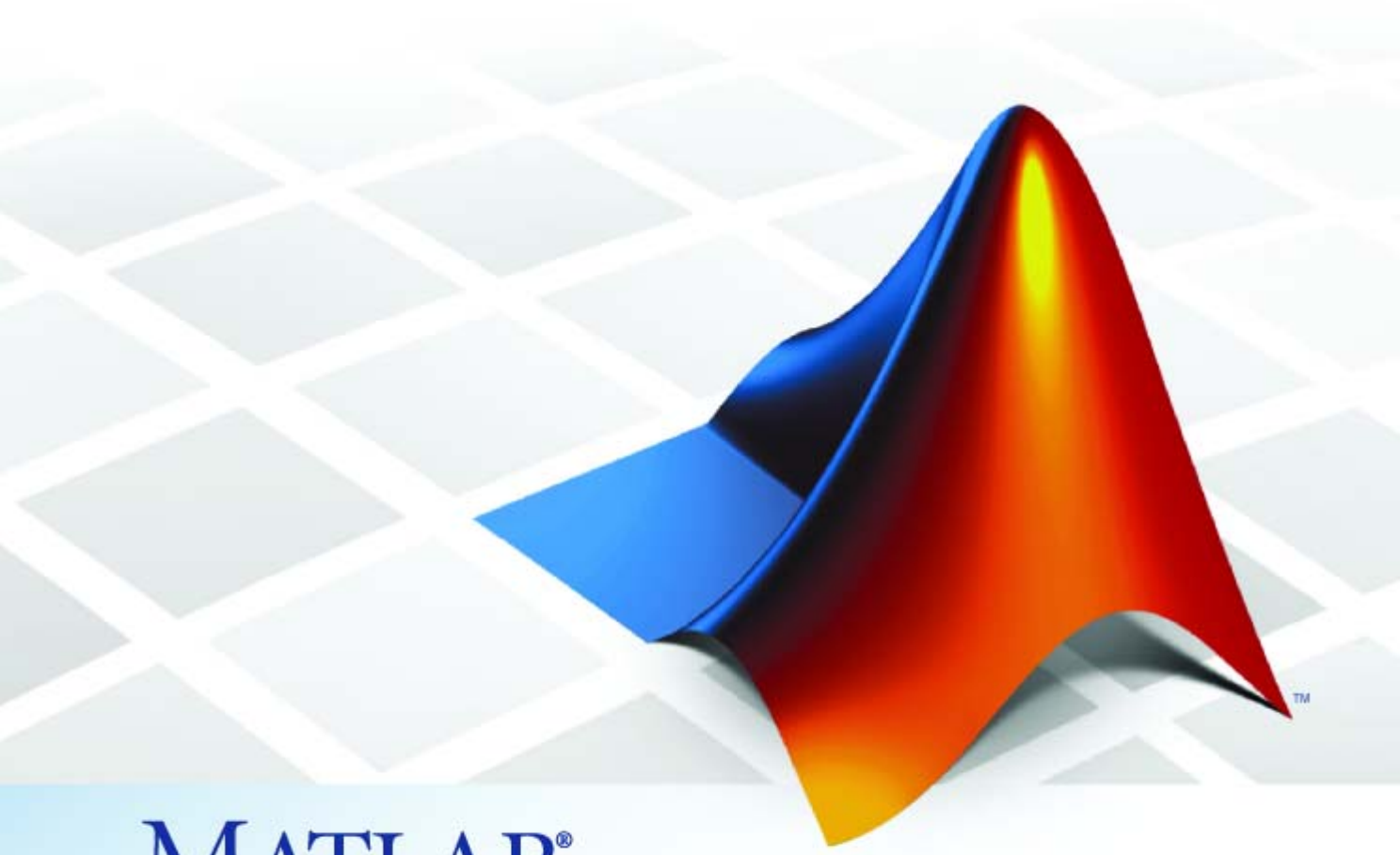
Stateflow® and Stateflow® Coder™ 7
API



MATLAB®
&SIMULINK®

The MathWorks™
Accelerating the pace of engineering and science

How to Contact The MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support

`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

For contact information about worldwide offices, see the MathWorks Web site.

Stateflow® and Stateflow® Coder™ API

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

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

Revision History

June 2004	Online only	Revised for Version 6.0 (Release 14)
October 2004	Online only	Revised for Version 6.1 (Release 14SP1)
March 2005	Online only	Revised for Version 6.2 (Release 14SP2)
September 2005	Online only	Revised for Version 6.3 (Release 14SP3)
March 2006	Online only	Revised for Version 6.4 (Release 2006a)
September 2006	Online only	Revised for Version 6.5 (Release 2006b)
September 2007	Online only	Rereleased for Version 7.0 (Release 2007b)
March 2008	Online only	Revised for Version 7.1 (Release 2008a)
October 2008	Online only	Revised for Version 7.2 (Release 2008b)
March 2009	Online only	Revised for Version 7.3 (Release 2009a)
September 2009	Online only	Revised for Version 7.4 (Release 2009b)
March 2010	Online only	Revised for Version 7.5 (Release 2010a)

Contents

1 Using the API

3 API Object Properties and Methods

4 API Method Reference

Index

1

Using the API

Overview of the Stateflow API

In this section...

What Is the Stateflow API?

The Stateflow® Application Programming Interface (API) is a tool you use to create or change Stateflow charts with MATLAB® commands. By placing Stateflow API commands in a MATLAB script, you can automate chart editing processes in a single command.

Applications for the Stateflow API include:

- Creating a script that performs common graphical edits and simplifies editing of Stateflow charts
- Creating a script that creates a repetitive "base" Stateflow chart
- Creating a script that produces a specialized report of your model

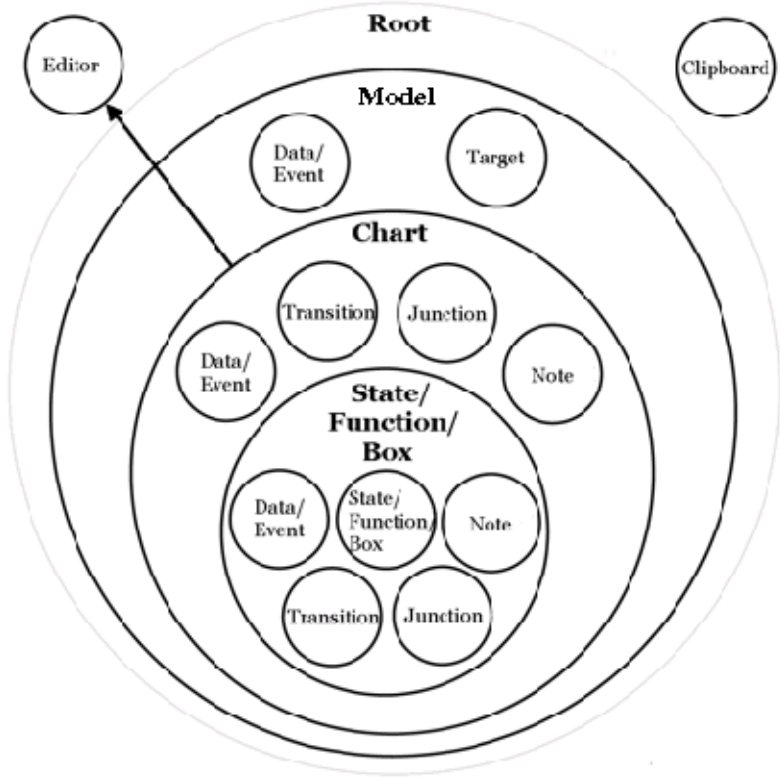
The Stateflow API consists of objects that represent actual Stateflow objects. For example, an API object of type State represents a Stateflow state, an API object of type Junction represents a Stateflow junction, and so on.

Each API object has properties and methods you use to perform editing operations on it. The correspondence between API object and Stateflow object is so close that what you do to a Stateflow API object affects the object it represents in the Stateflow Editor, and what you do to a graphical object in the Stateflow Editor affects the Stateflow API object that represents it.

Note You cannot undo any operation in the Stateflow Editor that you perform using the Stateflow API. If you perform an editing operation through the API, the undo and redo buttons are disabled from undoing and redoing any prior operations.

Stateflow API Object Hierarchy

Stateflow API objects represent actual Stateflow objects in a Stateflow chart. Like Stateflow objects, API objects contain or are contained by other Stateflow objects. For example, if state `A` contains state `B` in the Stateflow Editor, then the API object for state `A` contains the API object for state `B`. This diagram shows the Stateflow API hierarchy of objects:



Rules of containment define the Stateflow hierarchy and the Stateflow API object hierarchy. For example, charts can contain states but states cannot contain charts. The hierarchy of Stateflow objects appears in the section "Stateflow Hierarchy of Objects" in the Stateflow and Stateflow® Coder™ User's Guide. The Stateflow API hierarchy consists of these layers of containment:

- **Root** — The Root object (only one exists) is the parent of all Stateflow API objects. It is a placeholder at the top of the Stateflow API hierarchy to distinguish Stateflow objects from other objects, such as Simulink® model objects. You automatically create the Root object when you load a model containing a Stateflow chart or call the function `sfnew` to create a new model with a Stateflow chart.
- **Model** — Objects of type Model are available through the Stateflow Root object. Model objects are equivalent to Simulink models from a Stateflow chart perspective. Model objects can hold objects of type Chart, Data, Event, and Target.
- **Chart** — Within any Model object (model) there can be any number of chart objects. Within each object of type Chart, there can be objects of type State, Function, Box, Note, Data, Event, Transition, and Junction. These objects represent the components of a Stateflow chart.
- **State/Function/Box** — Nested within objects of type State, Function, and Box, there can be other objects of type State, Function, Box, Note, Junction, Transition, Data, and Event. Levels of nesting can continue indefinitely.

The preceding figure also shows two object types that exist outside the Stateflow containment hierarchy:

- **Editor** — Though not a part of the Stateflow containment hierarchy, an object of type Editor provides access to the purely graphical aspects of objects of type Chart. For each Chart object, there is an Editor object that provides API access to the Stateflow Editor.
- **Clipboard** — The Clipboard object has two methods, `copy` and `pasteTo`, that use the clipboard as a staging area to implement copy and paste functionality in the Stateflow API.

Getting a Handle on Stateflow API Objects

You manipulate Stateflow objects by manipulating the Stateflow API objects that represent them. You manipulate Stateflow API objects through a MATLAB variable called a *handle*.

