scalar or 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

| | |
|---|---|
| getParent | Identify parent of object |
| dialog | Open properties dialog box |
| isCommented | Determine if graphical object is commented out |
| view | Display object in editing environment |
| highlight | Highlight graphical object |
| fitToView | Zoom in on graphical object |

## 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";
```

**Add Exit Port and Junction to Atomic Subchart**

Find the `Stateflow.AtomicSubchart` object that corresponds to the atomic subchart A in the chart ch.

```
atomicSubchart = find(ch,"-isa","Stateflow.AtomicSubchart",Name="A");
```

Add an exit junction to the atomic subchart. Use the `Subchart` property of the atomic subchart as the parent of the exit junction. Display the value of the `PortType` property of the exit junction.

```
exitJunction = Stateflow.Port(atomicSubchart.Subchart,"ExitJunction");
exitJunction.PortType
```

```
ans =

    'ExitJunction'
```

Set the label of the exit junction to `"exit"`.

```
exitJunction.labelString = "exit";
```

Find the `Stateflow.Port` object for the matching exit port. Display the value of the `PortType` property of the exit port.

```
exitPort = Stateflow.findMatchingPort(exitJunction);
exitPort.PortType
```

```
ans =

    'ExitPort'
```

Display the label of the exit port.

```
exitPort.labelString
```

```
ans =

    'exit'
```

## See Also

**Functions**
find | Stateflow.findMatchingPort

**Objects**
Stateflow.Box | Stateflow.Chart | Stateflow.Port | Stateflow.State

**Topics**
"Overview of the Stateflow API" on page 1-2
"Create Reusable Subcomponents by Using Atomic Subcharts"
"Summary of Stateflow API Objects and Properties" on page 1-34

**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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

**Name — Name of box**
"" (default) | string scalar | character vector

Name of the box, specified as a string scalar or character vector.

**LabelString — Label for box**
"?" (default) | string scalar | character vector

Label for the box, specified as a string scalar or 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 "Comment Out Objects in a Stateflow 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 state, box, or function that contains it.

**CommentText — Comment text**
"" (default) | string scalar | character vector

Comment text for the box, specified as a string scalar or 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) | string scalar | character vector

Description for the box, specified as a string scalar or character vector.

**Document — Document link**
"" (default) | string scalar | character vector

Document link for the box, specified as a string scalar or 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
find             Identify specified objects in hierarchy
getChildren    Identify children of object
getParent      Identify parent of object
dialog          Open properties dialog box
isCommented   Determine if graphical object is commented out
view            Display object in editing environment
highlight      Highlight graphical object
fitToView      Zoom in on graphical object

## 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"
"Summary of Stateflow API Objects and Properties" on page 1-34

**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 | setdiff | sfclipboard

**Objects**
Stateflow.State

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**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 "Model Finite State Machines by Using Stateflow Charts" and "Create Charts by Using the Stateflow API" on page 1-27.

## 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:

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

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

**Name — Name of chart**
`"Chart"` (default) | string scalar | character vector

Name of the chart, specified as a string scalar or 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`). 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 "Define Exclusive and Parallel Modes by Using State 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
string scalar | character vector

Name of the active state data object for the chart, specified as a string scalar or 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
string scalar | character vector

Name of the enumerated data type for the active state data object for the chart, specified as a string scalar or 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) | string scalar | character vector

Sample time for activating the chart, specified as a string scalar or 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**
string scalar | character vector

Default `fimath` object, specified as a string scalar or 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**

**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 — Debugger properties**
Stateflow.ChartDebug object

Debugger properties for the chart, specified as a Stateflow.ChartDebug object with this property:

- **Breakpoints.OnEntry** — Whether to set the On Chart Entry breakpoint, specified as a numeric or logical 1 (true) or 0 (false).

For more information, see "Set Breakpoints to Debug Charts".

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.

**StateFont — Font for state labels**
Stateflow.StateFont object

Font for the box, function, and state labels in the chart, specified as a `Stateflow.StateFont` object with these properties:

- **Name** — Font name, specified as a string scalar or character vector. This property also determines the font for annotations in the chart.
- **Angle** — Font angle, specified as `"NORMAL"` or `"ITALIC"`.
- **Weight** — Font weight, specified as `"NORMAL"` or `"BOLD"`.
- **Size** — Default font size for new boxes, functions, and states, specified as a scalar. This property also determines the default font size for new annotations in the chart.

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

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

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

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

**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.

**TransitionFont — Font for transition labels**
Stateflow.TransFont object

Font for the transition labels in the chart, specified as a `Stateflow.TransFont` object with these properties:

- **Name** — Font name, specified as a string scalar or character vector.
- **Angle** — Font angle, specified as `"NORMAL"` or `"ITALIC"`.
- **Weight** — Font weight, specified as `"NORMAL"` or `"BOLD"`.
- **Size** — Default font size for new transitions, specified as a scalar.

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

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

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

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

**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.

**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) | string scalar | character vector

Description for the chart, specified as a string scalar or character vector.

`Document` **— Document link**
`""` (default) | string scalar | character vector

Document link for the chart, specified as a string scalar or 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

| | |
|---|---|
| find | Identify specified objects in hierarchy |
| getChildren | Identify children of object |
| defaultTransitions | Identify default transitions in specified object |
| dialog | Open properties dialog box |
| view | Display object in editing environment |
| fitToView | Zoom in on graphical object |

## 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**
sfnew | sfroot | add_block

**Topics**
"Overview of the Stateflow API" on page 1-2
"Model Finite State Machines by Using Stateflow Charts"
"Specify Properties for Stateflow Charts"
"Create Charts by Using the Stateflow API" on page 1-27
"Summary of Stateflow API Objects and Properties" on page 1-34

**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.SLFunction`
- `Stateflow.State`
- `Stateflow.TruthTable`

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Interface**

**`Name` — Name of data object**
`"data"` (default) | string scalar | character vector

Name of the data object, specified as a string scalar or 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 string scalar or 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".

To specify the initial value of the data object, use the Props.InitialValue property.

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".

**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.

**Data Specification**

**DataType — Type of data object**
`"Inherit: From definition in chart"` (default) | `"double"` | `"single"` | `"int32"` | `"uint32"` | `"boolean"` | ...

Type of the data object, specified as a string scalar or 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"`
  - `"string"`
  - `"ml"` (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 1-27.

**Props — Data specification properties**
`Stateflow.DataProps` object

Data specification properties, specified as a `Stateflow.DataProps` object with these properties:

- **Type.Method** — Method for setting the type of the data object, specified as a string scalar or character vector.

- 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".

- **Type.BusObject** — Name of the `Simulink.Bus` object that defines the data object, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Bus Object"`. For more information, see "Access Bus Signals Through Stateflow Structures".

- **Type.EnumType** — Name of the enumerated type that defines the data object, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Enumerated"`. For more information, see "Reference Values by Name by Using Enumerated Data".

- **Type.Expression** — Expression that evaluates to the data type of the data object, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Expression"`. For more information, see "Specify Data Properties by Using MATLAB Expressions".

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

- **Type.WordLength** — Word length, in bits, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts".

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

- **Type.Fixpt.FractionLength** — Fraction length, in bits, specified as a string scalar or character vector. This property applies only when the `Type.Method` property is `"Fixed point"` and the `Type.Fixpt.ScalingMode` property is `"Binary point"`.

- **Type.Fixpt.Slope** — Slope, specified as a string scalar or character vector. This property applies only when the `Type.Method` property is `"Fixed point"` and the `Type.Fixpt.ScalingMode` property is `"Slope and bias"`.

- **Type.Fixpt.Bias** — Bias, specified as a string scalar or character vector. This property applies only when the `Type.Method` property is `"Fixed point"` and the `Type.Fixpt.ScalingMode` property is `"Slope and bias"`.

- **Type.Fixpt.Lock** — Whether to prevent replacement of the fixed-point type with an autoscaled type chosen by the Fixed-Point Tool (Fixed-Point Designer), specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `Type.Method` property of the data object is `"Fixed point"`.

- **Array.Size** — Size of the data object, specified as a string scalar or character vector. For more information, see "Specify Size of Stateflow Data".

- **Array.IsDynamic** — Whether the data object has variable size, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property is equivalent to the **Variable Size** check box in the **Property Inspector**, the Model Explorer, or the Data properties dialog box. For more information, see "Declare Variable-Size Data in Stateflow Charts".

- **`Array.FirstIndex`** — Index for the first element of the array data object, specified as a string scalar or 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".
- **`Complexity`** — Whether the data object accepts complex values, specified as `"On"` or `"Off"`. For more information, see "Complex Data in Stateflow Charts".
- **`InitialValue`** — Initial value, specified as a string scalar or character vector. For more information, see "Initial value".
- **`Range.Minimum`** — Minimum value, specified as a string scalar or character vector. For more information, see "Limit range".
- **`Range.Maximum`** — Maximum value, specified as a string scalar or character vector. For more information, see "Limit range".
- **`ResolveToSignalObject`** — 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".
- **`Unit.Name`** — Unit of measurement, specified as a string scalar or character vector. This property applies only to input and output data. For more information, see "Specify Units for Stateflow Data".

**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.

**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.

**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 and Test Point Monitoring**

**LoggingInfo — Signal logging properties**
`Stateflow.SigLoggingInfo` object

Signal logging properties for the data object, specified as a `Stateflow.SigLoggingInfo` object with these properties:

- **DataLogging** — Whether to enable signal logging, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **DecimateData** — Whether to limit the amount of logged data, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **Decimation** — Decimation interval, specified as an integer scalar. This property applies only when the `DecimateData` property is `true`.
- **LimitDataPoints** — Whether to limit the number of data points to log, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **MaxPoints** — Maximum number of data points to log, specified as an integer scalar. This property applies only when the `LimitDataPoints` property is `true`.
- **NameMode** — Source of the signal name, specified as `"SignalName"` or `"Custom"`.
- **LoggingName** — Custom signal name, specified as a string scalar or character vector. This property applies only when the `NameMode` property is `"Custom"`.

Signal logging saves the values of the data object to the MATLAB workspace during simulation. For more information, see "Log Simulation Output for States and Data".

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

**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`). You can monitor testpoints with a floating scope during simulation. You can also log test point values to the MATLAB workspace. For more information, see "Monitor Test Points in Stateflow Charts".

**Debugging**

**Debug — Debugger properties**
`Stateflow.DataDebug` object

Debugger properties for the data object, specified as a `Stateflow.DataDebug` object with this property:

- **Watch** — 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".

Example: `data.Debug.Watch = true;`

**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) | string scalar | character vector

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

**`Document` — Document link**
"" (default) | string scalar | character vector

Document link for the data object, specified as a string scalar or 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

getParent    Identify parent of object
dialog       Open properties dialog box
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";
```

## Compatibility Considerations

**Stateflow no longer supports creating machine-parented data**
*Errors starting in R2021b*

Starting in R2021b, the `Stateflow.Data` function does not support arguments of type `Stateflow.Machine`. The presence of machine-parented data in a model prevents the reuse of generated code and other code optimizations. This type of data is also incompatible with many Simulink and Stateflow features. To make Stateflow data accessible to other charts and blocks in a model, use data store memory. For more information, see "Best Practices for Using Data in Charts" and "Access Data Store Memory from a Chart".

## See Also

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

**Topics**
"Overview of the Stateflow API" on page 1-2
"Add Stateflow Data"
"Set Data Properties"
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# Stateflow.DataArray

Array properties for data and messages

## Description

Use a `Stateflow.DataArray` object to specify the array properties for a data object or message.

## Creation

Each data object and message has its own `Stateflow.DataArray` object. To access the `Stateflow.DataArray` object, use the `Props.Array` property for the `Stateflow.Data` or `Stateflow.Message` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

### `Size` — Size
`"-1"` (default) | string scalar | character vector

Size of the data object or message data, specified as a string scalar or character vector. For more information, see "Specify Size of Stateflow Data".

### `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`). This property is equivalent to the **Variable Size** check box in the **Property Inspector**, the Model Explorer, or the Data properties dialog box. For more information, see "Declare Variable-Size Data in Stateflow Charts".

### `FirstIndex` — Index for first element of array
string scalar | character vector

Index for the first element of the array data object, specified as a string scalar or 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".

## Examples

### Specify Size of Data

Access the `Stateflow.Props` and `Stateflow.DataArray` objects for the `Stateflow.Data` object x.

```
properties = x.Props;
array = properties.Array;
```

Specify the size of the data object.

```
array.size = "[2 3]";
```

## See Also

`Stateflow.Data` | `Stateflow.Message`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34
"Specify Size of Stateflow Data"
"Declare Variable-Size Data in Stateflow Charts"
"Save final value to base workspace"

**Introduced before R2006a**

# Stateflow.DataDebug

Debugger properties for data

## Description

Use a `Stateflow.DataDebug` object to specify the debugger properties for a data object.

## Creation

Each data object has its own `Stateflow.DataDebug` object. To access the `Stateflow.DataDebug` object, use the `Debug` property for the `Stateflow.Data` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

### `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".

## Examples

### Add Data to Breakpoints and Watch Window

Access the `Stateflow.DataDebug` object for the `Stateflow.Data` object x.

```
debug = x.Debug;
```

Add the data object to the Breakpoints and Watch window.

```
debug.Watch = true;
```

## See Also
`Stateflow.Data`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34
"View Data in the Breakpoints and Watch Window"

**Introduced before R2006a**

# Stateflow.DataProps

Data specification properties for data and messages

## Description

Use a `Stateflow.DataProps` object to specify the data properties for a data object or message.

## Creation

Each data object and message has its own `Stateflow.DataProps` object. To access the `Stateflow.DataProps` object, use the `Props` property for the `Stateflow.Data` or `Stateflow.Message` object.

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Type — Data type properties**
Stateflow.DataType object

Data type properties, specified as a `Stateflow.DataType` object with these properties:

- **Method** — Method for setting the type of the data object or message, specified as a string scalar or character vector.

  - 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"`.

  - For messages, use `"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 Model Explorer and the Data properties dialog box. For more information, see "Specify Type of Stateflow Data".

- **BusObject** — Name of the `Simulink.Bus` object that defines the data object or message data, specified as a string scalar or character vector. This property applies only when the `Method` property of the data object is `"Bus Object"`. For more information, see "Access Bus Signals Through Stateflow Structures".

- **EnumType** — Name of the enumerated type that defines the data object or message data, specified as a string scalar or character vector. This property applies only when the `Method` property of the data object is `"Enumerated"`. For more information, see "Reference Values by Name by Using Enumerated Data".

- **Expression** — Expression that evaluates to the data type of the data object or message data, specified as a string scalar or character vector. This property applies only when the `Method` property of the data object is `"Expression"`. For more information, see "Specify Data Properties by Using MATLAB Expressions".

- **Signed** — Signedness, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts".

- **WordLength** — Word length, in bits, specified as a string scalar or character vector. This property applies only when the `Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts".

- **Fixpt.ScalingMode** — 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 `Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts".