The first handle you need in the Stateflow API is a handle to the Root object, which is the parent of all objects in the Stateflow API. In this command, the function `sfroot` returns a handle to the Root object:

```
rt = sfroot
```

Once you have a Root object handle, you can find a handle to the Model object for the Simulink model with which you want to work. Once you have a handle to a Model object, you can find a handle to a Chart object for the chart you want to edit. Later, when you create objects or find existing objects in a Stateflow chart, you receive a handle to the object that you can use to manipulate the actual object in the Stateflow Editor.

To learn how to use API object handles to create and edit Stateflow charts, see "Quick Start for the Stateflow API" on page 1-7.

What Are API Object Properties and Methods?

Once you obtain handles to Stateflow API objects, you can manipulate the Stateflow objects that they represent through the properties and methods that each Stateflow API object possesses. You access the properties and methods of an object through a handle to the object.

API Object Properties

API properties correspond to values that you normally set for an object through the user interface of the Stateflow Editor. For example, you can change the position of a transition by changing the Position property of the Transition object that represents the transition. In the Stateflow Editor, you can click-drag the source, end, or midpoint of a transition to change its position.

API Object Methods

API methods are similar to functions for creating, finding, changing, or deleting the objects they belong to. They provide services that are normally provided by the Stateflow Editor. For example, you can delete a transition in the Stateflow Editor by calling the `delete` method of the Transition object that represents the transition. Deleting a transition in the Stateflow Editor is normally done by selecting a transition and pressing the **Delete** key.

Common API Properties and Methods

Stateflow API objects have some common properties and methods. For example, all API objects have an `Id` and a `Description` property. All API objects have a `get` and a `set` method for viewing or changing the properties of an object, respectively. Most API objects also have a delete method. Methods held in common among all Stateflow objects are listed in the reference section "All Object Methods" on page 2-5.

Unique API Properties and Methods

Each API object also has properties and methods unique to its type. For example, a State object has a `Position` property containing the spatial coordinates for the state it represents in the Stateflow Editor. A Data object, however, has no `Position` property.

Quick Start for the Stateflow API

In this section...

Creating a New Model and Chart

Create a new model by following these steps:

1. Close down all Simulink models.
2. Use the function `sfnew` to create a new chart.

   The `sfnew` function creates a new untitled Simulink model with a new Stateflow chart in it. Do not open the Stateflow chart.

You now have only one Simulink model in memory. You are now ready to access the API Model object that represents the model itself.

Accessing the Model Object

In the Stateflow API, each model you create or load into memory is represented by an object of type Model. Before accessing the Stateflow chart you created in the previous section, you must first connect to its Model object. However, in the Stateflow API, all Model objects are contained by the Stateflow API Root object, so you must use the Root object returned by the function `sfroot` to access a Model object:

1. Use this command to obtain a handle to the Root object:

   ```
   rt = sfroot
   ```

2. Use the handle to the Root object, `rt`, to find the Model object representing your new untitled Simulink model and assign it a handle `m` in this command:

```
m = rt.find('-isa','Simulink.BlockDiagram')
```

If, instead of one model, there are several models open, this command returns an array of different Model objects that you can access through indexing (`m(1)`,`m(2)`,...). You can identify a specific Model object using the properties of each model, particularly the `Name` property, which is the name of the model. For example, you can use the `Name` property to find a Model object named **myModel** with this command:

```
m = rt.find('-isa', 'Simulink.BlockDiagram', '-and',
'Name','myModel')
```

However, since you now have only one model loaded, the object handle `m` in the command for step 2 returns the Model object for the model that you just created. You are now ready to use `m` to access the empty Stateflow chart so that you can start filling it with Stateflow objects.

Accessing the Chart Object

In "Accessing the Model Object" on page 1-7, you accessed the Model object containing your new chart to return a handle to the Model object for your new model, `m`. Perform these steps to access the new Stateflow chart:

1. Access the new Chart object and assign it to the workspace variable `chart` as follows:

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

   In the preceding command, the `find` method of the Model object `m` returns an array of all charts belonging to that model. Because you created only one chart, the result of this command is the chart you created. If you created several charts, the `find` method returns an array of charts that you could access through indexing (for example, `chart(1)`, `chart(2)`, and so on).

   You can also use standard function notation instead of dot notation for the preceding command. In this case, the first argument is the Model object handle, `m`.

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

2. Open the Stateflow chart with this API command:

```
chart.view
```

The preceding command calls the `view` method of the Chart object whose handle is `chart`. The specified chart appears in the Stateflow Editor. You should now have an empty Stateflow chart in front of you. Other Stateflow API objects have `view` methods as well.

Creating New Objects in the Chart

In the previous section, you created a handle to the new Chart object, `chart`. Continue by creating new objects for your chart using these steps:

1. Create a new state in the Chart object `chart` with this command:

   ```
   sA = Stateflow.State(chart)
   ```

   This command is a Stateflow API constructor for a new state in which `Stateflow.State` is the object type for a state, `chart` is a workspace variable containing a handle to the parent chart of the new state, and `sA` is a workspace variable to receive the returned handle to the new state.

   An empty state now appears in the upper left-hand corner of the Stateflow Editor.

2. Use the `chart.view` command to bring the Stateflow Editor to the foreground for viewing.

3. Assign a name and position to the new state by assigning values to the properties of the new State object as follows:

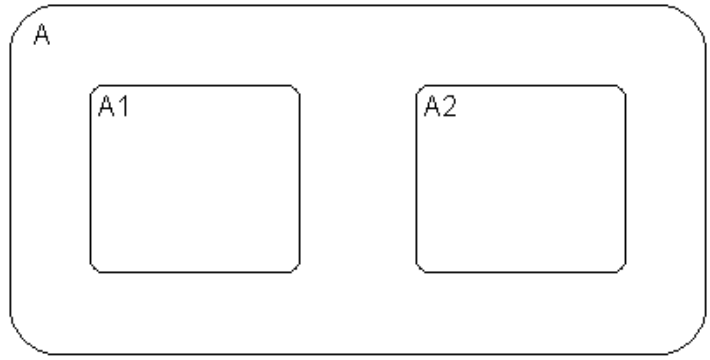
   ```
   sA.Name = 'A'
   sA.Position = [50 50 310 200]
   ```

4. Create new states A1 and A2 inside state A and assign them properties with these commands:

   ```
   sA1 = Stateflow.State(chart)
   sA1.Name = 'A1'
   sA1.Position = [80 120 90 60]
   sA2 = Stateflow.State(chart)
   sA2.Name = 'A2'
   ```

```
sA2.Position = [240 120 90 60]
```

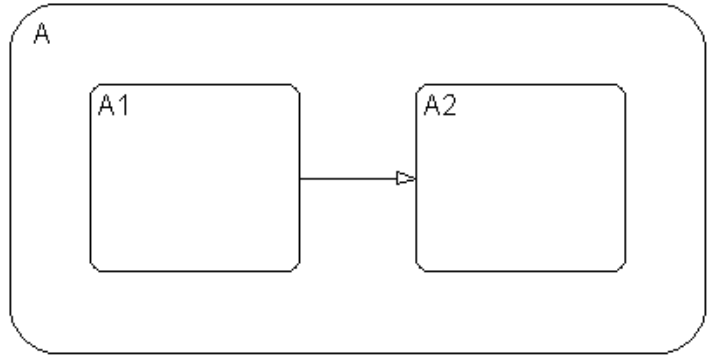
These commands create and use the workspace variables `sA`, `sA1`, and `sA2` as handles to the new states, which now appear as follows.



5 Create a transition from the 3 o'clock position (right side) of state A1 to the 9 o'clock position (left side) of state A2 with these commands:

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

A transition now appears as shown.



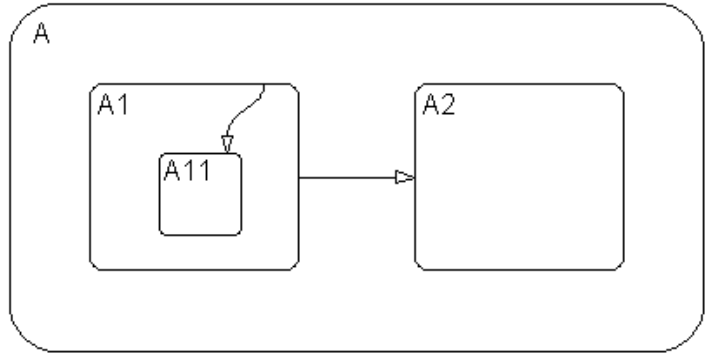
6 Draw, name, and position a new state A11 inside A1 with these commands:

```
sA11 = Stateflow.State(chart)
sA11.Name = 'A11'
sA11.Position = [90 130 35 35]
```

7 Draw an inner transition from the 1 o'clock position of state A1 to the 1 o'clock position of state A11 with these commands:

```
tA1A11 = Stateflow.Transition(chart)
tA1A11.Source = sA1
tA1A11.Destination = sA11
tA1A11.SourceOClock = 1.
tA1A11.DestinationOClock = 1.
```

Your Stateflow chart now appears as shown.



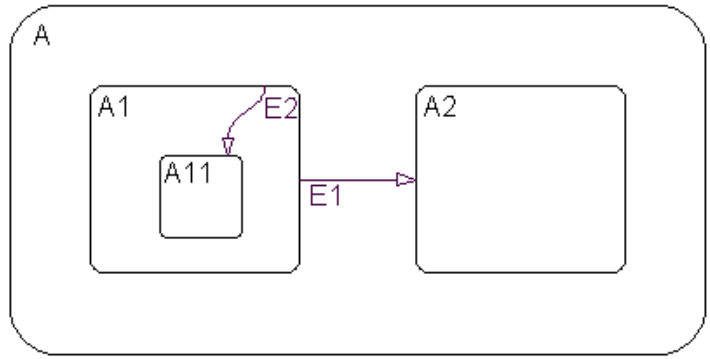
8 Add the label `E1` to the transition from state `A1` to state `A2` with this command:

```
tA1A2.LabelString = 'E1'
```

9 Add the label `E2` to the transition from state `A1` to state `A11` with this command:

```
tA1A11.LabelString = 'E2'
```

The Stateflow chart now appears as shown.



Both the state and transition labels in our example are simple one-line labels. To enter more complex multiline labels, see "Entering Multiline Labels" on page 1-38. Labels for transitions also have a `LabelPosition` property you can use to move the labels to better locations.

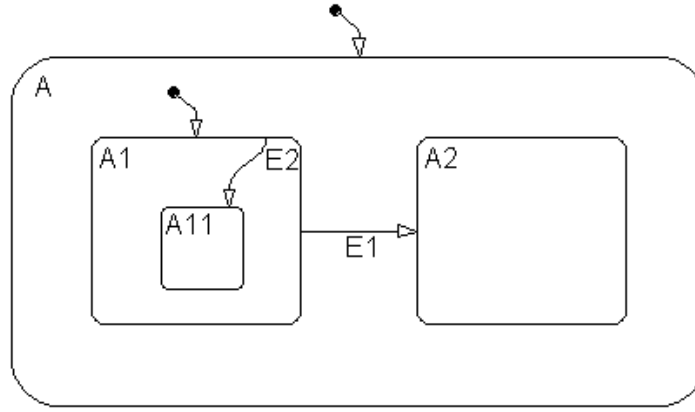
10 Use these commands to move the label for the transition from A1 to A2 to the right by 15 pixels:

```
pos = tA1A2.LabelPosition
pos(1) = pos(1)+15
tA1A2.LabelPosition = pos
```

11 Use these commands to finish your new chart by adding default transitions to states A and A1 with source points 20 pixels above and 10 pixels to the left of the top midpoint of each state:

```
dtA = Stateflow.Transition(chart)
dtA.Destination = sA
dtA.DestinationOClock = 0
xsource = sA.Position(1)+sA.Position(3)/2-10
ysource = sA.Position(2)-20
dtA.SourceEndPoint = [xsource ysource]
dtA1 = Stateflow.Transition(chart)
dtA1.Destination = sA1
dtA1.DestinationOClock = 0
xsource = sA1.Position(1)+sA1.Position(3)/2-10
ysource = sA1.Position(2)-20
dtA1.SourceEndPoint = [xsource ysource]
```

You now have this complete Stateflow chart.



12 Save the Simulink model with its new Stateflow chart to the current folder as `myModel.mdl`:

```
sfsave(m.Name, 'myModel')
```

This command uses the `Name` property of the Model object `m` for saving the model under a new name.

You are now finished with "Quick Start for the Stateflow API" on page 1-7. You can continue with "Accessing the Properties and Methods of Objects" on page 1-14, or you can go to "Creating a MATLAB Script of API Commands" on page 1-42 to see how to create a script of the API commands you used in this Quick Start section.

Accessing the Properties and Methods of Objects

In this section...

Naming Conventions for Properties and Methods

By convention, all properties begin with a capital letter, for example, the property `Name`. However, if a property consists of concatenated words, the words following the first word are capitalized, for example, the property `LabelString`. The same naming convention applies to methods, with the exception that a method name must begin with a letter in lowercase; for example, the method `find`.

Using Dot Notation with Properties and Methods

You can access the properties and methods of an object by adding a period (`.`) and the name of the property or method to the end of an object's handle variable. For example, this command returns the Type property for a State object represented by the handle `s`:

```
stype = s.Type
```

This command calls the `dialog` method of the State object `s` to open a properties dialog box for that state:

```
s.dialog
```

Nesting Dot Notation

You can nest smaller dot expressions in larger dot expressions of properties. For example, the `Chart` property of a State object returns the Chart object of the containing chart. Therefore, the expression `s.Chart.Name` returns the name of the chart containing the State whose object is `s`.

Methods can also be nested in dot expressions. For example, if the State object `sA1` represents state A1 in the final Stateflow chart at the end of "Creating New Objects in the Chart" on page 1-9, this command returns the string label for state A1's inner transition to its state A11.

```
label = sA1.innerTransitionsOf.LabelString
```

The preceding command uses the `LabelString` property of a Transition object and the `innerTransitions` method for a State object. It works as shown only because state A1 has one inner transition. If state A1 has more than one transition, you must first find all the inner transitions and then use an array index to access each one, as shown below:

```
innerTransitions = sA1.innerTransitionsOf
label1 = innerTransitions(1).LabelString
label2 = innerTransitions(2).LabelString
and so on...
```

Using Function Notation with Methods

As an alternative to dot notation, you can access object methods with standard function call notation. For example, you can use the `get` method to access the Name property of a Chart object, `ch`, through one of these commands:

```
name = ch.get('Name')
name = get(ch,'Name')
```

If you have array arguments to methods you call, use function notation. This example returns a vector of strings with the names of each chart in the array of Chart objects `chartArray`:

```
names = get(chartArray, 'Name')
```

If, instead, you attempt to use the `get` command with this dot notation, an error results:

```
names = chartArray.get('Name')
```

Displaying Properties and Methods

In this section...

Displaying Properties

To access the names of all properties for any particular object, use the `get` method. For example, if the object `s` is a State object, enter this command to list the properties and current values for any State object:

```
get(s)
```

To get a quick description for each property, use the `help` method. For example, if `s` is a State object, this command returns a list of State object properties, each with a small accompanying description:

```
s.help
```

Note Some properties do not have a description, because their names are considered descriptive enough.

Displaying the Names of Methods

Use the `methods` method to list the methods for any object. For example, if the object `t` is a handle to a Transition object, use this command to list the methods for any Transition object:

```
t.methods
```

Note These internal methods may be displayed by the `methods` method for an object, but you cannot use them and they are not documented: `areChildrenOrdered`, `getChildren`, `getDialogInterface`, `getDialogSchema`, `getDisplayClass`, `getDisplayIcon`, `getDisplayLabel`, `getFullName`, `getHierarchicalChildren`, `getPreferredProperties`, `isHierarchical`, `isLibrary`, `isLinked`, `isMasked`.

Use a combination of the `get` method and the `classhandle` method to list only the names of the methods for an object. For example, list the names of the methods for the Transition object `t` with this command:

```
get(t.classhandle.Methods, 'Name')
```

Displaying Property Subproperties

Some properties are objects that have properties referred to as subproperties. For example, when you invoke the command `get(ch)` on a chart object, `ch`, the output displays the following for the `StateFont` property:

```
StateFont: [1x1 Stateflow.StateFont]
```

This value indicates that the `StateFont` property of a state has subproperties. To view the subproperties of `StateFont`, enter the command `get(ch.StateFont)` to see something like this:

```
Name: 'Helvetica'
Size: 12
Weight: 'NORMAL'
Angle: 'NORMAL'
```

This list shows that `Name`, `Size`, `Weight`, and `Angle` are subproperties of the property `StateFont`. In the API reference pages for this guide (see Chapter 2, “API Object Reference”), these properties are listed by their full names: `StateFont.Name`, `StateFont.Size`, and so on.

Displaying Enumerated Values for Properties

Many API object properties can be set only to one of a group of enumerated strings. You can identify these properties from the API reference pages (see

Chapter 2, “API Object Reference”). When you use the `get` method to access object properties (see “Displaying Properties” on page 1-16) the values for these properties appear as strings of capital letters.

You can use the `set` method to display a list of acceptable strings for a property requiring enumerated values. For example, if `ch` is a handle to a Chart object, you can display the allowed enumerated values for the `Decomposition` property of that chart with this command:

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

Creating and Destroying API Objects

In this section...

About Creating and Destroying API Objects

You create (construct), parent (contain), and delete (destroy) objects in Stateflow charts through constructor methods in the Stateflow API. For all but the Editor and Clipboard objects, creating objects establishes a handle to them that you can use for accessing their properties and methods to make modifications to Stateflow charts.

Stateflow objects are contained (parented) by other objects as defined in the Stateflow hierarchy of objects (see "Stateflow API Object Hierarchy" on page 1-3). You control containment of nongraphical objects in the Model Explorer.

Creating Stateflow Objects

You create a Stateflow object as the child of a parent object through API constructor methods. Each Stateflow object type has its own constructor method. See "Constructor Methods" on page 2-21 for a list of the valid constructor methods.

Use this process to create Stateflow objects with the Stateflow API:

1. Access the parent object to obtain a handle to it.

   When you first begin populating a model or chart, this means that you must get a handle to the Stateflow Model object or a particular Chart object. See "Accessing the Model Object" on page 1-7 and "Accessing the Chart Object" on page 1-8.

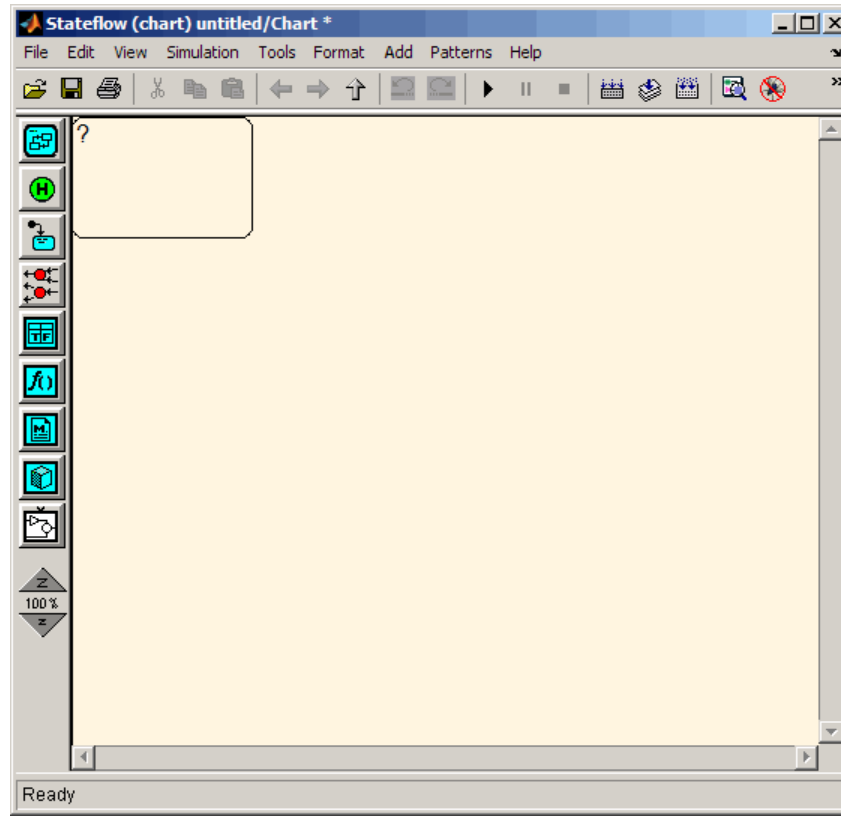
See also “Accessing Existing Stateflow Objects” on page 1-23 for a more general means of accessing (getting an object handle to) an existing Stateflow object.

2 Call the appropriate constructor method for the creation of the object using the parent (containing) object as an argument.

For example, this command creates and returns a handle `s` to a new state object in the chart object with the handle `ch`:

```
s = Stateflow.State(ch)
```

By default, the newly created state from the preceding command appears in the upper left corner of the Stateflow chart (at *x*-*y* coordinates 0,0).