- **Fixpt.FractionLength** — Fraction length, in bits, specified as a string scalar or character vector. This property applies only when the `Method` property is `"Fixed point"` and the `Fixpt.ScalingMode` property is `"Binary point"`.

- **Fixpt.Slope** — Slope, specified as a string scalar or character vector. This property applies only when the `Method` property is `"Fixed point"` and the `Fixpt.ScalingMode` property is `"Slope and bias"`.

- **Fixpt.Bias** — Bias, specified as a string scalar or character vector. This property applies only when the `Method` property is `"Fixed point"` and the `Fixpt.ScalingMode` property is `"Slope and bias"`.

- **Fixpt.Lock** — Whether to prevent replacement of the fixed-point type with an autoscaled type chosen by the Fixed-Point Tool (Fixed-Point Designer), specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `Method` property of the data object is `"Fixed point"`.

**Array — Array properties**
Stateflow.DataArray object

Array properties, specified as a `Stateflow.DataArray` object with these properties:

- **Size** — Size of the data object or message data, specified as a string scalar or character vector. For more information, see "Specify Size of Stateflow Data".

- **IsDynamic** — Whether the data object has variable size, specified as a numeric or logical 1 (`true`) or 0 (`false`). This property is equivalent to the **Variable Size** check box in the **Property Inspector**, the Model Explorer, or the Data properties dialog box. For more information, see "Declare Variable-Size Data in Stateflow Charts".

- **FirstIndex** — Index for the first element of the array data object, specified as a string scalar or 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".

**Complexity — Whether data object or message accepts complex values**
"Off" (default) | "On"

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

**InitialValue — Initial value**
"" (default) | string scalar | character vector

Initial value, specified as a string scalar or character vector.

**Range — Range of acceptable values**
`Stateflow.DataRange` object

Range of acceptable values for the data object, specified as a `Stateflow.DataRange` object with these properties:

- `Minimum` — Minimum value, specified as a string scalar or character vector.
- `Maximum` — Maximum value, specified as a string scalar or character vector.

This property does not apply to message data. For more information, see "Limit range".

**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`). This property does not apply to message data. For more information, see "Resolve Data Properties from Simulink Signal Objects".

**Unit — Unit of measurement for input and output data**
`Stateflow.Unit` object

Unit of measurement for input and output data objects, specified as a `Stateflow.Unit` object with this property:

- `Name` — Name of the unit of measurement, specified as a string scalar or character vector.

This property applies only to input and output data. For more information, see "Specify Units for Stateflow Data".

## Examples

### Specify Fixed-Point Data Type

Access the `Stateflow.Props`, `Stateflow.DataType`, and `Stateflow.FixptType` objects for the `Stateflow.Data` object x.

```
properties = x.Props;
type = properties.Type;
fixpt = type.Fixpt;
```

Specify the fixed-point properties.

```
type.Method = "Fixed point";
type.Signed = true;
type.WordLength = "5";
fixpt.ScalingMode = "Binary point";
fixpt.FractionLength = "2";
```

Verify the data type.

```
x.DataType
```

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

### Specify Size of Data

Access the `Stateflow.Props` and `Stateflow.DataArray` objects for the `Stateflow.Data` object x.

```
properties = x.Props;
array = properties.Array;
```

Specify the size of the data object.

```
array.size = "[2 3]";
```

### Specify Range of Values for Data

Access the `Stateflow.Props` and `Stateflow.DataRanges` objects for the `Stateflow.Data` object x.

```
properties = x.Props;
range = properties.Range;
```

Specify the minimum and maximum acceptable values.

```
range.Minimum = "0";
range.Maximum = "1024";
```

### Specify Units for Data

Access the `Stateflow.Props` and `Stateflow.Unit` objects for the `Stateflow.Data` object x.

```
properties = x.Props;
unit = properties.Unit;
```

Specify the units as meters.

```
unit.Name = "m";
```

## See Also
`Stateflow.Data` | `Stateflow.Message`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# Stateflow.DataRange

Range of acceptable values for data

## Description

Use a `Stateflow.DataRange` object to specify the range of acceptable values for a data object. For more information, see "Limit range".

## Creation

Each data object and message has its own `Stateflow.DataRange` object. However, the object only applies for `Stateflow.Data` objects. To access the `Stateflow.DataRange` object, use the `Props.Range` property for the `Stateflow.Data` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

### Minimum — Minimum value
`""` (default) | string scalar | character vector

Minimum value, specified as a string scalar or character vector.

### Maximum — Maximum value
`""` (default) | string scalar | character vector

Maximum value, specified as a string scalar or character vector.

## Examples

### Specify Range of Values for Data

Access the `Stateflow.Props` and `Stateflow.DataRanges` objects for the `Stateflow.Data` object `x`.

```
properties = x.Props;
range = properties.Range;
```

Specify the minimum and maximum acceptable values.

```
range.Minimum = "0";
range.Maximum = "1024";
```

## See Also
`Stateflow.Data`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# Stateflow.DataType

Data type properties for data and messages

## Description

Use a `Stateflow.DataType` object to specify the data type properties for a data object or message.

## Creation

Each data object and message has its own `Stateflow.DataType` object. To access the `Stateflow.DataType` object, use the `Props.Type` property for the `Stateflow.Data` or `Stateflow.Message` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

### Method — Method for setting data type
`"Inherited"` (default) | `"Built-in"` | `"Bus Object"` | `"Enumerated"` | `"Expression"` | `"Fixed point"`

Method for setting the type of the data object or message, specified as a string scalar or character vector.

- 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"`.

- For messages, use `"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 Model Explorer and the Data properties dialog box. For more information, see "Specify Type of Stateflow Data".

### BusObject — Name of `Simulink.Bus` object
`""` (default) | string scalar | character vector

Name of the `Simulink.Bus` object that defines the data object or message data, specified as a string scalar or character vector. This property applies only when the `Method` property of the data object is `"Bus Object"`. For more information, see "Access Bus Signals Through Stateflow Structures".

### EnumType — Name of enumerated type
`""` (default) | string scalar | character vector

Name of the enumerated type that defines the data object or message data, specified as a string scalar or character vector. This property applies only when the `Method` property of the data object is `"Enumerated"`. For more information, see "Reference Values by Name by Using Enumerated Data".

**Expression — Expression that evaluates to data type**
`""` (default) | string scalar | character vector

Expression that evaluates to the data type of the data object or message data, specified as a string scalar or character vector. This property applies only when the `Method` property of the data object is `"Expression"`. For more information, see "Specify Data Properties by Using MATLAB Expressions".

**Signed — Signedness**
`true` or 1 (default) | `false` or 0

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

**WordLength — Word length**
`"16"` (default) | string scalar | character vector

Word length, in bits, specified as a string scalar or character vector. This property applies only when the `Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts".

**Fixpt — Fixed-point properties**
`Stateflow.FixptType` object

Fixed-point properties, specified as a `Stateflow.FixptType` object with these properties:

- **ScalingMode** — Method for scaling the fixed-point data object or message data, specified as `"Binary point"`, `"Slope and bias"`, or `"None"`.
- **FractionLength** — Fraction length, in bits, specified as a string scalar or character vector. This property applies only when the `ScalingMode` property is `"Binary point"`.
- **Slope** — Slope, specified as a string scalar or character vector. This property applies only when the `ScalingMode` property is `"Slope and bias"`.
- **Bias** — Bias, specified as a string scalar or character vector. This property applies only when the `ScalingMode` property is `"Slope and bias"`.
- **Lock** — Whether to prevent replacement of the fixed-point type with an autoscaled type chosen by the Fixed-Point Tool (Fixed-Point Designer), specified as a numeric or logical 1 (`true`) or 0 (`false`).

This property applies only when the `Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts".

## Examples

### Specify Fixed-Point Data Type

Access the `Stateflow.Props`, `Stateflow.DataType`, and `Stateflow.FixptType` objects for the `Stateflow.Data` object x.

```
properties = x.Props;
type = properties.Type;
fixpt = type.Fixpt;
```

Specify the fixed-point properties.

```
type.Method = "Fixed point";
type.Signed = true;
type.WordLength = "5";
fixpt.ScalingMode = "Binary point";
fixpt.FractionLength = "2";
```

Verify the data type.

```
x.DataType
```

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

## See Also
Stateflow.Data | Stateflow.Message

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**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
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
"Summary of Stateflow API Objects and Properties" on page 1-34

**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 is 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. See "Create and Configure MATLAB S-Functions" (Simulink).

For more information, see "Implement MATLAB Functions in Simulink with MATLAB Function Blocks" (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 Blocks 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` to add a MATLAB Function with the name `MATLAB Function` to a model called `myModel`:

```
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:

```
block = find(sfroot,"-isa","Stateflow.EMChart", ...
    "Path","myModel/MATLAB Function");
```

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API

objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

### `Name` — Name of MATLAB Function block
"MATLAB Function" (default) | string scalar | character vector

Name of the MATLAB Function block, specified as a string scalar or character vector.

### `Script` — Code for MATLAB Function block
string scalar | character vector

Code for the MATLAB Function block, specified as a string scalar or 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 function `newline` to create newline characters:

```
str = "function y=f(x)" + newline + ...
    "y=x+1;" + newline + ...
    "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`). For more information, see "Declare Variable-Size 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).

### `VectorOutputs1D` — Whether MATLAB Function block outputs column vectors as one-dimensional data
false or 0 (default) | true or 1

Whether the MATLAB Function block outputs column vectors as one-dimensional data, specified as a numeric or logical `0` (`false`) or `1` (`true`). For more information, see "Interpret output column vectors as one-dimensional data" (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) | string scalar | character vector

Sample time for activating the MATLAB Function block, specified as a string scalar or 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**
string scalar | character vector

Default `fimath` object, specified as a string scalar or 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**
string scalar | 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) | string scalar | character vector

Description for the MATLAB Function block, specified as a string scalar or character vector.

**Document — Document link**
`""` (default) | string scalar | character vector

Document link for the MATLAB Function block, specified as a string scalar or 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

| | |
|---|---|
| find | Identify specified objects in hierarchy |
| getChildren | Identify children of object |
| dialog | Open properties dialog box |
| view | Display object in editing environment |

## Examples

### Program MATLAB Function Block

Access the `Stateflow.EMChart` object for a MATLAB Function block named `My Function` in a model called `myModel`.

```
block = find(sfroot,"-isa","Stateflow.EMChart", ...
    "Path","myModel/My Function");
```

Store the MATLAB code to calculate the mean and standard deviation for a vector of values as a string scalar.

```
str = "function [mean,stdev] = stats(vals)" + newline + ...
    "% Calculates a statistical mean and a standard" + newline + ...
    "% deviation for the values in vals." + newline + newline + ...
    "len = length(vals);" + newline + ...
    "mean = avg(vals,len);" + newline + ...
    "stdev = sqrt(sum(((vals-avg(vals,len)).^2))/len);" + newline + ...
    "plot(vals,""-+"");" + newline + newline + ...
    "function mean = avg(array,size)" + newline + ...
    "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 named My Function.

```
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",blockPath);
block.Script = fileread("myFunction.m");
```

**Find Number of MATLAB Function Blocks in Model**

Open a Simulink model called myModel.

```
open_system("myModel")
```

Find the MATLAB Function blocks in the model, including the block in the library.

```
blocks = find(sfroot,"-isa","Stateflow.EMChart");
```

**Note** This command finds objects for the MATLAB Function blocks in all open models and libraries. To find only the MATLAB Function blocks in myModel, close every file except myModel or replace sfroot with get_param("myModel","Object").

Count the number of blocks.

```
numel(blocks)
```

## See Also

**Blocks**
MATLAB Function

**Functions**
sfroot | add_block | fileread | numel

**Objects**
MATLABFunctionConfiguration

**Topics**
"Overview of the Stateflow API" on page 1-2
"Specify MATLAB Function Block Properties" (Simulink)
"Summary of Stateflow API Objects and Properties" on page 1-34

**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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

**Name — Name of MATLAB function**
"" (default) | string scalar | character vector

Name of the MATLAB function, specified as a string scalar or character vector.

**LabelString — Label for MATLAB function**
"?" (default) | string scalar | character vector

Label for the MATLAB function, specified as a string scalar or character vector.

**Script — Code for MATLAB function**
string scalar | character vector

Code for the MATLAB function, specified as a string scalar or 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 function `newline` to create newline characters:

  ```
  str = "function y=f(x)" + newline + ...
      "y=x+1;" + newline + ...
      "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 "Comment Out Objects in a Stateflow 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 state, box, or function that contains it.

**CommentText — Comment text**
"" (default) | string scalar | character vector

Comment text for the MATLAB function, specified as a string scalar or 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 MATLAB 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**
string scalar | character vector

Default `fimath` object, specified as a string scalar or 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 as long as the function is not part of a recursion.

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) | string scalar | character vector

Description for the MATLAB function, specified as a string scalar or character vector.

**`Document` — Document link**
"" (default) | string scalar | character vector

Document link for the MATLAB function, specified as a string scalar or 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

| | |
|---|---|
| find | Identify specified objects in hierarchy |
| getChildren | Identify children of object |
| getParent | Identify parent of object |
| dialog | Open properties dialog box |
| isCommented | Determine if graphical object is commented out |
| view | Display object in editing environment |
| highlight | Highlight graphical object |
| fitToView | Zoom in on graphical object |

## 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"
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# Stateflow.ChartBreakpoints

Breakpoint properties for chart or state transition table

## Description

Use a `Stateflow.ChartBreakpoints` object to specify the breakpoint properties for a chart or state transition table. For more information, see "Set Breakpoints to Debug Charts".

## Creation

Each chart and state transition table has its own `Stateflow.ChartBreakpoints` object. To access the `Stateflow.ChartBreakpoints` object, use the `Debug.Breakpoints` property of the `Stateflow.Chart` or `Stateflow.StateTransitionTableChart` object.

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

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

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

### Examples

#### Set Breakpoint for Chart

Access the `Stateflow.ChartDebug` and `Stateflow.ChartBreakpoints` objects for the `Stateflow.Chart` object `ch`.

```
debug = ch.Debug;
breakpoints = debug.Breakpoints;
```

Set the `On Chart Entry` breakpoint.

```
breakpoints.OnEntry = true;
```

## See Also
`Stateflow.Chart` | `Stateflow.StateTransitionTableChart`

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

"Summary of Stateflow API Objects and Properties" on page 1-34
"Set Breakpoints to Debug Charts"

**Introduced before R2006a**

# Stateflow.ChartDebug

Debugger properties for chart or state transition table

## Description

Use a `Stateflow.ChartDebug` object to specify the debugger properties for a chart or state transition table.

## Creation

Each chart and state transition table has its own `Stateflow.ChartDebug` object. To access the `Stateflow.ChartDebug` object, use the `Debug` property for the `Stateflow.Chart` or `Stateflow.StateTransitionTableChart` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Breakpoints — Breakpoint properties**
Stateflow.ChartBreakpoints object

Breakpoint properties for the chart or state transition table, specified as a `Stateflow.ChartBreakpoints` object with this property:

- **OnEntry** — Whether to set the `On Chart Entry` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).

For more information, see "Set Breakpoints to Debug Charts".

## Examples

**Set Breakpoint for Chart**

Access the `Stateflow.ChartDebug` and `Stateflow.ChartBreakpoints` objects for the `Stateflow.Chart` object ch.

```
debug = ch.Debug;
breakpoints = debug.Breakpoints;
```

Set the `On Chart Entry` breakpoint.

```
breakpoints.OnEntry = true;
```

## See Also

Stateflow.Chart | Stateflow.StateTransitionTableChart

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34
"Set Breakpoints to Debug Charts"

**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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Interface**

**Name — Name of event**
"event" (default) | string scalar | character vector

Name of the event, specified as a string scalar or 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 string scalar or 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 — Debugger properties**
Stateflow.EventDebug object

Debugger properties for the event, specified as a Stateflow.EventDebug object with these properties:

• **Breakpoints.StartBroadcast** — Whether to set the Start of Broadcast breakpoint, specified as a numeric or logical 1 (true) or 0 (false).

• **Breakpoints.EndBroadcast** — Whether to set the End of Broadcast breakpoint, specified as a numeric or logical 1 (true) or 0 (false).

For more information, see "Set Breakpoints to Debug Charts".

Example: event.Debug.Breakpoints.StartBroadcast = true;

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) | string scalar | character vector

Description for the event, specified as a string scalar or character vector.

**Document — Document link**
"" (default) | string scalar | character vector

Document link for the event, specified as a string scalar or 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

getParent    Identify parent of object
dialog       Open properties dialog box
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"

**Introduced before R2006a**

# Stateflow.EventBreakpoints

Breakpoint properties for event

## Description

Use a `Stateflow.EventBreakpoints` object to specify the breakpoint properties for an event. For more information, see "Set Breakpoints to Debug Charts".

## Creation

Each event has its own `Stateflow.EventBreakpoints` object. To access the `Stateflow.EventBreakpoints` object, use the `Debug.Breakpoints` property of the `Stateflow.Event` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**StartBroadcast — Whether to set `Start of Broadcast` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `Start of Broadcast` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**EndBroadcast — Whether to set `End of Broadcast` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `End of Broadcast` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).

## Examples

**Set Breakpoints for Event**

Access the `Stateflow.EventDebug` and `Stateflow.EventBreakpoints` objects for the `Stateflow.Event` object `event`.

```
debug = event.Debug;
breakpoints = debug.Breakpoints;
```

Set the `Start of Broadcast` and `End of Broadcast` breakpoints.

```
breakpoints.StartBroadcast = true;
breakpoints.EndBroadcast = true;
```

## See Also
`Stateflow.Event`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34
"Set Breakpoints to Debug Charts"

**Introduced before R2006a**

# Stateflow.EventDebug

Debugger properties for event

## Description

Use a `Stateflow.EventDebug` object to specify the debugger properties for an event.

## Creation

Each event has its own `Stateflow.EventDebug` object. To access the `Stateflow.EventDebug` object, use the `Debug` property for the `Stateflow.Event` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Breakpoints — Breakpoint properties**
`Stateflow.EventBreakpoints` object

Breakpoint properties for the event, specified as a `Stateflow.EventBreakpoints` object with these properties:

- **StartBroadcast** — Whether to set the `Start of Broadcast` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **EndBroadcast** — Whether to set the `End of Broadcast` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).

For more information, see "Set Breakpoints to Debug Charts".

## Examples

**Set Breakpoints for Event**

Access the `Stateflow.EventDebug` and `Stateflow.EventBreakpoints` objects for the `Stateflow.Event` object `event`.

```
debug = event.Debug;
breakpoints = debug.Breakpoints;
```

Set the `Start of Broadcast` and `End of Broadcast` breakpoints.

```
breakpoints.StartBroadcast = true;
breakpoints.EndBroadcast = true;
```

## See Also

`Stateflow.Event`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34
"Set Breakpoints to Debug Charts"

**Introduced before R2006a**

# Stateflow.FixptType

Fixed-point properties for data and messages

## Description

Use a `Stateflow.FixptType` object to specify the fixed-point properties for a data object or message. For more information, see "Fixed-Point Data in Stateflow Charts".

## Creation

Each data object and message has its own `Stateflow.FixptType` object. To access the `Stateflow.FixptType` object, use the `Props.Type.Fixpt` property for the `Stateflow.Data` or `Stateflow.Message` object.

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**ScalingMode — Method for scaling fixed-point data**
"None" (default) | "Binary point" | "Slope and bias"

Method for scaling the fixed-point data object or message data, specified as `"Binary point"`, `"Slope and bias"`, or `"None"`.

**FractionLength — Fraction length**
"" (default) | string scalar | character vector

Fraction length, in bits, specified as a string scalar or character vector. This property applies only to fixed-point data when the `ScalingMode` property is `"Binary point"`.

**Slope — Slope**
"" (default) | string scalar | character vector

Slope, specified as a string scalar or character vector. This property applies only to fixed-point data when the `ScalingMode` property is `"Slope and bias"`.

**Bias — Bias**
"" (default) | string scalar | character vector

Bias, specified as a string scalar or character vector. This property applies only to fixed-point data when the `ScalingMode` property is `"Slope and bias"`.

**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 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 "Iterative Fixed-Point Conversion Using the Fixed-Point Tool" (Fixed-Point Designer).

## Examples

### Specify Fixed-Point Data Type

Access the `Stateflow.Props`, `Stateflow.DataType`, and `Stateflow.FixptType` objects for the `Stateflow.Data` object x.

```
properties = x.Props;
type = properties.Type;
fixpt = type.Fixpt;
```

Specify the fixed-point properties.

```
type.Method = "Fixed point";
type.Signed = true;
type.WordLength = "5";
fixpt.ScalingMode = "Binary point";
fixpt.FractionLength = "2";
```

Verify the data type.

```
x.DataType

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

## See Also
`Stateflow.Data` | `Stateflow.Message`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

### Content

#### Name — Name of graphical function
"" (default) | string scalar | character vector

Name of the graphical function, specified as a string scalar or character vector.

**LabelString — Label for graphical function**
"?" (default) | string scalar | character vector

Label for the graphical function, specified as a string scalar or 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 "Comment Out Objects in a Stateflow 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 state, box, or function that contains it.

**CommentText — Comment text**
"" (default) | string scalar | character vector

Comment text for the graphical function, specified as a string scalar or 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 graphical 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 graphical 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 graphical 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 graphical 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` — Debugger properties**
`Stateflow.FunctionDebug` object

Debugger properties for the graphical function, specified as a `Stateflow.FunctionDebug` object with this property:

- **`Breakpoints.OnDuring`** — Whether to set the `During Function Call` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).

For more information, see "Set Breakpoints to Debug Charts".

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 as long as the function is not part of a recursion.

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) | string scalar | character vector

Description for the graphical function, specified as a string scalar or character vector.

**Document — Document link**
"" (default) | string scalar | character vector

Document link for the graphical function, specified as a string scalar or 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

| | |
|---|---|
| find | Identify specified objects in hierarchy |
| getChildren | Identify children of object |
| getParent | Identify parent of object |
| defaultTransitions | Identify default transitions in specified object |
| dialog | Open properties dialog box |
| isCommented | Determine if graphical object is commented out |
| view | Display object in editing environment |
| highlight | Highlight graphical object |
| fitToView | Zoom in on graphical object |

## 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"
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# Stateflow.FunctionBreakpoint

Breakpoint properties for graphical or truth table function

## Description

Use a `Stateflow.FunctionBreakpoint` object to specify the breakpoint properties for a graphical or truth table function. For more information, see "Set Breakpoints to Debug Charts".

## Creation

Each graphical and truth table function has its own `Stateflow.FunctionBreakpoint` object. To access the `Stateflow.FunctionBreakpoint` object, use the `Debug.Breakpoints` property of the `Stateflow.Function` or `Stateflow.TruthTable` object.

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**OnDuring — Whether to set `During Function Call` breakpoint**
`false` or 0 (default) | `true` or 1

Whether to set the `During Function Call` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).

## Examples

**Set Breakpoint for Graphical Function**

Access the `Stateflow.FunctionDebug` and `Stateflow.FunctionBreakpoints` objects for the `Stateflow.Function` object `f`.

```
debug = f.Debug;
breakpoints = debug.Breakpoints;
```

Set the `During Function Call` breakpoint.

```
breakpoints.OnDuring = true;
```

## See Also
`Stateflow.Function` | `Stateflow.TruthTable`

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

"Summary of Stateflow API Objects and Properties" on page 1-34
"Set Breakpoints to Debug Charts"

**Introduced before R2006a**

# Stateflow.FunctionDebug

Debugger properties for graphical or truth table function

## Description

Use a `Stateflow.FunctionDebug` object to specify the debugger properties for a graphical or truth table function.

## Creation

Each graphical or truth table function has its own `Stateflow.FunctionDebug` object. To access the `Stateflow.FunctionDebug` object, use the `Debug` property for the `Stateflow.Function` or `Stateflow.TruthTable` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Breakpoints — Breakpoint properties**
Stateflow.FunctionBreakpoint object

Breakpoint properties for the graphical or truth table function, specified as a `Stateflow.FunctionBreakpoint` object with this property:

- **OnDuring** — Whether to set the `During Function Call` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).

For more information, see "Set Breakpoints to Debug Charts".

## Examples

**Set Breakpoint for Graphical Function**

Access the `Stateflow.FunctionDebug` and `Stateflow.FunctionBreakpoints` objects for the `Stateflow.Function` object `f`.

```
debug = f.Debug;
breakpoints = debug.Breakpoints;
```

Set the `During Function Call` breakpoint.

```
breakpoints.OnDuring = true;
```

## See Also

Stateflow.Function | Stateflow.TruthTable

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34
"Set Breakpoints to Debug Charts"

**Introduced before R2006a**

# Stateflow.Junction

Connective or history 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 "Resume Prior Substate 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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**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 "Comment Out Objects in a Stateflow 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 state, box, or function that contains it.

**CommentText — Comment text**
"" (default) | string scalar | character vector

Comment text added to the junction, specified as a string scalar or 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 — Position and size of junction**
Stateflow.JunctionPosition object

Position and size of the junction, specified as a Stateflow.JunctionPosition object with these properties:

- **Center** — 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.
- **Radius** — Radius of the junction, specified as a scalar.

Example: junction.Position.Center = [31.41 27.18];

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) | string scalar | character vector

Description for the junction, specified as a string scalar or character vector.

**Document — Document link**
"" (default) | string scalar | character vector

Document link for the junction, specified as a string scalar or 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

| | |
|---|---|
| getParent | Identify parent of object |
| sinkedTransitions | Identify transitions with specified destination |
| sourcedTransitions | Identify transitions with specified source |
| dialog | Open properties dialog box |
| isCommented | Determine if graphical object is commented out |
| view | Display object in editing environment |
| highlight | Highlight graphical object |
| fitToView | Zoom in on graphical object |

## 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"
"Resume Prior Substate Activity by Using History Junctions"
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# Stateflow.JunctionPosition

Position and size of junctions

## Description

Use a `Stateflow.JunctionPosition` object to control the position and size of a connective or history junction.

## Creation

Each junction has its own `Stateflow.JunctionPosition` object. To access the `Stateflow.JunctionPosition` object, use the `Position` property for the `Stateflow.Junction` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

### 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.

### Radius — Radius of junction
7 (default) | scalar

Radius of the junction, specified as a scalar.

## Examples

### Change Size of Entry Junction

Set the radius of the connective junction `junction` to `10`.

```
junction.Position.Radius = 10;
```

## See Also
`Stateflow.Junction`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**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:

```
machine = find(sfroot,"-isa","Stateflow.Machine",Name="myModel");
```

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**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 — Debugger properties**
Stateflow.MachineDebug object

Debugger properties for charts in the machine, specified as a `Stateflow.MachineDebug` object with these properties:

- **Animation.Enabled** — 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.

- **Animation.Delay** — 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 |

- **Animation.MaintainHighlighting** — 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.Enabled = true;

Example: machine.Debug.Animation.Delay = -1;

**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) | string scalar | character vector

Creator of the machine, specified as a string scalar or character vector.

**Modified — Record of modifications**
`""` (default) | string scalar | character vector

Record of modifications to the machine, specified as a string scalar or character vector.

**Version — Version**
`"none"` (default) | string scalar | character vector

Version of the machine, specified as a string scalar or character vector.

**Description — Description**
`""` (default) | string scalar | character vector

Description for the machine, specified as a string scalar or character vector.

**Document — Document link**
`""` (default) | string scalar | character vector

Document link for the machine, specified as a string scalar or 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.

**2-105**

## Object Functions

find      Identify specified objects in hierarchy
dialog    Open properties dialog box

## Examples

### Update Machine Version

Update the `Modified` and `Version` properties of machine `machine`.

```
machine.Modified = string(datetime);
oldVersion = str2double(machine.Version);
if isnan(oldVersion)
    machine.Version = "1";
else
    machine.Version = string(oldVersion+1);
end
```

## See Also

### Functions
`sfroot`

### Objects
`Stateflow.Chart` | `Stateflow.StateTransitionTableChart` |
`Stateflow.TruthTableChart` | `Stateflow.EMChart`

### Topics
"Overview of the Stateflow API" on page 1-2
"Machine Properties"
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# Stateflow.MachineAnimation

Animation properties for charts in Stateflow machine

## Description

Use a `Stateflow.MachineAnimation` object to specify the animation properties for the charts in a Stateflow machine.

## Creation

Each Stateflow machine has its own `Stateflow.MachineAnimation` object. To access the `Stateflow.MachineAnimation` object, use the `Debug.Animation` property for the `Stateflow.Machine` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

### 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.

### 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 `Enable` property of the machine is `true`.

### MaintainHighlighting — Whether to maintain highlighting of active states
`false` or 0 (default) | `true` or 1

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`).

## Examples

### Set Animation Speed to Lightning Fast

Access the `Stateflow.MachineDebug` and `Stateflow.MachineAnimation` objects for the `Stateflow.Machine` object `machine`.

```
debug = machine.Debug;
animation = debug.Animation;
```

Enable animation and set delay to `-1`.

```
animation.Enabled = true;
animation.Delay = -1;
```

## See Also
`Stateflow.Machine`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# Stateflow.MachineDebug

Debugger properties for charts in Stateflow machine

## Description

Use a `Stateflow.MachineDebug` object to specify the debugger properties for the charts in a Stateflow machine.

## Creation

Each Stateflow machine has its own `Stateflow.MachineDebug` object. To access the `Stateflow.MachineDebug` object, use the `Debug` property for the `Stateflow.Machine` object.

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Animation — Animation properties**
Stateflow.MachineAnimation object

Animation properties for charts in the machine, specified as a `Stateflow.MachineAnimation` object with these properties:

- **Enabled** — 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.
- **Delay** — 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 |

- **MaintainHighlighting** — 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`).

## Examples

**Set Animation Speed to Lightning Fast**

Access the `Stateflow.MachineDebug` and `Stateflow.MachineAnimation` objects for the `Stateflow.Machine` object `machine`.