The constructor returns a handle to an API object for the newly created Stateflow object. Use this handle to display or change the object through its properties and methods.

3 Use the object handle returned by the constructor to make changes to the object in the Stateflow chart.

For example, you can now use the handle `s` to set its name (`Name` property) and position (`Position` property). You can also connect it to other states or junctions by creating a Transition object and setting its `Source` or `Destination` property to `s`. See "Creating New Objects in the Chart" on page 1-9 for examples.

Use the preceding process to create all Stateflow objects in your chart. "Creating New Objects in the Chart" on page 1-9 gives examples for creating states and transitions. Objects of other types are created just as easily. For example, this command creates and returns a handle (`d1`) for a new Data object belonging to the state A (handle `sA`):

```
d1 = Stateflow.Data(sA)
```

Note Currently, there is no constructor for a Stateflow chart. To create a chart with the Stateflow API you must use the `sfnew` function.

Establishing the Parent (Container) of an Object

As discussed in the previous section, "Creating Stateflow Objects" on page 1-19, the Stateflow API constructor establishes the parent for a newly created object by taking a handle for the parent object as an argument to the constructor.

Graphical Object Parentage

When graphical objects (states, boxes, notes, functions, transitions, junctions) are created, they appear completely inside their containing parent object. In the Stateflow Editor, graphical containment is a necessary and sufficient condition for establishing the containing parent.

Repositioning a graphical object through its `Position` property can change an object's parent or cause an undefined parent error condition. Parsing a chart in which the edges of one object overlap with another produces an undefined parent error condition that cannot be resolved by the Stateflow parser. You can check for this condition by examining the value of the `BadIntersection` property of a Chart object, which equals 1 if the edges of a graphical object overlap with other objects. You need to set the size and position of objects so that they are clearly positioned and separate from other objects.

Nongraphical Object Parentage

When nongraphical objects (data, events, and targets) are created, they appear in the Model Explorer at the hierarchical level of their owning object. Containment for nongraphical objects is established through the Model Explorer only. See "Using the Model Explorer with Stateflow Objects" in the Stateflow and Stateflow Coder User's Guide.

Destroying Stateflow Objects

Each Stateflow object of type State, Box, Function, Note, Transition, Junction, Event, Data, or Target has a destructor method named `delete`. In this example, a State object, `s`, is deleted:

```
s.delete
```

The preceding command is equivalent to performing a mouse select and keyboard delete operation in the Stateflow Editor. Upon deletion, graphical Stateflow objects are sent to the clipboard; nongraphical objects, such as data and events, are completely deleted. The workspace variable `s` still exists but is no longer a handle to the deleted state.

Accessing Existing Stateflow Objects

In this section...

About Stateflow Object Handles

Creating Stateflow objects through the Stateflow API gives you an immediate handle to the newly created objects (see "Creating Stateflow Objects" on page 1-19). You can also connect to Stateflow objects that already exist for which you have no current API handle.

Finding Objects and Properties

There are several object methods that you use to traverse the Stateflow hierarchy to locate existing objects. Chief among these is the versatile `find` method.

With the `find` method, you specify what to search for by specifying combinations of these types of information:

- The type of object to find
- A property name for the object to find and its value

This example searches through Model object `m` to return every State object with the name `'On'`.

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

If a `find` command finds more than one object that meets its specifications, it returns an array of qualifying objects. This example returns an array of all charts in your model:

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

Use array indexing to access individual properties and methods for a chart. For example, if the preceding command returns three Stateflow charts, this command returns the Name property of the second chart found:

```
name2 = chartArray(2).Name
```

Tip To access the property of a linked Stateflow object, do one of the following:

1. Open the library model explicitly.
2. View a linked subsystem or block in the main model.
3. Compile or simulate the model.

Doing one of those steps loads a library model into the Simulink workspace. Just opening a main model that refers to a linked Stateflow chart does not guarantee that the Stateflow API can find a linked chart.

By default, the `find` command finds objects at all depths of containment within an object. This includes the zeroth level of containment, which is the searched object itself. For example, suppose that state `A`, which corresponds to State object `sA`, contains two states, `A1` and `A2`. Use a `find` command that finds all the states in `A`:

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

The preceding command finds three states: `A`, `A1`, and `A2`.

Note Be careful when specifying the objects you want to find with the `find` method for a Root or Model object. Using the `find` method for these objects can return Simulink objects matching the arguments you specify. For example, if `rt` is a handle to the Root object, the command `find('Name', 'ABC')` might return a Simulink subsystem or block named `ABC`. See the reference for the `find` method for a full description of the method and its parameters.

Finding Objects at Different Levels of Containment

Once you find a particular object in a Stateflow chart by its name or another property, you might want to find the objects that it contains (children), or the object that contains it (parent). To find child objects, use the `find` method. To find a parent object, use the method `up`.

Finding Child Objects

The `find` method finds objects at the depth of containment within an object that you specify. If you want to limit the containment search depth with the `find` command, use the `depth` switch. For example, to find all the objects in State object `sA` at the first level of containment, use this command:

```
objArray = sA.find('-depth', 1)
```

Don't forget, however, that the `find` command always includes the zeroth level of containment, which is the object itself. So, the preceding command also includes state A in the list of objects found. However, you can exclude state A from the vector of objects in `objArray` with the MATLAB function `setdiff` as follows:

```
objArray = setdiff(objArray, sA)
```

This command returns a collection of all junctions at the first level of containment inside the state A that is represented by State object `sA`:

```
juncArray = sA.find('-isa','Stateflow.Junction','-depth',1)
```

This command returns an array of all transitions inside state A at all levels of containment:

```
transArray = sA.find('-isa','Stateflow.Transition')
```

Finding a Parent Object

The `up` method finds the parent container object of any given object. In the example Stateflow chart in "Creating New Objects in the Chart" on page 1-9, state A contains states A1 and A2. Also, state A1 contains state A11. In the example, sA11 is a handle to the state A11. This means that

```
>> pA11 = sA11.up;
>> pA11.Name

ans =

A1
```

returns a handle `pA11` to the state A1, the parent of state A11, and

```
>> ppA11 = pA11.up;
>> ppA11.Name

ans =

A
```

returns a handle `ppA11` to the state A, the parent of state A1.

Retrieving Recently Selected Objects

You can retrieve the most recently selected objects in a Stateflow chart by using the `sfgco` function. This function returns object handles or a vector of handles depending on these conditions:

If...	Then `sfgco` returns...
There are no open charts	An empty matrix
There is no selection list	Handle of the chart most recently clicked
You select one object in a chart	Handle to the selected object
You select multiple objects in a chart	Vector of handles for the selected objects
You select objects in multiple charts	Handles of the most recently selected objects in the most recently selected chart

For example, suppose you run the `sf_boiler` demo and open the Stateflow chart called `Bang-Bang Controller`. If you select the `Off` state in the chart, `sfgco` returns:

```
ans =

               Path: 'sf_boiler/Bang-Bang Controller/Heater'
                 Id: 20
            Machine: [1x1 Stateflow.Machine]
               Name: 'Off'
        Description: ''
        LabelString: [1x27 char]
           FontSize: 12
          ArrowSize: 8
          TestPoint: 0
              Chart: [1x1 Stateflow.Chart]
    BadIntersection: 0
          Subviewer: [1x1 Stateflow.Chart]
           Document: ''
                Tag: []
    RequirementInfo: ''
     ExecutionOrder: 0
      HasOutputData: 0
           Position: [31.7440 40.9730 214.1807 88.1000]
      Decomposition: 'EXCLUSIVE_OR'
               Type: 'OR'
         IsSubchart: 0
          IsGrouped: 1
              Debug: [1x1 Stateflow.StateDebug]
```

Getting and Setting the Properties of Objects

Once you obtain a particular object, you can access its properties directly or through the `get` method. For example, you obtain the description for a State object `s` with one of these commands:

- `od = s.Description`
- `od = s.get('Description')`
- `od = get(s, 'Description')`

You change the properties of an object directly or through the `set` method. For example, you change the description of the State object `s` with one of these commands:

- `s.Description = 'This is the On state.'`
- `s.set('Description', 'This is the On state.')`
- `set(s, 'Description', 'This is the On state.')`

Moving Graphical Objects

In this section...

How to Move Objects Programmatically

To move a graphical object programmatically, choose one of these techniques:

Technique	Example
Change the `Position` property of the object directly.	`object.Position = [40 40 100 60])`
Use the `set` method to change the `Position` property of the object.	`object.set('Position', [40 40 100 60])`
	`set(object, 'Position', [40 40 100 60])`

In each 1-by-4 array, the first two values are the (x,y) coordinates of the upper left corner of the object. The last two values are the width and height, respectively.

Note These programmatic techniques work only for objects that have the `Position` property.

Example of Moving Subcharted States

You can adjust the locations of two subcharted states as follows:

1. Open the `sf_elevator` model.
2. Get a handle to the root object.

```
rt = sfroot;
```

3 Get handles to the subcharted states, `Elevator_A` and `Elevator_B`.

```
eA = rt.find('-isa','Stateflow.State','Name','Elevator_A');
eB = rt.find('-isa','Stateflow.State','Name','Elevator_B');
```

4 Change the chart positions of the two subcharts.

```
eA.set('Position', [20 100 70 60]);
eB.set('Position', [100 100 70 60]);
```

The following changes occur:

- The `Elevator_A` subchart moves to the location (20,100) from the upper left corner of the chart.
- The `Elevator_B` subchart moves to the location (100,100) from the upper left corner of the chart.
- For both subcharted states, the width is 70 and the height is 60.

Rules for Moving Objects Programmatically

- The object you move must be visible in the Stateflow Editor.

To...	You must be viewing...
Change the position or size of the subchart icon	The parent or container of the subchart
Change the size of the subchart boundary in the subviewer	The contents of the subchart

  - To switch to a chart-level view programmatically, use these commands:

```
rt = sfroot;
chart = rt.find('-isa','Stateflow.Chart','Name','name');
chart.view;
```

  - To view the contents of a subcharted state programmatically, use these commands:

```
rt = sfroot;
sub = rt.find('-isa','Stateflow.State','Name','name');
sub.view;
```

- When you view the contents of a subcharted state, box, or graphical function in the subviewer, you cannot change the position of the subchart boundary programmatically.
- For objects in a subcharted state, box, or graphical function, you cannot use the `set` method to move these objects between different levels of the chart hierarchy. See “Copying Objects” on page 1-32 for directions on copying and pasting objects from one container object to another.

Copying Objects

In this section...