```
debug = machine.Debug;
animation = debug.Animation;
```

Enable animation and set delay to `-1`.

```
animation.Enabled = true;
animation.Delay = -1;
```

## See Also
`Stateflow.Machine`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

#### Interface

**Name — Name of message**
`"message"` (default) | string scalar | character vector

Name of the message, specified as a string scalar or character vector.

**Scope — Scope of message**
`"Output"` (default) | `"Input"` | `"Local"`

Scope of the message, 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 string scalar or 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"`.

To specify the initial value of the message data, use the `Props.InitialValue` property.

For more information, see "Initial Value".

**Priority — Priority**
`"300"` (default) | string scalar | character vector

Priority for the message, specified as a string scalar or 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 Specification**

**DataType — Data type of message**
`"Inherit: Same as Simulink"` (default) | `"double"` | `"single"` | `"int32"` | `"uint32"` | `"boolean"` | ...

Data type of the message, specified as a string scalar or 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"`
  - `"string"`

- "ml" (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 1-27.

**Props — Data specification properties**
`Stateflow.DataProps` object

Data specification properties, specified as a `Stateflow.DataProps` object with these properties:

- **Type.Method** — 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 Model Explorer and the Data properties dialog box. For more information, see "Specify Type of Stateflow Data".

- **Type.BusObject** — Name of the `Simulink.Bus` object that defines the message data, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Bus Object"`. For more information, see "Access Bus Signals Through Stateflow Structures".

- **Type.EnumType** — Name of the enumerated type that defines the message data, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Enumerated"`. For more information, see "Reference Values by Name by Using Enumerated Data".

- **Type.Expression** — Expression that evaluates to the data type of the message data, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Expression"`. For more information, see "Specify Data Properties by Using MATLAB Expressions".

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

- **Type.WordLength** — Word length, in bits, specified as a string scalar or character vector. This property applies only when the `Type.Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts".

- **Type.Fixpt.ScalingMode** — Method for scaling the fixed-point message data, specified as `"Binary point"`, `"Slope and bias"`, or `"None"`. This property applies only when the `Type.Method` property of the data object is `"Fixed point"`. For more information, see "Fixed-Point Data in Stateflow Charts".

- **Type.Fixpt.FractionLength** — Fraction length, in bits, specified as a string scalar or character vector. This property applies only when the `Type.Method` property is `"Fixed point"` and the `Type.Fixpt.ScalingMode` property is `"Binary point"`.

- **Type.Fixpt.Slope** — Slope, specified as a string scalar or character vector. This property applies only when the `Type.Method` property is `"Fixed point"` and the `Type.Fixpt.ScalingMode` property is `"Slope and bias"`.

- **Type.Fixpt.Bias** — Bias, specified as a string scalar or character vector. This property applies only to when the `Type.Method` property is `"Fixed point"` and the `Type.Fixpt.ScalingMode` property is `"Slope and bias"`.

- **Type.Fixpt.Lock** — Whether to prevent replacement of the fixed-point type with an autoscaled type chosen by the Fixed-Point Tool (Fixed-Point Designer), specified as a numeric or logical 1 (`true`) or 0 (`false`). This property applies only when the `Type.Method` property of the data object is `"Fixed point"`.

- **`Array.Size`** — Size of the message data, specified as a string scalar or character vector. For more information, see "Specify Size of Stateflow Data".
- **`Complexity`** — Whether the message accepts complex values, specified as `"On"` or `"Off"`. For more information, see "Complex Data in Stateflow Charts".
- **`InitialValue`** — Initial value, specified as a string scalar or character vector.

**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.

**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.

**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) | string scalar | character vector

Description for the message, specified as a string scalar or character vector.

**Document — Document link**
`""` (default) | string scalar | character vector

Document link for the message, specified as a string scalar or 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

getParent    Identify parent of object
dialog       Open properties dialog box
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"
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced in R2015b**

# Stateflow.NoteFont

Font properties for annotations

## Description

Use a `Stateflow.NoteFont` object to specify the font properties for an annotation.

## Creation

Each annotation has its own `Stateflow.NoteFont` object. To access the `Stateflow.NoteFont` object, use the `Font` property for the `Stateflow.Annotation` object.

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

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

This property is read-only.

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

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

Font angle, specified as `"NORMAL"` or `"ITALIC"`.

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

Font weight, specified as `"NORMAL"` or `"BOLD"`.

**Size — Font size**
scalar

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

## Examples

**Change Font Properties for Annotation**

Access the `Stateflow.NoteFont` object for the `Stateflow.Annotation` object `annotation`.

```
font = annotation.Font;
```

Set the font angle to italics, the font weight to bold, and the font size to 8.

```
font.Angle = "ITALIC";
font.Weight = "BOLD";
font.Size = 8;
```

## See Also
`Stateflow.Annotation`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# Stateflow.Port

Entry or exit port in state or atomic subchart

# Description

Use `Stateflow.Port` objects to create ports and junctions that provide entry and exit connections across boundaries in the Stateflow hierarchy. Entry and exit ports improve componentization by isolating the transition logic for entering and exiting states. Unlike supertransitions, they can be used in atomic subcharts. For more information, see "Create Entry and Exit Connections Across State Boundaries".

Entry and exit ports are located on the boundary of a state or atomic subchart. Each port has a matching junction that marks the entry or exit point inside the state or atomic subchart. The port and junction are represented by separate `Stateflow.Port` objects.

# Creation

## Syntax

```
port = Stateflow.Port(parent,portType)
```

**Description**

`port = Stateflow.Port(parent,portType)` creates a `Stateflow.Port` object of the specified port type in the parent. The function creates a second `Stateflow.Port` object for the matching entry or exit port on the boundary of the parent. To identify the matching `Stateflow.Port` object, use the function `Stateflow.findMatchingPort`.

**Input Arguments**

**parent — Parent for new entry or exit junction**
`Stateflow.State` object | `Stateflow.Chart` object

Parent for the new entry or exit junction, specified as a `Stateflow.State` object or as a `Stateflow.Chart` object that is the subchart of a `Stateflow.AtomicSubchart` object.

**portType — Type of junction**
`"EntryJunction"` | `"ExitJunction"`

Type of junction, specified as `"EntryJunction"` or `"ExitJunction"`.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

**LabelString — Label for port or junction**
"" (default) | string scalar | character vector

Label for the port or junction, specified as a string scalar or character vector. Changing this property automatically sets the `LabelString` property for the matching `Stateflow.Port` object to the same value.

**PortType — Type of port or junction**
`'EntryJunction'` | `'EntryPort'` | `'ExitJunction'` | `'ExitPort'`

This property is read-only.

Type of port or junction, specified as one of these values:

- `'EntryJunction'` — Entry junction inside a state or atomic subchart
- `'EntryPort'` — Entry port on the boundary of a state or atomic subchart
- `'ExitJunction'` — Exit junction inside a state or atomic subchart
- `'ExitPort'` — Exit port on the boundary of a state or atomic subchart

**IsExplicitlyCommented — Whether to comment out junction and port pair**
`false` or 0 (default) | `true` or 1

Whether to comment out the junction and port pair, specified as a numeric or logical 1 (`true`) or 0 (`false`). Setting this property to `true` is equivalent to right-clicking the entry or exit junction and selecting **Comment Out**. This property applies only to entry and exit junctions. Attempting to set this property on an entry or exit port results in an error. For more information, see "Comment Out Objects in a Stateflow Chart".

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

This property is read-only.

Whether the port or junction is implicitly commented out, specified as a numeric or logical 1 (`true`) or 0 (`false`). The port or junction is implicitly commented out when you comment out a state, box, or function that contains it. Additionally, entry and exit ports are implicitly commented out when you comment out their matching entry or exit junction.

**CommentText — Comment text**
"" (default) | string scalar | character vector

Comment text added to the entry or exit junction, specified as a string scalar or character vector. This property applies only to entry and exit junctions 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 — Position and size of port or junction**
`Stateflow.PortPosition` object

Position and size of the port or junction, specified as a `Stateflow.PortPosition` object with these properties:

- **Center** — Position of the center of the port or junction, specified as a two-element numeric vector `[x y]` of coordinates relative to the upper left corner of the chart.
- **Radius** — Radius of the port or junction, specified as a scalar.

Example: `port.Position.Center = [31.41 27.18];`

Example: `port.Position.Radius = 16.18;`

**LabelPosition — Position and size of port or junction label**
`[-10 -15 2 16]` (default) | `[left top width height]`

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

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

Size of incoming transition arrows, specified as a scalar.

**Hierarchy**

**Chart — Chart that contains port or junction**
`Stateflow.Chart` object

This property is read-only.

Chart that contains the port or junction, specified as a `Stateflow.Chart` object.

**Subviewer — Subviewer for port or junction**
`Stateflow.Chart` object | `Stateflow.State` object | `Stateflow.Box` object

This property is read-only.

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

**Home — Home state or subchart**
`Stateflow.State` object | `Stateflow.AtomicSubchart` object

This property is read-only.

Home state or subchart, specified as a `Stateflow.State` or `Stateflow.AtomicSubchart` object. The home of an entry or exit port is the state or subchart whose boundary contains the port. This property applies only to entry and exit ports.

**Machine — Machine that contains port or junction**
`Stateflow.Machine` object

This property is read-only.

Machine that contains the port or 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 port or junction in the model hierarchy, specified as a character vector.

**Linked — Whether port or junction has matching junction or port**
true or 1 (default) | false or 0

This property is read-only.

Whether the port or junction has a matching junction or port, specified as a numeric or logical 1 (true) or 0 (false). This property is used to detect internal inconsistencies in the chart.

**Identification**

**Description — Description**
"" (default) | string scalar | character vector

Description for the port or junction, specified as a string scalar or character vector.

**Document — Document link**
"" (default) | string scalar | character vector

Document link for the port or junction, specified as a string scalar or character vector.

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

User-defined tag for the port or 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 port or 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

| | |
|---|---|
| getParent | Identify parent of object |
| sinkedTransitions | Identify transitions with specified destination |
| sourcedTransitions | Identify transitions with specified source |
| dialog | Open properties dialog box |

| | |
|---|---|
| isCommented | Determine if graphical object is commented out |
| view | Display object in editing environment |
| highlight | Highlight graphical object |
| fitToView | Zoom in on graphical object |

## Examples

**Add Exit Port and Junction to Atomic Subchart**

Find the `Stateflow.AtomicSubchart` object that corresponds to the atomic subchart A in the chart ch.

```
atomicSubchart = find(ch,"-isa","Stateflow.AtomicSubchart",Name="A");
```

Add an exit junction to the atomic subchart. Use the `Subchart` property of the atomic subchart as the parent of the exit junction. Display the value of the `PortType` property of the exit junction.

```
exitJunction = Stateflow.Port(atomicSubchart.Subchart,"ExitJunction");
exitJunction.PortType
```

```
ans =

    'ExitJunction'
```

Set the label of the exit junction to `"exit"`.

```
exitJunction.labelString = "exit";
```

Find the `Stateflow.Port` object for the matching exit port. Display the value of the `PortType` property of the exit port.

```
exitPort = Stateflow.findMatchingPort(exitJunction);
exitPort.PortType
```

```
ans =

    'ExitPort'
```

Display the label of the exit port.

```
exitPort.labelString
```

```
ans =

    'exit'
```

## Tips

- If you move an entry or exit junction to a different parent, Stateflow deletes the `Stateflow.Port` object for the matching port and creates a `Stateflow.Port` object on the new parent. To identify the new matching port, use the `Stateflow.findMatchingPort` function.

## See Also

**Functions**
find | Stateflow.findMatchingPort

**Objects**
Stateflow.AtomicSubchart | Stateflow.Chart | Stateflow.State

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34
"Create Entry and Exit Connections Across State Boundaries"

**Introduced in R2021b**

# Stateflow.PortPosition

Position and size of entry or exit ports

## Description

Use a `Stateflow.PortPosition` object to control the position and size of an entry or exit port.

## Creation

Each entry or exit port has its own `Stateflow.PortPosition` object. To access the `Stateflow.PortPosition` object, use the `Position` property for the `Stateflow.Port` object.

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

#### Center — Position of center of port or junction
[x y]

Position of the center of the port or junction, specified as a two-element numeric vector [x y] of coordinates relative to the upper left corner of the chart.

#### Radius — Radius of port or junction
scalar

Radius of the port or junction, specified as a scalar.

## Examples

### Change Size of Entry Junction

Set the radius of the entry junction `entryJunction` to `10`.

```
entryJunction.Position.Radius = 10;
```

## See Also
`Stateflow.Port`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced in R2021b**

# Stateflow.SigLoggingInfo

Signal logging properties for states and data

## Description

Use a `Stateflow.SigLoggingInfo` object to specify the signal logging properties for a state or data object. Signal logging saves the self activity of a state or the values of a data object to the MATLAB workspace during simulation. For more information, see "Log Simulation Output for States and Data".

## Creation

Each state, atomic subchart, Simulink based state, and data object has its own `Stateflow.SigLoggingInfo` object. To access the `Stateflow.SigLoggingInfo` object, use the `LoggingInfo` property for the `Stateflow.State`, `Stateflow.AtomicSubchart`, `Stateflow.SimulinkBasedState`, or `Stateflow.Data` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**DataLogging — Whether to enable signal logging**
false or 0 (default) | true or 1

Whether to enable signal logging for the state or data object, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**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 `Decimation` property. For more information, see "Decimation".

**Decimation — Decimation interval**
2 (default) | scalar

Decimation interval, specified as an integer scalar. This property applies only when the `DecimateData` property is `true`. The default value of 2 means that the chart logs every other sample.

**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 `MaxPoints` property. For more information, see "Limit data points to last".

### MaxPoints — Maximum number of data points to log
5000 (default) | scalar

Maximum number of data points to log, specified as an integer scalar. This property applies only when the `LimitDataPoints` property is `true`. The default value of `5000` means the chart logs only the last 5000 data points generated by the simulation.

### NameMode — Source of signal name
"SignalName" (default) | "Custom"

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

- `"SignalName"` — Use the name of the state or data object.
- `"Custom"` — Use the custom signal name specified by the `LoggingName` property.

For more information, see "Logging name".

### LoggingName — Custom signal name
string scalar | character vector

Custom signal name for the state or data object, specified as a string scalar or character vector. This property applies only when the `NameMode` property is `"Custom"`.

## Examples

### Enable Signal Logging for Data

Access the `SigLoggingInfo` object for the `Stateflow.Data` object x.

```
log = x.LoggingInfo;
```

Enable logging for the data object and specify a custom signal name.

```
log.DataLogging = true;
log.NameMode = "Custom";
log.LoggingName = "My Data";
```

## See Also
Stateflow.Data | Stateflow.State | Stateflow.AtomicSubchart | Stateflow.SimulinkBasedState

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34
"Log Simulation Output for States and Data"

**Introduced before R2006a**

# 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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

**Name — Name of Simulink based state**
"" (default) | string scalar | character vector

Name of the Simulink based state, specified as a string scalar or 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 "Comment Out Objects in a Stateflow 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 state or box that contains it.

**CommentText — Comment text**
`""` (default) | string scalar | character vector

Comment text added to the Simulink based state, specified as a string scalar or character vector. This property applies only when the `IsExplicitlyCommented` property is `true`. In the Stateflow Editor, when you point to the comment badge ![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 'AND' or 'OR'. 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**
string scalar | character vector

Name of the active state data object for the Simulink based state, specified as a string scalar or 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 string scalar or character vector. For Simulink based states, the only option is "SelfActivity".

**Signal Logging and Test Point Monitoring**

**LoggingInfo — Signal logging properties**
Stateflow.SigLoggingInfo object

Signal logging properties for the Simulink based state, specified as a `Stateflow.SigLoggingInfo` object with these properties:

- **DataLogging** — Whether to enable signal logging, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **DecimateData** — Whether to limit the amount of logged data, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **Decimation** — Decimation interval, specified as an integer scalar. This property applies only when the `DecimateData` property is `true`.
- **LimitDataPoints** — Whether to limit the number of data points to log, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **MaxPoints** — Maximum number of data points to log, specified as an integer scalar. This property applies only when the `LimitDataPoints` property is `true`.
- **NameMode** — Source of the signal name, specified as `"SignalName"` or `"Custom"`.
- **LoggingName** — Custom signal name, specified as a string scalar or character vector. This property applies only when the `NameMode` property is `"Custom"`.

Signal logging saves the self activity of the Simulink based state to the MATLAB workspace during simulation. For more information, see "Log Simulation Output for States and Data".

Example: `state.LoggingInfo.DataLogging = 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`). You can monitor testpoints with a floating scope during simulation. You can also log test point values to the MATLAB workspace. For more information, see "Monitor Test Points in Stateflow Charts".

**Debugging**

**Debug — Debugger properties**
`Stateflow.StateDebug` object

Debugger properties for the Simulink based state, specified as a `Stateflow.StateDebug` object with these properties:

- **OnEntry** — Whether to set the `On State Entry` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **OnDuring** — Whether to set the `During State` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **OnExit** — Whether to set the `On State Exit` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).

For more information, see "Set Breakpoints to Debug Charts".

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

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

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

**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) | string scalar | character vector

Description for the Simulink based state, specified as a string scalar or character vector.

### `Document` — Document link
`""` (default) | string scalar | character vector

Document link for the Simulink based state, specified as a string scalar or 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

| | |
|---|---|
| getParent | Identify parent of object |
| dialog | Open properties dialog box |
| isCommented | Determine if graphical object is commented out |
| view | Display object in editing environment |
| highlight | Highlight graphical object |
| fitToView | Zoom in on graphical object |

## 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"
"Summary of Stateflow API Objects and Properties" on page 1-34

**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 Functions 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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

**Name — Name of Simulink function**
"" (default) | string scalar | character vector

Name of the Simulink function, specified as a string scalar or character vector.

**LabelString — Label for Simulink function**
"?" (default) | string scalar | character vector

Label for the Simulink function, specified as a string scalar or 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 "Comment Out Objects in a Stateflow 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 state, box, or function that contains it.

**CommentText — Comment text**
"" (default) | string scalar | character vector

Comment text added to the Simulink function, specified as a string scalar or 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 Simulink 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 Simulink 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) | string scalar | character vector

Description for the Simulink function, specified as a string scalar or character vector.

### Document — Document link
`""` (default) | string scalar | character vector

Document link for the Simulink function, specified as a string scalar or 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 Simulink 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

| | |
|---|---|
| find | Identify specified objects in hierarchy |
| getChildren | Identify children of object |
| getParent | Identify parent of object |
| dialog | Open properties dialog box |
| isCommented | Determine if graphical object is commented out |
| view | Display object in editing environment |
| highlight | Highlight graphical object |
| fitToView | Zoom in on graphical object |

## 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 Functions in Stateflow Charts"
"Summary of Stateflow API Objects and Properties" on page 1-34

**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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

### Content

**Name — Name of state**
`""` (default) | string scalar | character vector

Name of the state, specified as a string scalar or character vector.

**LabelString — Label for state**
`"?"` (default) | string scalar | character vector

Label for the state, specified as a string scalar or character vector. For more information, see "Specify Labels in States and Transitions Programmatically" on page 1-24.

**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 1-24. 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 1-24. 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 1-24. 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 1-24. This property is supported only in Moore charts. For more information, see "Design Guidelines 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 1-24. 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 "Comment Out Objects in a Stateflow 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 state or box that contains it.

### CommentText — Comment text
`""` (default) | string scalar | character vector

Comment text added to the state, specified as a string scalar or 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 'AND' or 'OR'. 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**
string scalar | character vector

Name of the active state data object for the state, specified as a string scalar or 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**
string scalar | character vector

Name of the enumerated data type for the active state data object for the state, specified as a string scalar or 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 and Test Point Monitoring**

**`LoggingInfo` — Signal logging properties**
`Stateflow.SigLoggingInfo` object

Signal logging properties for the state, specified as a `Stateflow.SigLoggingInfo` object with these properties:

- **`DataLogging`** — Whether to enable signal logging, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **`DecimateData`** — Whether to limit the amount of logged data, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **`Decimation`** — Decimation interval, specified as an integer scalar. This property applies only when the `DecimateData` property is `true`.
- **`LimitDataPoints`** — Whether to limit the number of data points to log, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **`MaxPoints`** — Maximum number of data points to log, specified as an integer scalar. This property applies only when the `LimitDataPoints` property is `true`.
- **`NameMode`** — Source of the signal name, specified as `"SignalName"` or `"Custom"`.
- **`LoggingName`** — Custom signal name, specified as a string scalar or character vector. This property applies only when the `NameMode` property is `"Custom"`.

Signal logging saves the self activity of the state to the MATLAB workspace during simulation. For more information, see "Log Simulation Output for States and Data".

Example: `state.LoggingInfo.DataLogging = 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). You can monitor testpoints with a floating scope during simulation. You can also log test point values to the MATLAB workspace. For more information, see "Monitor Test Points in Stateflow Charts".

**Debugging**

**Debug — Debugger properties**
Stateflow.StateDebug object

Debugger properties for the state, specified as a Stateflow.StateDebug object with these properties:

- **OnEntry** — Whether to set the On State Entry breakpoint, specified as a numeric or logical 1 (true) or 0 (false).
- **OnDuring** — Whether to set the During State breakpoint, specified as a numeric or logical 1 (true) or 0 (false).
- **OnExit** — Whether to set the On State Exit breakpoint, specified as a numeric or logical 1 (true) or 0 (false).

For more information, see "Set Breakpoints to Debug Charts".

Example: state.Debug.Breakpoints.OnEntry = true;

Example: state.Debug.Breakpoints.OnDuring = true;

Example: state.Debug.Breakpoints.OnExit = true;

**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 static functions.
- **"Inline"** — Calls to state functions are replaced by code as long as the function is not part of a recursion.

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) | string scalar | character vector

Description for the state, specified as a string scalar or character vector.

**Document — Document link**
"" (default) | string scalar | character vector

Document link for the state, specified as a string scalar or 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

| | |
|---|---|
| find | Identify specified objects in hierarchy |
| getChildren | Identify children of object |
| getParent | Identify parent of object |
| defaultTransitions | Identify default transitions in specified object |
| innerTransitions | Identify inner transitions with specified source |
| outerTransitions | Identify outer transitions with specified source |
| sinkedTransitions | Identify transitions with specified destination |
| sourcedTransitions | Identify transitions with specified source |
| dialog | Open properties dialog box |
| isCommented | Determine if graphical object is commented out |
| view | Display object in editing environment |
| highlight | Highlight graphical object |
| fitToView | Zoom in on graphical object |

## 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();");
sA.LabelString = str;
```

- Enter a concatenated text expression that uses the function `newline` to create newline characters:

```
str = "A" + newline + ...
    "en: action1();" + newline + ...
    "du: action2();" + newline + ...
    "en,du: action3();"
sA.LabelString = str;
```
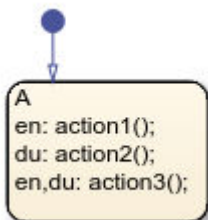


To extract the state name, `entry` action, and `during` action specified by the state label, enter:

```
name = state.Name

name =

    'A'

entry = state.EntryAction

entry =

    ' action1();
      action3();'

during = state.DuringAction

during =

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

For more information, see "Specify Labels in States and Transitions Programmatically" on page 1-24.

**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`.
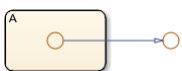
```
st.IsSubchart = false;
st.IsGrouped = false;
```

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 1-24
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# Stateflow.StateBreakpoints

Breakpoint properties for state

## Description

Use a `Stateflow.StateBreakpoints` object to specify the breakpoint properties for a state, atomic subchart, or Simulink based state. For more information, see "Set Breakpoints to Debug Charts".

## Creation

Each state, atomic subchart, and Simulink based state has its own `Stateflow.StateBreakpoints` object. To access the `Stateflow.StateBreakpoints` object, use the `Debug.Breakpoints` property of the `Stateflow.State`, `Stateflow.AtomicSubchart`, or `Stateflow.SimulinkBasedState` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

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

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

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

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

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

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

## Examples

**Set Breakpoints for State**

Access the `Stateflow.StateDebug` and `Stateflow.StateBreakpoints` objects for the `Stateflow.State` object `state`.

```
debug = state.Debug;
breakpoints = debug.Breakpoints;
```

Set the On State Entry, During State, and On State Exit breakpoints.

```
breakpoints.OnEntry = true;
breakpoints.OnDuring = true;
breakpoints.OnExit = true;
```

## See Also
Stateflow.State | Stateflow.SimulinkBasedState | Stateflow.AtomicSubchart

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34
"Set Breakpoints to Debug Charts"

**Introduced before R2006a**

# Stateflow.StateDebug

Debugger properties for state

## Description

Use a `Stateflow.StateDebug` object to specify the debugger properties for a state, atomic subchart, or Simulink based state.

## Creation

Each state, atomic subchart, and Simulink based state has its own `Stateflow.StateDebug` object. To access the `Stateflow.StateDebug` object, use the `Debug` property for the `Stateflow.State`, `Stateflow.AtomicSubchart`, or `Stateflow.SimulinkBasedState` object.

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Breakpoints — Breakpoint properties**
Stateflow.StateBreakpoints object

Breakpoint properties for the state, atomic subchart, or Simulink based state, specified as a `Stateflow.StateBreakpoints` object with these properties:

- **OnEntry** — Whether to set the `On State Entry` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **OnDuring** — Whether to set the `During State` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **OnExit** — Whether to set the `On State Exit` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).

For more information, see "Set Breakpoints to Debug Charts".

## Examples

### Set Breakpoints for State

Access the `Stateflow.StateDebug` and `Stateflow.StateBreakpoints` objects for the `Stateflow.State` object `state`.

```
debug = state.Debug;
breakpoints = debug.Breakpoints;
```

**2-151**

Set the `On State Entry`, `During State`, and `On State Exit` breakpoints.

```
breakpoints.OnEntry = true;
breakpoints.OnDuring = true;
breakpoints.OnExit = true;
```

## See Also

`Stateflow.State` | `Stateflow.SimulinkBasedState` | `Stateflow.AtomicSubchart`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34
"Set Breakpoints to Debug Charts"

**Introduced before R2006a**

# Stateflow.StateFont

Font for box, function, and state labels

## Description

Use a `Stateflow.StateFont` object to specify the font properties for box, function, and state labels in a chart.

## Creation

Each chart has its own `Stateflow.StateFont` object. To access the `Stateflow.StateFont` object, use the `StateFont` property for the `Stateflow.Chart` object.

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Name — Font name**
`"Helvetica"` (default) | string scalar | character vector

Font name, specified as a string scalar or character vector. This property also determines the font for annotations in the chart.

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

Font angle, specified as `"NORMAL"` or `"ITALIC"`.

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

Font weight, specified as `"NORMAL"` or `"BOLD"`.

**Size — Default font size**
12 (default) | scalar

Default font size for new boxes, functions, and states in the chart, specified as a scalar. This property also determines the default font size for new annotations in the chart.

### Examples

#### Change Font Properties for State Labels

Access the `Stateflow.StateFont` object for the `Stateflow.Chart` object ch.

```
font = ch.StateFont;
```

Set the font for box, function, and state labels to Arial. Set the font angle to italics and the font weight to bold. Set the default font size to 8.

```
font.Name = "Arial";
font.Angle = "ITALIC";
font.Weight = "BOLD";
font.Size = 8;
```

## See Also
`Stateflow.Chart`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**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:

```
table = find(sfroot,"-isa","Stateflow.StateTransitionTableChart", ...
    Path="myModel/State Transition Table");
```

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

**Name — Name of state transition table**
`"State Transition Table"` (default) | string scalar | character vector

Name of the state transition table, specified as a string scalar or 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`). 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**
string scalar | character vector

Name of the active state data object for the state transition table, specified as a string scalar or 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**
string scalar | character vector

Name of the enumerated data type for the active state data object for the state transition table, specified as a string scalar or 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) | string scalar | character vector

Sample time for activating the state transition table, specified as a string scalar or 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
string scalar | character vector

Default `fimath` object, specified as a string scalar or 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

### 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 — Debugger properties
`Stateflow.ChartDebug` object

Debugger properties for the state transition table, specified as a `Stateflow.ChartDebug` object with this property:

- **Breakpoints.OnEntry** — Whether to set the `On Chart Entry` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).

For more information, see "Set Breakpoints to Debug Charts".

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.

**2-159**

**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 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.

**StateFont — Font for state labels**
Stateflow.STTStateFont object

Font for the state labels in the chart that is automatically generated for the state transition table, specified as a Stateflow.STTStateFont object with these properties:

- **Name** — Font name, specified as a string scalar or character vector.
- **Angle** — Font angle, specified as "NORMAL" or "ITALIC".
- **Weight** — Font weight, specified as "NORMAL" or "BOLD".
- **Size** — Default font size for new states, specified as a scalar.

Example: table.StateFont.Name = "Arial";

Example: table.StateFont.Angle = "ITALIC";

Example: table.StateFont.Weight = "BOLD;

Example: table.StateFont.Size = 8;

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

Color for the 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.

**TransitionFont — Font for transition labels**
Stateflow.STTTransFont object

Font for the transition labels in the chart that is automatically generated for the state transition table, specified as a `Stateflow.STTTransFont` object with these properties:

- **Name** — Font name, specified as a string scalar or character vector.
- **Angle** — Font angle, specified as `"NORMAL"` or `"ITALIC"`.
- **Weight** — Font weight, specified as `"NORMAL"` or `"BOLD"`.
- **Size** — Default font size for new transitions, specified as a scalar.

Example: `table.TransitionFont.Name = "Arial";`

Example: `table.TransitionFont.Angle = "ITALIC";`

Example: `table.TransitionFont.Weight = "BOLD";`

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

**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.

**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) | string scalar | character vector

Description for the state transition table, specified as a string scalar or character vector.

**`Document` — Document link**
`""` (default) | string scalar | character vector

Document link for the state transition table, specified as a string scalar or 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

| | |
|---|---|
| find | Identify specified objects in hierarchy |
| getChildren | Identify children of object |
| dialog | Open properties dialog box |
| 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**
`sfnew` | `sfroot` | `add_block`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Model Finite State Machines by Using Stateflow Charts"
"State Transition Tables in Stateflow"
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced in R2012b**

# Stateflow.STTStateFont

Font for state labels in state transition tables

## Description

Use a `Stateflow.STTStateFont` object to specify the font properties for state labels in the chart that is automatically generated for a state transition table.

## Creation

Each state transition table has its own `Stateflow.STTStateFont` object. To access the `Stateflow.STTStateFont` object, use the `StateFont` property for the `Stateflow.StateTransitionTableChart` object.

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Name — Font name**
"Helvetica" (default) | string scalar | character vector

Font name, specified as a string scalar or character vector.

**Angle — Font angle**
"NORMAL" (default) | "ITALIC"

Font angle, specified as "NORMAL" or "ITALIC".

**Weight — Font weight**
"NORMAL" (default) | "BOLD"

Font weight, specified as "NORMAL" or "BOLD".

**Size — Default font size**
12 (default) | scalar

Default font size for new states in the state transition table, specified as a scalar.

### Examples

#### Change Font Properties for State Labels

Access the `Stateflow.STTStateFont` object for the `Stateflow.StateTransitionTableChart` object `stt`.