Accessing the Clipboard Object

The Clipboard object (only one exists) provides an interface to the clipboard used in copying Stateflow objects. You cannot directly create or destroy the Clipboard object as you do other Stateflow API objects. However, you can attach a handle to it to use its properties and methods to copy Stateflow objects.

You create a handle to the Clipboard object by using the `sfclipboard` function as follows:

```
cb = sfclipboard
```

Clipboard objects have two methods, `copy` and `pasteTo`, that together provide the functionality to copy objects from one object to another. The `copy` method copies the specified objects to the Clipboard object, and the `pasteTo` method pastes the contents of the clipboard to a new container.

copy Method Limitations

The `copy` method is subject to these limitations for all objects:

- The objects you copy must be *all* graphical (states, boxes, functions, transitions, junctions) or *all* nongraphical (data, events, targets).

  You cannot copy a mixture of graphical and nongraphical objects to the clipboard in the same copy operation.

- To maintain the transition connections and containment relationships between copied objects, you must copy the entire array of related objects.

All related objects must be part of the array of objects copied to the clipboard. For example, if you try to copy two states connected by a transition to another container, you can only accomplish this by copying both the states and the transition at the same time. That is, you must do a single copy of a single array containing both the states and the transition that connects them.

If you copy a grouped state to the clipboard, you copy all the objects contained in the state, as well as all the relations among the objects in the grouped state. See "Copying by Grouping (Recommended)" on page 1-33.

Copying Graphical Objects

The `copy` method is subject to these limitations for all graphical objects:

- Copying graphical objects also copies the Data, Event, and Target objects that the graphical objects contain.
- If all copied objects are graphical, they must all be visible in the same subviewer.

  In other words, all graphical objects copied in a single copy command must reside in the same chart or subchart.

Copying by Grouping (Recommended)

Copying a grouped state in a Stateflow chart copies not only the state but all of its contents. By grouping a state before you copy it, you can copy it and all of its contained objects at all levels of containment with the Stateflow API. This method is the simplest way of copying objects. Use it whenever possible.

You use the Boolean `IsGrouped` property for a state to group that state. If you set the `IsGrouped` property for a state to a value of true (=1), it is grouped. If you set `IsGrouped` to a value of false (=0), the state is not grouped.

This example procedure copies state A to the chart X through grouping. In this example, assume that you already have a handle to state A and chart X through the MATLAB variables `sA` and `chX`, respectively:

1. If the state to copy is not already grouped, group it along with its contents by setting the `IsGrouped` property for that state to true (=1).

```
prevGrouping = sA.IsGrouped
if (prevGrouping == 0)
 sA.IsGrouped = 1
end
```

2 Get a handle to the Clipboard object.

```
cb = sfclipboard
```

3 Copy the grouped state to the clipboard using the Clipboard object.

```
cb.copy(sA)
```

4 Paste the grouped object to its new container.

```
cb.pasteTo(chX)
```

5 Set the copied state and its source state to its previous `IsGrouped` property value.

```
sA.IsGrouped=prevGrouping
sNew=chX.find('-isa','Stateflow.State','-and','Name',sA.Name)
sNew.IsGrouped=prevGrouping
```

Copying Objects Individually

You can copy specific objects from one object to another. However, in order to preserve transition connections and containment relations between objects, you must copy all the connected objects at once. To accomplish this, use the general technique of appending objects from successive finds in the MATLAB workspace to a growing array of objects before copying the finished object array to the clipboard.

Using the example of the Stateflow chart at the end of "Creating New Objects in the Chart" on page 1-9, you can copy states A1, A2, and the transition connecting them to another state, B, with these API commands, where `sA` and `sB` are object handles to states A and B, respectively.

```
objArrayS = sA.find('-isa','Stateflow.State','-depth',1);
objArrayT = sA.find('-isa','Stateflow.Transition','-depth',1);
sourceObjs = {objArrayS ; objArrayT};
```

```
cb = sfclipboard;
cb.copy(sourceObjs);
cb.pasteTo(sB);
```

You can also copy nongraphical data, events, and target objects individually. However, since there is no way for these objects to find their new owners, you must ensure that you copy each of these objects separately to its appropriate owner object.

Note Copying objects individually is harder than copying grouped objects. See "Copying by Grouping (Recommended)" on page 1-33.

Using the Editor Object

In this section...

About Editor Objects

The Editor object provides access to the purely graphical properties and methods of Chart objects. Each Chart object has its own Editor object.

Accessing the Editor Object

You cannot directly create or destroy the Editor and Clipboard objects as you do other Stateflow API objects. However, you can attach a handle to them to use their properties and methods for modifications to Stateflow charts.

When you create a chart, an Editor object is automatically created for it. If `ch` is a workspace handle to a chart, you create a handle to the Editor object for that chart with this command:

```
ed = ch.Editor
```

Changing the Display in the Stateflow Editor

Use the handle `ed` from the preceding example to access the Editor object properties and methods. For example, this command calls the `zoomIn` method to zoom in the chart by a factor of 20%:

```
ed.zoomIn
```

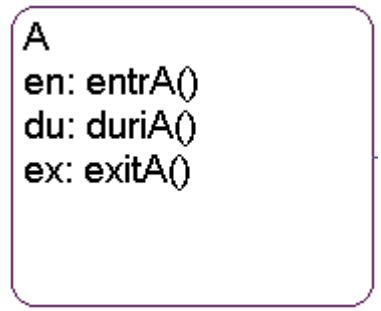
Or, you can simply set the `ZoomFactor` property to an absolute zoom factor of 150%:

```
ed.ZoomFactor = 1.5
```

You can also use an Editor object to change the window position of the Stateflow Editor. For a reference to all the Editor object's properties and methods, see "Editor Properties" on page 2-30 and "Editor Methods" on page 2-31.

Entering Multiline Labels

In the examples shown thus far of entering labels for states and transitions, only a simple one-line expression has been used. This figure shows state A with a multiline label.



There are two ways to enter multiline labels for both states and transitions. In these examples, `sA` is a handle to the State object in the Stateflow API for state A:

- Use the MATLAB function `sprintf`:

  ```
  str = sprintf('A\nen: entrA()\ndu: duriA()\nex: exitA()')
  sA.LabelString = str
  ```

  In this example, carriage returns are inserted into a string expression with the escape sequence \n.

- Use a concatenated string expression:

  ```
  str = ['A',10,'entr: entrA()',10,'du: duriA()',
        10,'ex: exitA()']
  sA.LabelString = str
  ```

  In this example, carriage returns are inserted into a concatenated string expression with the ASCII equivalent of a carriage return, the integer 10.

Creating Default Transitions

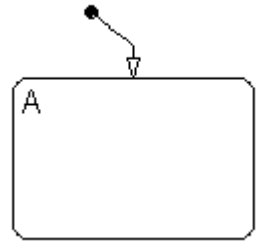
Default transitions differ from normal transitions in not having a source object. You can create a default transition with these steps:

1. Create a transition.
2. Attach the destination end of the transition to an object.
3. Position the source endpoint for the transition.

If you assume that the variable sA is a handle to state A, these commands create a default transition and position its source 25 pixels above and 15 pixels to the left of the top midpoint of state A:

```
dt = Stateflow.Transition(sA)
dt.Destination = sA
dt.DestinationOClock = 0
xsource = sA.Position(1)+sA.Position(3)/2-15
ysource = sA.Position(2)-25
dt.SourceEndPoint = [xsource ysource]
```

The created default transition has the following appearance:

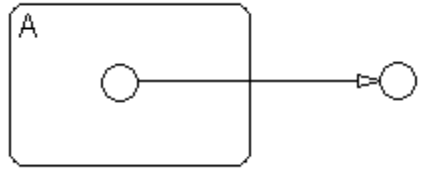


This method is also used for adding the default transitions toward the end of the example Stateflow chart constructed in “Creating New Objects in the Chart” on page 1-9.

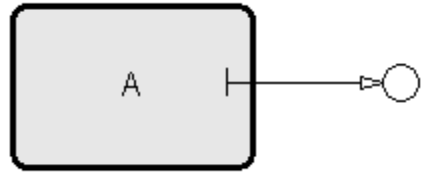
Making Supertransitions

The Stateflow API does not currently support the direct creation of supertransitions. Supertransitions are transitions between a state or junction in a top-level chart and a state or junction in one of its subcharts, or between states residing in different subcharts at the same or different levels in a chart. For a better understanding of supertransitions, see "What Is a Supertransition?" in the Stateflow and Stateflow Coder User's Guide.

You can use a workaround for indirectly creating supertransitions. In this example, a supertransition is desired from a junction inside a subchart to a junction outside the subchart. In order to use the Stateflow API to create the supertransition in this example, first use the API to create the superstate as an ordinary state with a transition between its contained junction and a junction outside it.



Now set the `IsSubchart` property of the state A to true (=1).



This makes state `A` a subchart, and the transition between the junctions is now a supertransition.

You can also connect supertransitions to and from objects in an existing subchart (state A, for example) with these steps:

1. Save the original position of subchart A to a temporary workspace variable.

   For example, if the subchart A has the API handle sA, store its position with this command:

   ```
   sA_pos = sA.Position
   ```

2. Convert subchart A to a state by setting its IsSubchart property to false (=0).

   ```
   sA.IsSubchart = 0
   ```

3. Ungroup state A by setting its IsGrouped property to false (=0).

   ```
   sA.IsGrouped = 0
   ```

   When convert a subchart a normal state, it stays grouped to hide the contents of the subchart. When you ungroup the subchart, it might resize to display its contents.

4. Make the necessary transition connections.

   See "Creating New Objects in the Chart" on page 1-9 for an example of creating a transition.

5. Set the IsSubchart property of state A back to true (=1).

   For example, `sA.IsSubchart = 1`

6. Assign subchart A its original position.

   ```
   sA.Position = sA_pos
   ```

   When you convert a subchart to a normal state and ungroup it, it might resize to accommodate the size of its contents. The first step of this procedure stores the original position of the subchart so that this can be restored after the transition connection is made.