```
font = stt.StateFont;
```

Set the font for state labels to Arial. Set the font angle to italics and the font weight to bold. Set the default font size to 8.

```
font.Name = "Arial";
font.Angle = "ITALIC";
font.Weight = "BOLD";
font.Size = 8;
```

## See Also

`Stateflow.StateTransitionTableChart`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# Stateflow.STTTransFont

Font properties for transition labels in state transition tables

## Description

Use a `Stateflow.STTTransFont` object to specify the font properties for transition labels in the chart that is automatically generated for a state transition table.

## Creation

Each state transition table has its own `Stateflow.STTTransFont` object. To access the `Stateflow.STTTransFont` object, use the `TransitionFont` property for the `Stateflow.StateTransitionTableChart` object.

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Name — Font name**
"Helvetica" (default) | string scalar | character vector

Font name, specified as a string scalar or character vector.

**Angle — Font angle**
"NORMAL" (default) | "ITALIC"

Font angle, specified as "NORMAL" or "ITALIC".

**Weight — Font weight**
"NORMAL" (default) | "BOLD"

Font weight, specified as "NORMAL" or "BOLD".

**Size — Default font size**
12 (default) | scalar

Default font size for new transitions in the state transition table, specified as a scalar.

### Examples

#### Change Font Properties for Transition Labels

Access the `Stateflow.STTTransFont` object for the `Stateflow.StateTransitionTableChart` object `stt`.

```
font = stt.TransitionFont;
```

Set the font for transition labels to Arial. Set the font angle to italics and the font weight to bold. Set the default font size to 8.

```
font.Name = "Arial";
font.Angle = "ITALIC";
font.Weight = "BOLD";
font.Size = 8;
```

## See Also

`Stateflow.StateTransitionTableChart`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# Stateflow.TransBreakpoints

Breakpoint properties for transition

## Description

Use a `Stateflow.TransBreakpoints` object to specify the breakpoint properties for a transition. For more information, see "Set Breakpoints to Debug Charts".

## Creation

Each transition has its own `Stateflow.TransBreakpoints` object. To access the `Stateflow.TransBreakpoints` object, use the `Debug.Breakpoints` property of the `Stateflow.Transition` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**`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, specified as a numeric or logical 1 (`true`) or 0 (`false`).

**`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, specified as a numeric or logical 1 (`true`) or 0 (`false`).

## Examples

### Set Breakpoints for Transition

Access the `Stateflow.TransDebug` and `Stateflow.TransBreakpoints` objects for the `Stateflow.Transition` object `transition`.

```
debug = transition.Debug;
breakpoints = debug.Breakpoints;
```

Set the `When Transition is Tested` and `When Transition is Valid` breakpoints.

```
breakpoints.WhenTested = true;
breakpoints.WhenValid = true;
```

## See Also

`Stateflow.Transition`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34
"Set Breakpoints to Debug Charts"

**Introduced before R2006a**

# Stateflow.TransDebug

Debugger properties for transition

# Description

Use a `Stateflow.TransDebug` object to specify the debugger properties for a transition.

# Creation

Each transition has its own `Stateflow.TransDebug` object. To access the `Stateflow.TransDebug` object, use the `Debug` property for the `Stateflow.Transition` object.

## Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

### Breakpoints — Breakpoint properties
`Stateflow.TransBreakpoints` object

Breakpoint properties for the transition, specified as a `Stateflow.TransBreakpoints` object with these properties:

- **WhenTested** — Whether to set the `When Transition is Tested` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **WhenValid** — Whether to set the `When Transition is Valid` breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).

For more information, see "Set Breakpoints to Debug Charts".

## Examples

### Set Breakpoints for Transition

Access the `Stateflow.TransDebug` and `Stateflow.TransBreakpoints` objects for the `Stateflow.Transition` object `transition`.

```
debug = transition.Debug;
breakpoints = debug.Breakpoints;
```

Set the `When Transition is Tested` and `When Transition is Valid` breakpoints.

```
breakpoints.WhenTested = true;
breakpoints.WhenValid = true;
```

## See Also

Stateflow.Transition

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34
"Set Breakpoints to Debug Charts"

**Introduced before R2006a**

# Stateflow.TransFont

Font properties for transition labels

## Description

Use a `Stateflow.TransFont` object to specify the font properties for transition labels in a chart.

## Creation

Each chart has its own `Stateflow.TransFont` object. To access the `Stateflow.TransFont` object, use the `TransitionFont` property for the `Stateflow.Chart` object.

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Name — Font name**
"Helvetica" (default) | string scalar | character vector

Font name, specified as a string scalar or character vector.

**Angle — Font angle**
"NORMAL" (default) | "ITALIC"

Font angle, specified as `"NORMAL"` or `"ITALIC"`.

**Weight — Font weight**
"NORMAL" (default) | "BOLD"

Font weight, specified as `"NORMAL"` or `"BOLD"`.

**Size — Default font size**
12 (default) | scalar

Default font size for new transitions in the chart, specified as a scalar.

### Examples

#### Change Font Properties for Transition Labels

Access the `Stateflow.TransFont` object for the `Stateflow.Chart` object ch.

```
font = ch.TransitionFont;
```

Set the font for transition labels to Arial. Set the font angle to italics and the font weight to bold. Set the default font size to 8.

```
font.Name = "Arial";
font.Angle = "ITALIC";
font.Weight = "BOLD";
font.Size = 8;
```

## See Also
Stateflow.Chart

**Topics**
"Overview of the Stateflow API" on page 1-2
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# 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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

**LabelString — Label for transition**
"" (default) | string scalar | character vector

Label for the transition, specified as a string scalar or character vector. For more information, see "Specify Labels in States and Transitions Programmatically" on page 1-24.

### 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 1-24.

### 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 1-24.

### 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 1-24.

### 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 1-24.

### 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 "Comment Out Objects in a Stateflow 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 its source, its destination, or a state, box, or function that contains it.

**`CommentText` — Comment text**
`""` (default) | string scalar | character vector

Comment text added to the transition, specified as a string scalar or 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.AtomicSubchart` object | object | `Stateflow.Junction` object | `Stateflow.SimulinkBasedState` object | `Stateflow.State`

Source of the transition, specified as an empty array or a Stateflow API object of one of these types:

- `Stateflow.AtomicSubchart`
- `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.AtomicSubchart` object | object | `Stateflow.Junction` object | `Stateflow.SimulinkBasedState` object | `Stateflow.State`

Destination of the transition, specified as an empty array or a Stateflow API object of one of these types:

- `Stateflow.AtomicSubchart`
- `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 — Debugger properties
Stateflow.TransDebug object

Debugger properties for the transition, specified as a `Stateflow.TransDebug` object with these properties:

- **Breakpoints.WhenTested** — Whether to set the When Transition is Tested breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).
- **Breakpoints.WhenValid** — Whether to set the When Transition is Valid breakpoint, specified as a numeric or logical 1 (`true`) or 0 (`false`).

For more information, see "Set Breakpoints to Debug Charts".

Example: `transition.Debug.Breakpoints.WhenTested = true;`

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) | string scalar | character vector

Description for the transition, specified as a string scalar or character vector.

**`Document` — Document link**
`""` (default) | string scalar | character vector

Document link for the transition, specified as a string scalar or 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

getParent       Identify parent of object
dialog          Open properties dialog box
isCommented     Determine if graphical object is commented out
view            Display object in editing environment
highlight       Highlight graphical object
fitToView       Zoom in on graphical object

## 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**

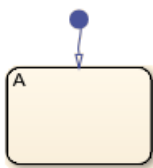Create a `Stateflow.Transition` object in the `Stateflow.Chart` object ch.

```
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`.
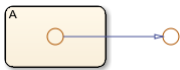
```
st.IsSubchart = false;
st.IsGrouped = false;
```

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.AtomicSubchart` | `Stateflow.Chart` | `Stateflow.Function` | `Stateflow.Junction` | `Stateflow.SimulinkBasedState` | `Stateflow.State`

**Topics**
"Overview of the Stateflow API" on page 1-2
"Transition Between Operating Modes"
"Summary of Stateflow API Objects and Properties" on page 1-34

**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

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

### Name — Name of truth table
"" (default) | string scalar | character vector

Name of the truth table, specified as a string scalar or character vector.

### LabelString — Label for truth table
"?" (default) | string scalar | character vector

Label for the truth table, specified as a string scalar or 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 "Comment Out Objects in a Stateflow 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 state, box, or function that contains it.

### CommentText — Comment text
"" (default) | string scalar | character vector

Comment text added to the truth table, specified as a string scalar or 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 truth table 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` — Debugger properties**
Stateflow.FunctionDebug object

Debugger properties for the truth table, specified as a Stateflow.FunctionDebug object with this property:

- **`Breakpoints.OnDuring`** — Whether to set the During Function Call breakpoint, 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". For more information, see "Set Breakpoints to Debug Charts".

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
string scalar | character vector

Default `fimath` object, specified as a string scalar or 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.

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 truth table in generated code.
- `"Function"` — The truth table is implemented as a separate C function.
- `"Inline"` — Calls to the truth table are replaced by code as long as the truth table is not part of a recursion.

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) | string scalar | character vector

Description for the truth table, specified as a string scalar or character vector.

**Document — Document link**
"" (default) | string scalar | character vector

Document link for the truth table, specified as a string scalar or 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

| | |
|---|---|
| find | Identify specified objects in hierarchy |
| getChildren | Identify children of object |
| getParent | Identify parent of object |
| dialog | Open properties dialog box |
| isCommented | Determine if graphical object is commented out |
| view | Display object in editing environment |
| highlight | Highlight graphical object |
| fitToView | Zoom in on graphical object |

## 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"
"Summary of Stateflow API Objects and Properties" on page 1-34

**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:

```
table = find(sfroot,"-isa","Stateflow.TruthTableChart", ...
    Path="myModel/Truth Table");
```

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Content**

**Name — Name of truth table**
`"Truth Table"` (default) | string scalar | character vector

Name of the truth table, specified as a string scalar or 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`). 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) | string scalar | character vector

Sample time for activating the truth table, specified as a string scalar or 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
string scalar | character vector

Default `fimath` object, specified as a string scalar or 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) | string scalar | character vector

Description for the truth table, specified as a string scalar or character vector.

**`Document` — Document link**
`""` (default) | string scalar | character vector

Document link for the truth table, specified as a string scalar or 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

| find | Identify specified objects in hierarchy |
| getChildren | Identify children of object |
| dialog | Open properties dialog box |
| 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**
sfnew | sfroot | add_block

**Topics**
"Overview of the Stateflow API" on page 1-2
"Model Finite State Machines by Using Stateflow Charts"
"Use Truth Tables to Model Combinatorial Logic"
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced before R2006a**

# Stateflow.Unit

Unit of measurement for input and output data

## Description

Use a `Stateflow.Unit` object to specify the unit of measurement for an input or output data object. For more information, see "Specify Units for Stateflow Data".

## Creation

Each data object and message has its own `Stateflow.Unit` object. However, the object only applies for `Stateflow.Data` objects when the `Scope` property is set to `"Input"` or `"Output"`. To access the `Stateflow.Unit` object, use the `Props.Unit` property for the `Stateflow.Data` object.

### Properties

Stateflow API objects have properties that correspond to the values you set in the Stateflow Editor. To access or modify a property, use dot notation. To access or modify multiple properties for multiple API objects, use the `get` and `set` functions, respectively. For more information, see "Modify Properties and Call Functions of Stateflow Objects" on page 1-11.

**Name — Name of unit of measurement**
`"inherit"` (default) | string scalar | character vector

Name of unit of measurement, specified as a string scalar or character vector. This property applies only to input and output data.