Creating a MATLAB Script of API Commands

In "Quick Start for the Stateflow API" on page 1-7, you created and saved a new model through a series of Stateflow API commands. You can include the same API commands in the following MATLAB script. This script allows you to quickly recreate the same model with the single command `makeMyModel`.

```
function makeMyModel
% Get all previous models loaded

rt = sfroot;
prev_models = rt.find('-isa','Simulink.BlockDiagram');

% Create new model, and get current models

sfnew;
curr_models = rt.find('-isa','Simulink.BlockDiagram');

% New model is current models - previous models

m = setdiff(curr_models, prev_models);

% Get chart in new model

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

% Create state A in chart

sA = Stateflow.State(chart);
sA.Name = 'A';
sA.Position = [45 45 300 150];

% Create state A1 inside of state A

sA1 = Stateflow.State(chart);
sA1.Name = 'A1';
sA1.Position = [80 80 90 80];

% Create state A2 inside of state A
```

```
sA2 = Stateflow.State(chart);
sA2.Name = 'A2';
sA2.Position = [220 80 90 80];

% Create a transition from A1 to A2

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

% Create state A11 inside of state A1

sA11 = Stateflow.State(chart);
sA11.Name = 'A11';
sA11.Position = [110 110 35 35];

% Create a transition from A1 to A11

tA1A11 = Stateflow.Transition(chart);
tA1A11.Source = sA1;
tA1A11.Destination = sA11;
tA1A11.SourceOClock = 1.;
tA1A11.DestinationOClock = 1.;

% Label transitions A1-A11 and A1-A2
%   to listen for events E1 and E2

tA1A2.LabelString = 'E1';
tA1A11.LabelString = 'E2';

% Create the Events E1 and E2

E1 = Stateflow.Event(chart);
E1.Name = 'E1';

% Move label for transition A1-A1 to the right a bit

pos = tA1A2.LabelPosition;
```

```
pos(1) = pos(1)+15;
tA1A2.LabelPosition = pos;

% Create a default transition to state A

dtA = Stateflow.Transition(chart);
dtA.Destination = sA;
dtA.DestinationOClock = 0;
xsource = sA.Position(1)+sA.Position(3)/2-10;
ysource = sA.Position(2)-20;
dtA.SourceEndPoint = [xsource ysource];

% Create a default transition to state A1

dtA1 = Stateflow.Transition(chart);
dtA1.Destination = sA1;
dtA1.DestinationOClock = 0;
xsource = sA1.Position(1)+sA1.Position(3)/2-10;
ysource = sA1.Position(2)-20;
dtA1.SourceEndPoint = [xsource ysource];
```

2

API Object Reference

Reference Table Column Descriptions

Reference tables for Stateflow API properties and methods have these columns:

- **Name** — The name for the property or method. Each property or method has a name that you use in dot notation along with a Stateflow object to set or obtain the property's value or call the method.
- **Type** — A data type for the property. Some types are other Stateflow API objects, such as the `Machine` property, which is the Machine object that contains this object.
- **Access** — An access type for the property. Properties that are listed as RW (read/write) can be read and changed. For example, the `Name` and `Description` properties of particular objects are RW. However, some properties are RO (read-only) because they are set by the MATLAB workspace itself.
- **Description** — A description for the property or method. For some properties, the equivalent GUI operations for setting it are also given.

All Object Methods

The following methods apply to all API objects including those of Stateflow charts. Only object-exclusive methods appear when you use the method `methods` to display methods for an object. However, the tables of methods for each API object that follow do list these methods as if they were their own.

For details on each method, see Chapter 4, "API Method Reference".

Method	Description
`delete`	Delete this object. Used with all objects except the Root, Machine, Chart, Clipboard, and Editor objects.
`disp`	Display the property names and their settings for this object.
`find`	Find all objects of this object that meet the specified criteria.
`get`	Return the specified property settings for this object.
`methods`	Display all nonglobal methods of this object.
`set`	Set the specified property of this object with a specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this object.
`up`	Return the parent (container) object of this object.

Box Properties

Stateflow API objects of type Box have the properties shown below. See also "Box Methods" on page 2-8.

Property	Type	Access	Description
`ArrowSize`	Double	RW	Size of transition arrows coming into this box (default = 8). Equivalent to selecting **Arrowhead Size** from the context menu for this box.
`BadIntersection`	Boolean	RO	If `true`, this box graphically intersects a state, Embedded MATLAB function, graphical function, truth table, or another box.
`Chart`	Chart	RO	Chart object containing this box.
`Description`	String	RW	Description of this box (default = `''`). Equivalent to entering a description in the **Description** field of the Box properties dialog box.
`Document`	String	RW	Document link to this box (default = `''`). Equivalent to entering the **Document Link** field of the Box properties dialog box.
`FontSize`	Double	RW	Size of the font (default = `12`) for the label text of this box. This property overrides the font size set for this box at creation by the `StateFont.Size` property of the containing Chart's object. Equivalent to selecting **Font Size** > **<font size>** in the context menu for this box.
`Id`	Integer	RO	Unique identifier assigned to this box to distinguish it from other objects loaded in memory.
`IsGrouped`	Boolean	RW	If set to `true` (default = `false`), group this box.

Property	Type	Access	Description
`IsSubchart`	Boolean	RW	If set to `true` (default = `false`), make this box a subchart.
`LabelString`	String	RW	Label for this box (default = `'?'`). Equivalent to typing the label for this box in its label text field in the Stateflow Editor.
`Machine`	Machine	RO	Machine that contains this box.
`Name`	String	RW	Name of this box (default = `''`). Equivalent to typing this box's name into the beginning of the label text field for this box in the Stateflow Editor.
`Position`	Rect	RW	Position and size of this box in the Stateflow chart, given in the form of a 1-by-4 array (default is `[0 0 90 60]`) consisting of the following: • (x,y) coordinates for the box's left upper vertex relative to the upper left vertex of the Stateflow Editor workspace • Width and height of the box
`Subviewer`	Chart or State	RO	State or chart in which this box can be graphically viewed.
`Tag`	Any Type	RW	Holds data of any type (default = `[]`) for this box.

Box Methods

Box objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Box Properties" on page 2-6.

Method	Description
`defaultTransitions`	Return the default transitions in this box at the top level of containment.
`delete`	Delete this box from the Stateflow chart.
`dialog`	Display the Box properties dialog box.
`disp`	Display the property names and their settings for this Box object.
`find`	Find all objects that this box contains that meet the specified criteria.
`fitToView`	Zoom in on this box and highlight it in the Stateflow Editor.
`get`	Return the specified property settings for this box.
`help`	Display a list of properties for this Box object with short descriptions.
`innerTransitions`	Return the inner transitions that originate with this box and terminate on a contained object.
`methods`	Display all nonglobal methods of this Box object.
`outerTransitions`	Return an array of transitions that exit the outer edge of this box and terminate on an object outside the containment of this box.
`set`	Set the specified property of this Box object with the specified value.
`sourcedTransitions`	Return all inner and outer transitions whose source is this box.
`struct`	Return and display a MATLAB structure containing the property settings of this Box object.
`view`	Display this box's chart in the Stateflow Editor with this box highlighted.

Chart Properties

Stateflow API objects of type Chart have the properties shown below. See also "Chart Methods" on page 2-19.

Property	Type	Access	Description
`ChartColor`	[R,G,B]	RW	Set the background color of your chart by using a 1-by-3 RGB array (default = `[1 0.9608 0.8824]`) with each value normalized on a scale of 0 to 1.
`ChartUpdate`	Enum	RW	Activation method of this chart. Can be `'INHERITED'` (default), `'DISCRETE'`, or `'CONTINUOUS'`. Equivalent to the **Update method** field in the Chart properties dialog box, which takes one of these selections: **Inherited**, **Discrete**, **Continuous**.

`Debug.`
`Breakpoints.`
`OnEntry`
	Boolean	RW	If set to `true` (default = `false`), set the chart entry breakpoint for this chart. Equivalent to selecting the **Chart Entry** check box in the Chart properties dialog box.

`Decomposition`	Enum	RW	Set this property to `'EXCLUSIVE_OR'` (default) to specify exclusive (OR) decomposition for the states at the first level of containment in this chart. Set to `'PARALLEL_AND'` to specify parallel (AND) decomposition for these states. Equivalent to the **Decomposition** selection in the context menu for a specific level of the chart hierarchy.

Property	Type	Access	Description
`Description`	String	RW	Description (default = `''`) of this state. Equivalent to entering a description in the **Description** field of the Chart properties dialog box.
`Dirty`	Boolean	RW	If set to `true` (default = `false`), this chart has changed since being opened or saved.
`Document`	String	RW	Document link (default = `''`) to this chart. Equivalent to entering a link in the **Document Link** field of the Chart properties dialog box.
`Editor`	Editor	RO	Editor object for this chart.
`EnableBitOps`	Boolean	RW	If set to `true` (default = `false`), enables C-like bit operations in generated code for this chart. Equivalent to selecting the **Enable C-bit operations** check box in the Chart properties dialog box.
`EnableNonTerminalStates`	Boolean	RW	If set to `true` (default = `false`), enables super step semantics for the chart, as described in "Execution of a Chart with Super Step Semantics" in the Stateflow and Stateflow Coder User's Guide.
`EnableZeroCrossings`	Boolean	RW	If set to `true` (default = `true`), enables zero-crossing detection on state transitions for continuous-time simulation of Stateflow charts. Applies only when the `ChartUpdate` property for this chart is set to `'CONTINUOUS'`. See "When to Enable Zero-Crossing Detection" in the Stateflow and Stateflow Coder User's Guide.

Property	Type	Access	Description
`ErrorColor`	[R,G,B]	RW	Set the color for errors in your chart by using a 1-by-3 RGB array (default = `[1 0 0]`) with each value normalized on a scale of 0 to 1. Equivalent to changing the **Error** color in the Colors & Fonts dialog box under **Edit > Style**.
`ExecuteAtInitialization`	Boolean	RW	If set to `true` (default = `false`), this chart's state configuration is initialized at time zero instead of at the first input event. Equivalent to selecting the **Execute (enter) Chart At Initialization** check box in the Chart properties dialog box.
`ExportChartFunctions`	Boolean	RW	If set to `true` (default = `false`), graphical functions at chart level are made global. Equivalent to selecting the **Export Chart Level Graphical Functions (Make Global)** check box in the Chart properties dialog box.
`Iced`	Boolean	RO	Equivalent to property `Locked` (default = `false`) except that this property is used internally to lock this chart from change during activities such as simulation.
`Id`	Integer	RO	Unique identifier assigned to this chart to distinguish it from other objects loaded in memory.
`InitializeOutput`	Boolean	RW	Applies the initial value of outputs every time a chart wakes up, not only at time 0. See "Setting Properties for a Single Chart".