## Examples

### Specify Units for Data

Access the `Stateflow.Props` and `Stateflow.Unit` objects for the `Stateflow.Data` object x.

```
properties = x.Props;
unit = properties.Unit;
```

Specify the units as meters.

```
unit.Name = "m";
```

## See Also
`Stateflow.Data`

**Topics**

"Specify Units for Stateflow Data"

**Introduced before R2006a**

# API Object Function Reference

# copy

**Package:** `Stateflow`

Copy array of objects to clipboard
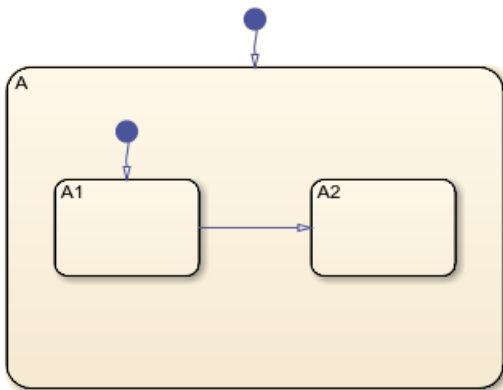
## Syntax

`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);
```

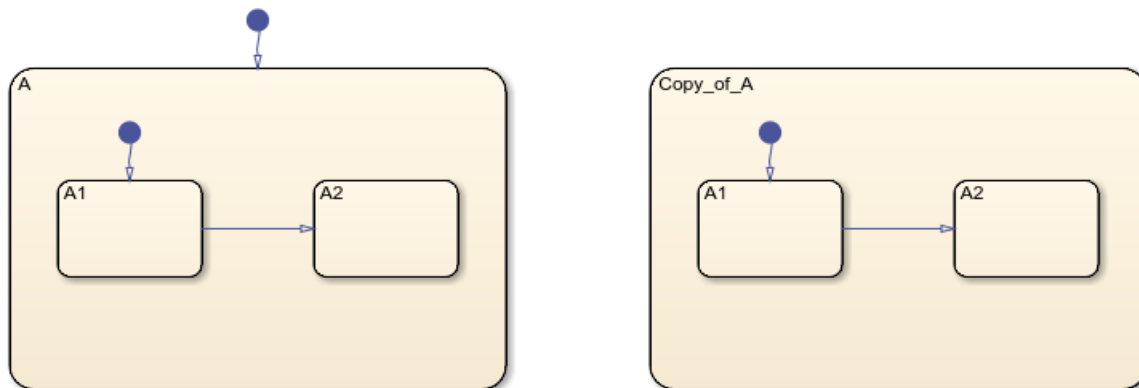**6** Restore the state properties to their original settings.

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

**7** Paste a copy of the objects from the clipboard to the chart.
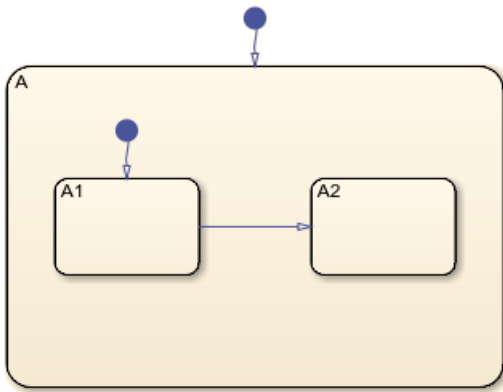
```
pasteTo(cb,ch);
```

**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;
   ```

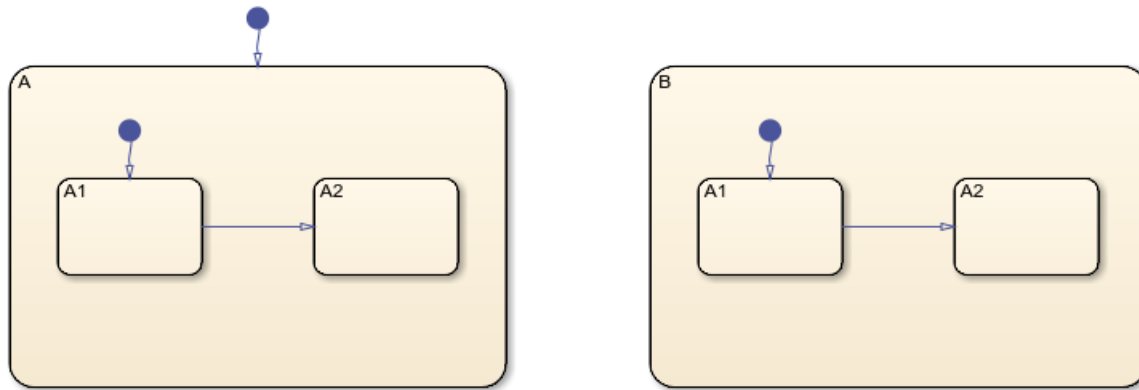5. Copy the objects in `objArray` and paste them in subchart B.

   ```
   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 3-2.
- 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 3-3.

## See Also

**Functions**
find | pasteTo | setdiff | sfclipboard

**Objects**
Stateflow.State | Stateflow.Clipboard

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

**Introduced before R2006a**

# defaultTransitions

**Package:** `Stateflow`

Identify default transitions in specified object

## Syntax

```
transitions = defaultTransitions(parent)
```

## Description

`transitions = defaultTransitions(parent)` returns an array of `Stateflow.Transition` objects that correspond to the default transitions at the top level of the specified parent object. For more information, see "Use Default Transitions to Specify Initial Substate Activity".

## Examples

**Identify Default Transitions**

Suppose that `ch` is the `Stateflow.Chart` object that corresponds to this chart.
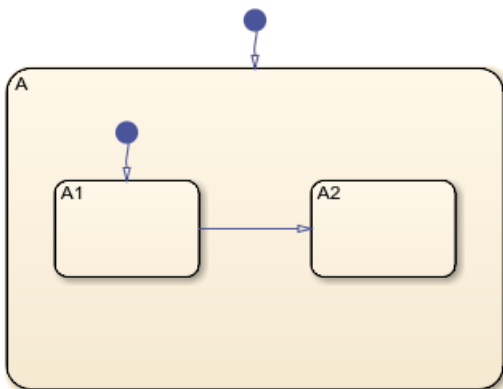


Identify the default transition at the top level of the chart. Display the name of the destination.

```
tr1 = defaultTransitions(ch);
tr1.Destination.Name
```

ans =

    'A'

Save the `Stateflow.State` object that corresponds to state A.

```
state = tr1.Destination;
```

Identify the default transition at the top level of the state A. Display the name of the destination.

```
tr2 = defaultTransitions(state);
tr2.Destination.Name

ans =

    'A1'
```

## Input Arguments

### `parent` — Parent object
`Stateflow.Chart` object | `Stateflow.Function` object | `Stateflow.State` object

Parent object, specified as a Stateflow API object of one of these types:

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

## Tips

- To identify default transitions inside a `Stateflow.Box`, call the `defaultTransitions` function on the `Stateflow.Chart`, `Stateflow.Function`, or `Stateflow.State` object that contains the box.

## See Also

**Functions**
find | getChildren | innerTransitions | outerTransitions | sinkedTransitions | sourcedTransitions

**Objects**
Stateflow.Box | Stateflow.Chart | Stateflow.Function | Stateflow.State | Stateflow.Transition

**Topics**
"Overview of the Stateflow API" on page 1-2
"Access Objects in Your Stateflow Chart" on page 1-6
"Use Default Transitions to Specify Initial Substate Activity"
"Group and Execute Transitions"

**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")
```

Find the chart named My Chart.

```
ch = find(sfroot,"-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.Port`
- `Stateflow.SimulinkBasedState`
- `Stateflow.SLFunction`
- `Stateflow.State`
- `Stateflow.StateTransitionTableChart`
- `Stateflow.Transition`
- `Stateflow.TruthTable`
- `Stateflow.TruthTableChart`

## See Also

`view` | `highlight` | `fitToView`

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

**Introduced before R2006a**

# find

**Package:** `Stateflow`

Identify 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` 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"));
ch = find(sfroot,"-isa","Stateflow.Chart","-and","-function",f);
```

## 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`
- `Simulink.BlockDiagram`
- `Stateflow.Box`
- `Stateflow.Chart`
- `Stateflow.EMChart`
- `Stateflow.EMFunction`
- `Stateflow.Function`
- `Stateflow.Machine`
- `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 "Summary of Stateflow API Objects and Properties" on page 1-34.

Example: `find(ch,"Name","A")` finds all objects in the chart `ch` whose `Name` property is A.

**`-isa` — Type of object**
character vector

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",class(object))` finds all objects of the same type 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`, `Simulink.BlockDiagram`, 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(sfroot,"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

`getChildren` | `getParent` | `strcmp`

**Topics**
"Access Objects in Your Stateflow Chart" on page 1-6
"Summary of Stateflow API Objects and Properties" on page 1-34
"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")`

Find the state named A.

`st = find(sfroot,"-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.Port`

- Stateflow.SimulinkBasedState
- Stateflow.SLFunction
- Stateflow.State
- Stateflow.Transition
- Stateflow.TruthTable

## See Also
view | highlight | zoomIn | zoomOut

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

**Introduced in R2008a**

# getChildren

**Package:** Stateflow

Identify children of object

## Syntax

```
objArray = getChildren(parent)
```

## Description

`objArray = getChildren(parent)` returns an array of objects that have the specified parent.

## Examples

### Identify Children of Chart

Suppose that `ch` is the `Stateflow.Chart` object that corresponds to this chart. This chart has two children, state `A` and a default transition. State `A` has four children, state `A1`, state `A2`, and two transitions.
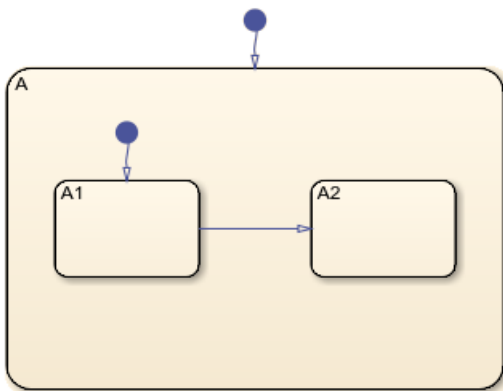


Identify the children of the chart `ch`. Display the object types of the children.

```
children = getChildren(ch);
arrayfun(@class,children,UniformOutput=false)
```

```
ans =

  2×1 cell array

    {'Stateflow.State'     }
    {'Stateflow.Transition'}
```

The first element in `children` is a state. Display the name of the state.

```
state = children(1);
state.Name

ans =

    'A'
```

Identify the children of state A. Display the object types of the children.

```
grandchildren = getChildren(state);
arrayfun(@class,grandchildren,UniformOutput=false)

ans =

  4×1 cell array

    {'Stateflow.State'     }
    {'Stateflow.State'     }
    {'Stateflow.Transition'}
    {'Stateflow.Transition'}
```

The first two elements in grandchildren are states. Display the names of the states.

```
grandchildren(1).Name

ans =

    'A1'
```

```
grandchildren(2).Name

ans =

    'A2'
```

## Input Arguments

**parent — Parent object**
Stateflow.Chart object | Stateflow.State object | Stateflow.Box object |
Stateflow.Function object | ...

Parent object, specified as a Stateflow API object of one of these types:

- Stateflow.Box
- Stateflow.Chart
- Stateflow.EMChart
- Stateflow.EMFunction
- Stateflow.Function
- Stateflow.SimulinkBasedState
- Stateflow.State
- Stateflow.SLFunction
- Stateflow.StateTransitionTableChart
- Stateflow.TruthTable

- `Stateflow.TruthTableChart`

## See Also

**Functions**
find | getParent | arrayfun | class

**Objects**
Stateflow.State | Stateflow.Box | Stateflow.Function

**Topics**
"Overview of the Stateflow API" on page 1-2
"Access Objects in Your Stateflow Chart" on page 1-6

**Introduced before R2006a**

# getParent

**Package:** `Stateflow`

Identify parent of object
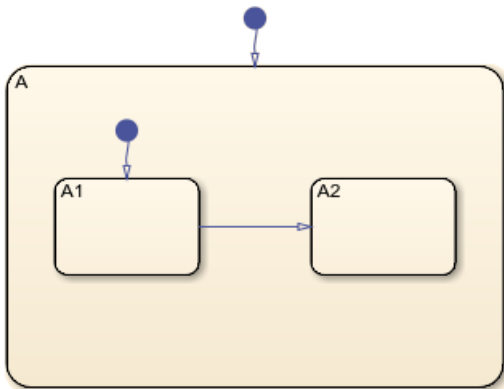
## Syntax

`parent = getParent(object)`

## Description

`parent = getParent(object)` returns the parent of an object in a Stateflow chart, State Transition Table, Truth Table, or MATLAB Function block.

## Examples

### Identify Parent of State

Suppose that `ch` is the `Stateflow.Chart` object that corresponds to this chart. In this chart, the parent of state `A1` is state A. The parent of state A is the chart.



Find the `Stateflow.State` object named `A1`.

`sA1 = find(ch,"-isa","Stateflow.State",Name="A1");`

Identify the parent of state `A1`. Display the name of the parent.

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

```
ans =

    'A'
```

Identify the parent of state A. Display the name of the parent.

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

ans =

    'Chart'
```

## Input Arguments

**object — Object**
Stateflow.State object | Stateflow.Box object | Stateflow.Function object | ...

Object in a Stateflow chart, State Transition Table, Truth Table, or MATLAB Function block, specified as a Stateflow API object of one of these types:

- Stateflow.Annotation
- Stateflow.AtomicBox
- Stateflow.AtomicSubchart
- Stateflow.Box
- Stateflow.Data
- Stateflow.EMFunction
- Stateflow.Event
- Stateflow.Function
- Stateflow.Junction
- Stateflow.Message
- Stateflow.Port
- Stateflow.SimulinkBasedState
- Stateflow.SLFunction
- Stateflow.State
- Stateflow.Transition
- Stateflow.TruthTable

## See Also

**Functions**
find | getChildren

**Objects**
Stateflow.State | Stateflow.Box | Stateflow.Function

**Topics**
"Overview of the Stateflow API" on page 1-2
"Access Objects in Your Stateflow Chart" on page 1-6

**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"`)`

Find the state named A.

`st = find(sfroot,`"-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.Port`
- `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

`view` | `fitToView` | `hilite_system` | `zoomIn` | `zoomOut`

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

**Introduced in R2012a**

# innerTransitions

**Package:** `Stateflow`

Identify inner transitions with specified source

## Syntax

```
transitions = innerTransitions(source)
```

## Description

`transitions = innerTransitions(source)` returns an array of `Stateflow.Transition` objects that correspond to the inner transitions of the specified source state. An inner transition is a transition that does not exit the source state. For more information, see "Control Chart Execution by Using Inner Transitions".

## Examples

### Identify Inner Transitions

Suppose that `ch` is the `Stateflow.Chart` object that corresponds to this chart.



Find the `Stateflow.State` object named A.

```
sA = find(ch,"-isa","Stateflow.State",Name="A");
```
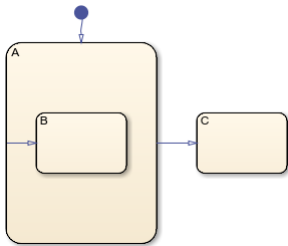
Identify the transition whose source is state A and whose destination is inside state A. Display the name of the destination.

```
tr = innerTransitions(sA);
tr.Destination.Name
```

```
ans =

    'B'
```

## Input Arguments

**source — Source state**
`Stateflow.State` object

Source state, specified as a `Stateflow.State` object.

## See Also

**Functions**
find | defaultTransitions | outerTransitions | sinkedTransitions | sourcedTransitions

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

**Topics**
"Overview of the Stateflow API" on page 1-2
"Access Objects in Your Stateflow Chart" on page 1-6
"Control Chart Execution by Using Inner Transitions"
"Group and Execute Transitions"

**Introduced before R2006a**

# isCommented

**Package:** Stateflow

Determine if graphical object is commented out

## Syntax

```
tf = isCommented(graphicalObject)
```

## Description

`tf = isCommented(graphicalObject)` returns a logical value that indicates if a graphical object is commented out. The function returns logical `1` (`true`) if:

- The graphical object is explicitly commented out. To explicitly comment out an object, set its `IsExplicitlyCommented` property to `true`. Alternatively, you can right-click the graphical object and select **Comment Out**.
- The graphical object is implicitly commented out. In this case, its `IsImplicitlyCommented` property has a value of `true`. An object is implicitly commented out when you explicitly comment out a state, box, or function that contains the object. Additionally,

  - Transitions are implicitly commented out when you comment out their source or destination.
  - Entry and exit ports are implicitly commented out when you comment out their matching entry or exit junction.

Otherwise, the function returns logical `0` (`false`).

## Examples

### Comment Out State

When you explicitly comment out a state, box, or function, you implicitly comment out all the graphical objects that it contains. For example, when you comment out state `A` in this chart, you also comment out its substates, `A1` and `A2`.
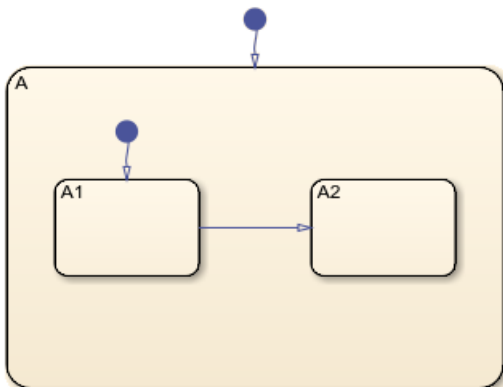
Find the `Stateflow.State` objects named A, A1, and A2.

```
sA = find(ch,"-isa","Stateflow.State",Name="A");
sA1 = find(ch,"-isa","Stateflow.State",Name="A1");
sA2 = find(ch,"-isa","Stateflow.State",Name="A2");
```

Check that state A and its substates are not commented out.

```
[isCommented(sA),isCommented(sA1),isCommented(sA2)]

ans =

  1×3 logical array

   0   0   0
```

Explicitly comment out state A.

```
sA.IsExplicitlyCommented = true;
```

Check that state A and its substates are commented out.

```
[isCommented(sA),isCommented(sA1),isCommented(sA2)]

ans =

  1×3 logical array

   1   1   1
```

## 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.Port`
- `Stateflow.SimulinkBasedState`
- `Stateflow.SLFunction`
- `Stateflow.State`
- `Stateflow.Transition`
- `Stateflow.TruthTable`

## See Also

**Functions**
find

**Objects**
Stateflow.State | Stateflow.Box | Stateflow.Function

**Topics**
"Overview of the Stateflow API" on page 1-2
"Comment Out Objects in a Stateflow Chart"
"Summary of Stateflow API Objects and Properties" on page 1-34

**Introduced in R2016a**

# outerTransitions

**Package:** `Stateflow`

Identify outer transitions with specified source

## Syntax

```
transitions = outerTransitions(source)
```

## Description

`transitions = outerTransitions(source)` returns an array of `Stateflow.Transition` objects that correspond to the outer transitions of the specified source state. An inner transition is a transition that does not exit the source state.

## Examples

### Identify Outer Transitions

Suppose that `ch` is the `Stateflow.Chart` object that corresponds to this chart.



Find the `Stateflow.State` object named A.

```
sA = find(ch,"-isa","Stateflow.State",Name="A");
```
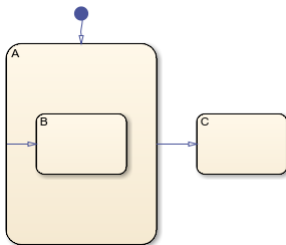
Identify the transition whose source is state A and whose destination is outside of state A. Display the name of the destination.

```
tr = outerTransitions(sA);
tr.Destination.Name
```

```
ans =

    'C'
```

## Input Arguments

**source — Source state**
`Stateflow.State` object

Source state, specified as a `Stateflow.State` object.

## See Also

**Functions**
find | defaultTransitions | innerTransitions | sinkedTransitions |
sourcedTransitions

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

**Topics**
"Overview of the Stateflow API" on page 1-2
"Access Objects in Your Stateflow Chart" on page 1-6
"Group and Execute Transitions"

**Introduced before R2006a**

# pasteTo

**Package:** `Stateflow`

Paste objects in clipboard to specified container object

## Syntax

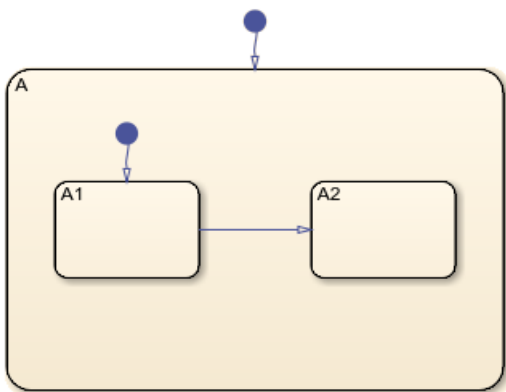`pasteTo(clipboard,parent)`

## 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.
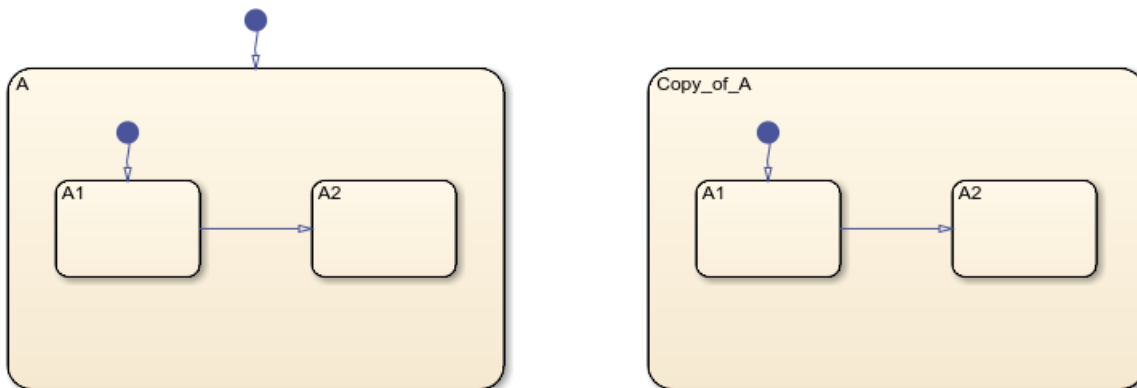
```
        sA.IsGrouped = prevGrouping;
        sA.Name = prevName;
```
**7** Paste a copy of the objects from the clipboard to the chart.
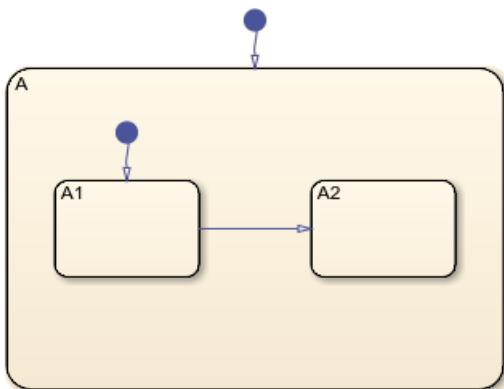
```
        pasteTo(cb,ch);
```
**8** Adjust the state properties of the new state.

```
        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;
```

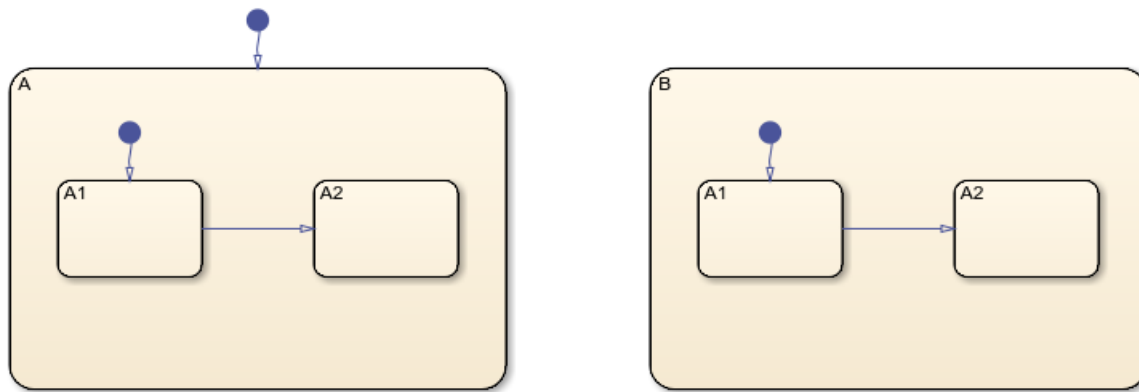5   Copy the objects in `objArray` and paste them in subchart B.

```
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 | setdiff | sfclipboard

**Objects**
Stateflow.State | Stateflow.Clipboard

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

**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**
string scalar | character array | `"clipboard"` | `""`

Source of the image, specified as a string scalar or 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

**Package:** `Stateflow`

Identify transitions with specified destination

## Syntax

```
transitions = sinkedTransitions(destination)
```
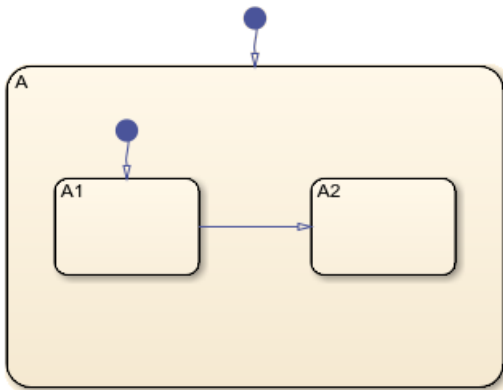
## Description

`transitions = sinkedTransitions(destination)` returns an array of `Stateflow.Transition` objects with the specified destination.

## Examples

### Identify Transitions With Specified Destination

Suppose that `ch` is the `Stateflow.Chart` object that corresponds to this chart.



Find the `Stateflow.State` object named A2.

```
sA2 = find(ch,"-isa","Stateflow.State",Name="A2");
```

Identify the transition whose destination is state A2. Display the name of the source.

```
tr = sinkedTransitions(sA2);
tr.Source.Name
```

```
ans =

    'A1'
```

## Input Arguments

**destination — Destination object**
Stateflow.Junction object | Stateflow.Port object | Stateflow.State object

Destination object, specified as a Stateflow API object of one of these types:

*   Stateflow.Junction
*   Stateflow.Port
*   Stateflow.State

## See Also

**Functions**
find | defaultTransitions | innerTransitions | outerTransitions | sourcedTransitions

**Objects**
Stateflow.Junction | Stateflow.Port | Stateflow.State | Stateflow.Transition

**Topics**
"Overview of the Stateflow API" on page 1-2
"Access Objects in Your Stateflow Chart" on page 1-6

**Introduced in R2012a**

# sourcedTransitions

**Package:** `Stateflow`

Identify transitions with specified source

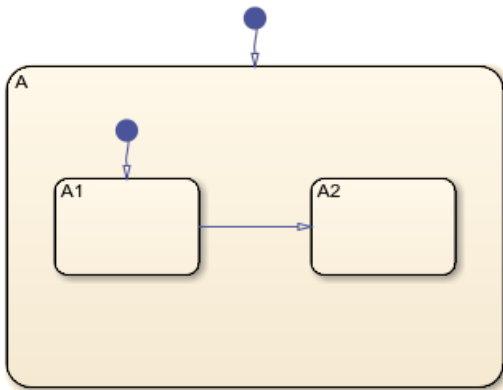## Syntax

```
transitions = sourcedTransitions(source)
```

## Description

`transitions = sourcedTransitions(source)` returns an array of `Stateflow.Transition` objects with the specified source.

## Examples

### Identify Transitions With Specified Source

Suppose that `ch` is the `Stateflow.Chart` object that corresponds to this chart.



Find the `Stateflow.State` object named `A1`.

```
sA1 = find(ch,"-isa","Stateflow.State",Name="A1");
```

Identify the transition whose source is state `A1`. Display the name of the destination.

```
tr = sinkedTransitions(sA1);
tr.Destination.Name
```

```
ans =

    'A2'
```

## Input Arguments

**source — Source object**
Stateflow.Junction object | Stateflow.Port object | Stateflow.State object

Source object, specified as a Stateflow API object of one of these types:

- Stateflow.Junction
- Stateflow.Port
- Stateflow.State

## See Also

**Functions**
find | defaultTransitions | innerTransitions | outerTransitions | sinkedTransitions

**Objects**
Stateflow.Junction | Stateflow.Port | Stateflow.State | Stateflow.Transition

**Topics**
"Overview of the Stateflow API" on page 1-2
"Access Objects in Your Stateflow Chart" on page 1-6

**Introduced before R2006a**

# up

**Package:** `Stateflow`

(Not recommended) Identify parent of object

---

**Note** Using up is not recommended. Use `getParent` instead.
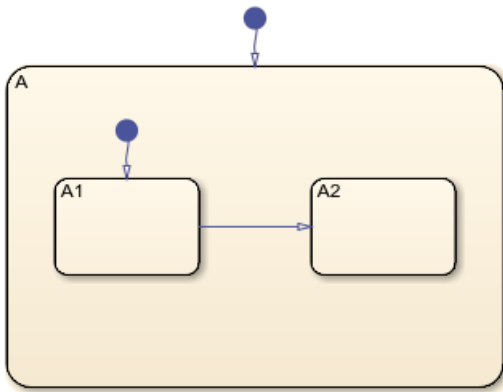
---

## Syntax

`parent = up(object)`

## Description

`parent = up(object)` returns the parent of an object in a Stateflow chart, State Transition Table, Truth Table, or MATLAB Function block.

## Examples

### Identify Parent of State

Suppose that `ch` is the `Stateflow.Chart` object that corresponds to this chart. In this chart, the parent of state `A1` is state A. The parent of state A is the chart.



Find the `Stateflow.State` object named `A1`.

`sA1 = find(ch,"-isa","Stateflow.State",Name="A1");`

Identify the parent of state `A1`. Display the name of the parent.

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

```
ans =

    'A'
```

Identify the parent of state A. Display the name of the parent.

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

```
ans =

    'Chart'
```

## Input Arguments

**`object` — Object**
`Stateflow.State` object | `Stateflow.Box` object | `Stateflow.Function` object | ...

Object in a Stateflow chart, State Transition Table, Truth Table, or MATLAB Function block, specified as a Stateflow API object of one of these types:

- `Stateflow.Annotation`
- `Stateflow.AtomicBox`
- `Stateflow.AtomicSubchart`
- `Stateflow.Box`
- `Stateflow.Data`
- `Stateflow.EMFunction`
- `Stateflow.Event`
- `Stateflow.Function`
- `Stateflow.Junction`
- `Stateflow.Message`
- `Stateflow.SimulinkBasedState`
- `Stateflow.SLFunction`
- `Stateflow.State`
- `Stateflow.Transition`
- `Stateflow.TruthTable`

## See Also

**Functions**
`find` | `getChildren` | `getParent`

**Objects**
`Stateflow.State` | `Stateflow.Box` | `Stateflow.Function`

**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.Junction`
  - `Stateflow.Port`
  - `Stateflow.State` with `IsSubchart` set to `false`
  - `Stateflow.Transition`

- The MATLAB Function 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")
```

Find the state named A.

```
st = find(sfroot,"-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.Port`
• `Stateflow.SimulinkBasedState`
• `Stateflow.SLFunction`
• `Stateflow.State`
• `Stateflow.StateTransitionTableChart`
• `Stateflow.Transition`
• `Stateflow.TruthTable`

- Stateflow.TruthTableChart

## See Also

highlight | fitToView | 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
view | highlight | fitToView | 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
`view` | `highlight` | `fitToView` | `zoomIn`

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

**Introduced before R2006a**