Property	Type	Access	Description
`JunctionColor`	[R,G,B]	RW	Set the color for junctions in your chart by using a 1-by-3 RGB array (default = `[0.6824 0.3294 0]`) with each value normalized on a scale of 0 to 1. Equivalent to changing the **Junction** color in the Colors & Fonts dialog box under **Edit > Style**.
`Locked`	Boolean	RW	If set to `true` (default = `false`), mark this chart as read-only and prohibit any write operations on it. Equivalent to selecting the **Lock Editor** check box in the Chart properties dialog box.
`Machine`	Machine	RO	Machine that contains this chart.
`Name`	String	RW	Name of this chart (default = `'Chart'`). Equivalent to changing the name of this chart's Stateflow block in a Simulink model.
`NonTerminalMaxCounts`	String	RW	Maximum number of transitions a Stateflow chart can take in one super step. Applies only when `EnableNonTerminalStates` is `true`. See "Execution of a Chart with Super Step Semantics" in the Stateflow and Stateflow Coder User's Guide.
`NonTerminalUnstableBehavior`	Enum	RW	Behavior of a Stateflow chart during simulation if it exceeds the maximum number of transitions specified in the `NonTerminalMaxCounts` property in a super step before reaching a stable state. Set this property to `'PROCEED'` (default) if you want the chart to go back to sleep with the last active state configuration. Set this property to `'THROW ERROR'` if you want the chart to generate an error. Applies only when `EnableNonTerminalStates`

Property	Type	Access	Description
			is `true`. See "Execution of a Chart with Super Step Semantics" in the Stateflow and Stateflow Coder User's Guide.
`SampleTime`	String	RW	Sample time for activating this chart (default = `''`). Applies only when the `ChartUpdate` property for this chart is set to `'DISCRETE'` (= **Discrete** in the **Update method** field in the Chart properties dialog box).
`SelectionColor`	[R,G,B]	RW	Color of selected items for this chart in a 1-by-3 RGB array (default value `[1 0 0.5176]`) with each value normalized on a scale of 0 to 1. Equivalent to changing the **Selection** color in the Colors & Fonts dialog box under **Edit > Style**.
`StateColor`	[R,G,B]	RW	Color of the state box in a 1-by-3 RGB array (default value `[0 0 0]`) with each value normalized on a scale of 0 to 1. Equivalent to changing the **State/Frame** color in the Colors & Fonts dialog box under **Edit > Style**.
`StateFont.Angle`	Enum	RW	Font angle for the labels of State, Box, Function, and Note objects. Can be `'ITALIC'` or `'NORMAL'` (default). Equivalent to **Italic** and **Regular** settings when changing the font style of **StateLabel** in the Colors & Fonts dialog box under **Edit > Style**. Use with property `StateFont.Weight` to achieve Bold Italic style. You can individually override this property with the `Font.Angle` property for Note objects.

Property	Type	Access	Description
`StateFont.Name`	String	RW	Font style (default = `'Helvetica'`) used for the labels of State, Box, Function, and Note objects. Enter a string for the font name (there are no selectable values). Font remains set to previous font for unrecognized font strings. Equivalent to changing the font of **StateLabel** in the Colors & Fonts dialog box under **Edit > Style**.
`StateFont.Size`	Integer	RW	Default font size for the labels of a new State, Box, Function, or Note object. Equivalent to changing the font size of **StateLabel** in the Colors & Fonts dialog box under **Edit > Style**. You can change the font size for an existing State, Box, or Function object with the `FontSize` property of that object. You can change the font size for an existing Note object with its `Font.Size` property.
`StateFont.Weight`	Enum	RW	Font weight for state labels. Can be `'BOLD'` or `'NORMAL'` (default). Equivalent to the **Bold** and **Regular** settings of **StateLabel** in the Colors & Fonts dialog box under **Edit > Style**. Use with the property `StateFont.Angle` to achieve Bold Italic style. You can individually override this property with the `Font.Weight` property for Note objects.

Property	Type	Access	Description
`StateLabelColor`	[R,G,B]	RW	Color of the state labels for this chart in a 1-by-3 RGB array (default = `[0 0 0]`) with each value normalized on a scale of 0 to 1. Equivalent to changing the label color of **StateLabel** in the Colors & Fonts dialog box under **Edit > Style**.
`StateMachineType`	Enum	RW	Type of state chart to create. Default is *Classic*, which provides the full set of Stateflow chart semantics. You can also create Mealy and Moore charts, which use a subset of Stateflow chart semantics (see "Building Mealy and Moore Charts").
`StatesWhenEnabling`	String	RW	If your chart contains function-call input events, specify how states behave when the event reenables the chart. Possible values include: • `held` — Maintain most recent values of the states. • `reset` — Revert to the initial conditions of the states. • `inherit` — Inherit this setting from the parent subsystem. If your chart does not contain function-call input events, this property has no effect. For more information, see "Controlling States When Function-Call Inputs Reenable Charts".

Property	Type	Access	Description
`StrongDataTyping WithSimulink`	Boolean	RW	If set to `true` (default), set strong data typing with Simulink I/O. Equivalent to selecting the **Use Strong Data Typing with Simulink I/O** check box in the Chart properties dialog box.
`SupportVariableSizing`	Boolean	RW	If set to `true` (default), support chart input and output data that vary in dimension during simulation. Equivalent to selecting the **Support variable-size arrays** check box in the Chart properties dialog box.
`Tag`	Any Type	RW	A field you can use to hold data of any type for this chart (default = `[]`).
`TransitionColor`	[R,G,B]	RW	Set the color for transitions in your chart by using a 1-by-3 RGB array (default = `[0.2902 0.3294 0.6039]`) with each value normalized on a scale of 0 to 1. Equivalent to changing the **Transition** color in the Colors & Fonts dialog box under **Edit > Style**.
`TransitionFont. Angle`	Enum	RW	Font angle for state labels. Can be `'ITALIC'` or `'NORMAL'` (default). Equivalent to **Italic** and **Regular** settings when you change the font style of **TransitionLabel** in the Colors & Fonts dialog box under **Edit > Style**. Use with property `StateFont.Weight` to achieve Bold Italic style.

Property	Type	Access	Description
`TransitionFont.Name`	String	RW	Font style (default = `'Helvetica'`) used for transition labels. Enter a string for font name (there are no selectable values). Font remains set to previous font for unrecognized font strings. Equivalent to changing the font of **TransitionLabel** in the Colors & Fonts dialog box under **Edit > Style**.
`TransitionFont.Size`	Integer	RW	Default font size (default = `12`) for transition labels. Truncated to closest whole number less than or equal to entered value. Equivalent to changing the font size of **TransitionLabel** in the Colors & Fonts dialog box under **Edit > Style**.
`TransitionFont.Weight`	Enum	RW	Font weight for transition labels. Can be `'BOLD'` or `'NORMAL'` (default). Equivalent to **Bold** and **Regular** settings when you change the font style of **TransitionLabel** in the Colors & Fonts dialog box under **Edit > Style**. Use with property `StateFont.Angle` to achieve Bold Italic style.
`TransitionLabelColor`	[R,G,B]	RW	Color of the transition labels for this chart in a 1-by-3 RGB array (default = `[0.2902 0.3294 0.6039]`) with each value normalized on a scale of 0 to 1. Equivalent to changing the label color of **TransitionLabel** in the Colors & Fonts dialog box under **Edit > Style**.

Property	Type	Access	Description
`UserSpecifiedState TransitionExecutionOrder`	Boolean	RW	If set to `true` (default = `false`), you have complete control of the order in which transitions originating from a source are tested for execution. Equivalent to selecting the **User specified state/transition execution order** check box in the Chart properties dialog box.
`Visible`	Boolean	RW	If set to `true` (default), display this chart in the Stateflow Editor.

Chart Methods

Chart objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Chart Properties" on page 2-9.

Method	Description
`defaultTransitions`	Return the default transitions in this Stateflow chart at the top level of containment.
`dialog`	Display the Chart properties dialog box.
`disp`	Display the property names and their settings for this Chart object.
`find`	Find all objects that this chart contains that meet the specified criteria.
`fitToView`	Zoom in on this chart in the Stateflow Editor.
`get`	Return the specified property settings for this chart.
`help`	Display a list of properties for this Chart object with short descriptions.
`methods`	Display all nonglobal methods of this Chart object.
`parse`	Parse this chart.
`set`	Set the specified property of this Chart object with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this Chart object.
`view`	Display this chart in the Stateflow Editor.

Clipboard Methods

The Clipboard object has the methods in the table below. For details on each method, see Chapter 4, “API Method Reference”.

Method	Description
`copy`	Copy the objects specified to this Clipboard object.
`get`	Return the specified property settings for this Clipboard object.
`help`	Display a list of properties for this Clipboard object with short descriptions.
`methods`	Display all nonglobal methods of this Clipboard object.
`pasteTo`	Paste the contents of this clipboard to the specified container object.
`set`	Set the specified property of this Clipboard object with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this Clipboard object.

Constructor Methods

The following methods create a new Stateflow object for a parent object specified as an argument in the general expression `o = Stateflow.Object(p)`, where `o` is a handle to an API object for the new Stateflow object, `p` is a handle to the parent object, and *`Object`* is the type of the object:

For details on each method, see Chapter 4, "API Method Reference".

Method	Description
`Stateflow.Box`	Create a box for a parent chart, state, box, or function.
`Stateflow.Data`	Create a data for a parent machine, chart, state, box, or function.
`Stateflow.EMFunction`	Create an Embedded MATLAB function for a parent chart or state.
`Stateflow.Event`	Create an event for a parent machine, chart, state, box, or function.
`Stateflow.Function`	Create a graphical function for a parent chart, state, box, or function.
`Stateflow.Junction`	Create a junction for a parent chart, state, box, or function.
`Stateflow.Note`	Create a note for a parent chart, state, box, or function.
`Stateflow.SLFunction`	Create a Simulink function for a parent chart or state.
`Stateflow.State`	Create a state for a parent chart, state, box, or function.
`Stateflow.Target`	Create a target for a parent machine.
`Stateflow.Transition`	Create a transition for a parent chart, state, box, or function.
`Stateflow.TruthTable`	Create a truth table for a parent chart or state.

Data Properties

Stateflow API objects of type Data have the properties shown below. See also "Data Methods" on page 2-29.

Property	Type	Access	Description
`CompiledSize`	String	RW	Size of data as determined by the compiler.
`CompiledType`	String	RW	Type of data as determined by the compiler.
`Debug.Watch`	Integer	RW	If set to `1` (default = `0`), causes the Debugger window to halt execution if this data is modified. Setting this property to `1` is equivalent to selecting the **Watch in debugger** check box in the Data properties dialog box.
`Description`	String	RW	Description of this data (default = `''`). Equivalent to entering a description in the **Description** field of the Data properties dialog box.
`Document`	String	RW	Document link to this data (default = `''`). Equivalent to entering a link in the **Document link** field of the Data properties dialog box.
`Id`	Integer	RO	Unique identifier assigned to this data to distinguish it from other objects in the model.
`InitializeMethod`	String	RW	Method for initializing value of this data, based on scope of data: • If scope is **Local** or **Output**, you can set `InitializeMethod` to **Expression** or **Parameter**. Equivalent to setting the **Initial value** field in the Data properties dialog box. • If scope is **Parameter**, **Input**, or **Data Store Memory**, you should set `InitializeMethod` to **Not Needed** as a read-only property. • If scope is **Constant**, you should set `InitializeMethod` to **Expression** as a read-only property.
`Machine`	Machine	RO	Stateflow machine that contains this data.

Property	Type	Access	Description
`Name`	String	RW	Name of this data. Equivalent to entering the name of this data in the **Name** field of the Data properties dialog box.
`OutputState`	Integer	RO	If set to `1` (default = `0`), this data represents the activity of the state in which it is defined as an output to a Simulink model. See "Outputting State Activity to a Simulink Model" in the Stateflow and Stateflow Coder User's Guide. Create this data for a state by using the State method `outputData`. Equivalent to selecting the **Output State Activity** check box for the state.
`Path`	String	RO	Location of this data in the model hierarchy
`Port`	Integer	RW	Port index number for this data (default = `1`).

`Props.`
`Array.`
`FirstIndex`
	String	RW	Index of the first element of this data (default = `0`) if it is an array (that is, `Props.Array.Size` > 1). Equivalent to entering a value of zero or greater in the **First index** field in the Data properties dialog box.

`Props.`
`Array.`
`Size`
	String	RW	Size of this data. Assigning a positive value indicates that the data is an array of the specified size (default = `0`). Equivalent to entering a positive value in the **Size** column for this data in the Model Explorer or the **Size** field in the Data properties dialog box.

`Props.`
`Initial`
`Value`
	String	RW	Initial value of this data (default = `''`). Equivalent to entering a value in the **Initial value** column for this data in the Model Explorer or the**Initial value** field in the Data properties dialog box.

`Props.`
`Range.`
`Maximum`
	String	RW	Maximum value (default = `''`) that this data can have during execution or simulation of the state machine. Equivalent to entering a value in the **Maximum** column for this data in the Model Explorer or the **Maximum** field in the Data properties dialog box.

Property	Type	Access	Description
`Props.Range.Minimum`	String	RW	Minimum value (default = `''`) that this data can have during execution or simulation of the state machine. Equivalent to entering a value in the **Minimum** column for this data in the Model Explorer or the **Minimum** field in the Data properties dialog box.
`Props.Type.BusObject`	String	RW	If `Props.Method` is `Bus Object`, you must set this property to the name of the `Simulink.Bus` object that defines this data (see "Working with Structures and Bus Signals in Stateflow Charts" in the Stateflow and Stateflow Coder User's Guide). Equivalent to setting the data type **Mode** to `Bus Object`, and entering the name of a `Simulink.Bus` object in the Data Type Assistant of the Data properties dialog box.
`Props.Type.Expression`	String	RW	If `Props.Method` is `Expression`, you must set this property to an expression that evaluates to a data type (see "Entering Expressions and Parameters for Data Properties" in the Stateflow and Stateflow Coder User's Guide). Equivalent to setting the data type **Mode** to `Expression`, and entering an expression in the Data Type Assistant of the Data properties dialog box.
`Props.Type.Fixpt.Bias`	String	RW	The bias value for fixed-point data (default = `0.0`) when `Props.Type.Fixpt.ScalingMode` equals **Slope and bias**; otherwise this value is ignored. Equivalent to entering a real number in the **Bias** field of the Data Type Assistant in the Data properties dialog box.
`Props.Type.Fixpt.FractionLength`	String	RW	The location of the binary point in fixed-point data (default = `0`) when `Props.Type.Fixpt.ScalingMode` equals **Binary point**; otherwise this value is ignored. Equivalent to entering a positive or negative integer in the **Fraction length** field of the Data Type Assistant in the Data properties dialog box.

Property	Type	Access	Description

`Props.`
`Type.`
`Fixpt.`
`Lock`
	Integer	RW	If set to `1` (default = `0`), prevents Simulink software from replacing the current fixed-point type with a type that the Fixed-Point Tool or Fixed-Point Advisor chooses. Equivalent to selecting **Lock data type setting against changes by the fixed-point tools** in the Data properties dialog box.

`Props.`
`Type.`
`Fixpt.`
`ScalingMode`
	String	RW	Method for scaling fixed point data to avoid overflow conditions and minimize quantization errors. The settings are:
			• **None** (the default)
			• **Binary point**
			• **Slope and bias**
			Equivalent to setting the **Scaling** field of the Data Type Assistant in the Data properties dialog box.

`Props.`
`Type.`
`Fixpt.`
`Slope`
	String	RW	The slope value for fixed-point data (default = `1.0`) when `Props.Type.Fixpt.ScalingMode` equals **Slope and bias**; otherwise this value is ignored. Equivalent to entering a positive real number in the **Slope** field of the Data Type Assistant in the Data properties dialog box.

`Props.`
`Type.`
`Method`
	String	RW	Method for setting the type of this data, based on scope:
			• If scope is **Local** , you can set this property to **Built-in**, **Expression**, **Bus Object**, or **Fixed point**.
			• If scope is **Constant**, you can set this property to **Built-in**, **Expression**, or **Fixed point**.
			• If scope is **Parameter**, you can set this property to **Inherited**, **Built-in**, **Expression**, or **Fixed point**.

Property	Type	Access	Description
			• If scope is **Input** or **Output**, you can set this property to **Inherited**, **Built-in**, **Expression**, **Bus Object**, or **Fixed point**. • If scope is **Data Store Memory**, you should set `Props.Type.Method` to **Inherited** as a read-only property. Equivalent to setting the **Mode** field of the Data Type Assistant in the Data properties dialog box.

`Props.`
`Type.`
`Signed`
	Integer	RW	If set to `1`, indicates that fixed-point data should be signed. The default value is `0`, indicating that the data is unsigned. Equivalent to setting the **Sign** field of the Data Type Assistant in the Data properties dialog box.

`Props.`
`Type.`
`Units`
	String	RW	Units of measurement for the data value (default = `''`).

`Props.`
`Type.`
`WordLength`
	String	RW	Size in bits of the word that will hold the quantized integer of fixed-point data. Equivalent to entering an integer between 0 and 32 in the **Word length** field of the Data Type Assistant in the Data properties dialog box.

`Props.` `SaveToWorkspace`	Integer	RW	If set to `1`, assigns the value of the data item to a variable of the same name in the model workspace at the end of simulation. The default value is `0`, indicating that the data is not saved to the base workspace. Equivalent to selecting the **Save final value to base workspace** check box in the Data properties dialog box.

Property	Type	Access	Description
Scope	Enum	RW	Scope of this data: • **Local** — Data defined in current Stateflow chart. • **Constant** — Read-only constant value that is visible to the parent Stateflow object and its children. • **Parameter** — Constant defined in the MATLAB workspace or derived from a Simulink parameter that is defined and initialized in the parent masked subsystem. • **Input** — If the parent is a graphical, truth table, or Embedded MATLAB function, the data is an input argument. Otherwise, it is provided by the Simulink model to the Stateflow chart via an input port. • **Output** — If the parent is a graphical, truth table, or Embedded MATLAB function, the data is a return value. Otherwise, it is provided by the Stateflow chart to the Simulink model via an output port. • **Data Store Memory** — Data that binds to a Simulink data store. • **Temporary** — Data that persists only during the execution of a function. • **Imported** — Data parented by the Simulink model, but defined in external code embedded in the Stateflow machine. • **Exported** — Data from the Simulink model that you provide to external code. Parent must be a Stateflow machine. Equivalent to setting the **Scope** field in the Data properties dialog box. See "Scope" in the Stateflow and Stateflow Coder User's Guide.

Property	Type	Access	Description
`SaveTo Workspace`	Integer	RW	If set to `1` (default = `0`), this data is saved to the MATLAB workspace. Setting this property to `1` is equivalent to selecting the **SaveToWorkspace** column entry for this data in the Model Explorer or selecting the **Save final value to base workspace** field in the Data properties dialog box.
`TestPoint`	Integer	RW	If set to `1` (default = `0`), sets this data as a Stateflow test point. You can monitor individual Stateflow test points with a floating scope during model simulation. You can also log test point values into MATLAB workspace objects. See "Monitoring Test Points in Stateflow Charts" in the Stateflow and Stateflow Coder User's Guide.

Data Methods

Data objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Data Properties" on page 2-22.

Method	Description
`delete`	Delete this data.
`dialog`	Display the Data properties dialog box.
`disp`	Display the property names and their settings for this Data object.
`get`	Return the specified property settings for this data.
`help`	Display a list of properties for this Data object with short descriptions.
`methods`	Display all nonglobal methods of this Data object.
`set`	Set the specified property of this Data object with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this Data object.
`view`	Display this data in the Data properties dialog box.

Editor Properties

The Editor object has the properties shown below. See also "Editor Methods" on page 2-31.

Property	Type	Access	Description
`WindowPosition`	Rect	RW	Position and size of this Stateflow chart given in the form of a 1-by-4 array consisting of the following: • (x,y) coordinates for the window's left bottom vertex relative to the lower left corner of the screen • Width and height of the box Default value = `[124.3125 182.8125 417 348.75]`
`ZoomFactor`	Double	RW	View magnification level (zoom factor) of this chart in the Stateflow Editor. A value of 1 corresponds to a zoom factor of 100%, 2 to a value of 200%, and so on. Default value = 1.

Editor Methods

The Editor object has the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Editor Properties" on page 2-30.

Method	Description
`disp`	Display the property names and their settings for this Editor object.
`get`	Return the specified property settings for the Editor object.
`help`	Display a list of properties for this Editor object with short descriptions.
`methods`	Display all nonglobal methods of this Editor object.
`set`	Set the specified property of this Editor object with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this Editor object.
`zoomIn and zoomOut`	Cause the Stateflow chart to zoom in or zoom out in the Stateflow Editor.

Embedded MATLAB Function Properties

Stateflow API objects for Embedded MATLAB functions have the properties shown below. See also "Embedded MATLAB Function Methods" on page 2-35.

Property	Type	Access	Description
`ArrowSize`	Double	RW	Size of transition arrows coming into the Embedded MATLAB function in the Stateflow chart (default = `8`).
`BadIntersection`	Boolean	RO	If `true`, this Embedded MATLAB function graphically intersects a state, box, graphical function, truth table, or another Embedded MATLAB function.
`Chart`	Chart	RO	Chart object containing this Embedded MATLAB function.
`Description`	String	RW	Description of this Embedded MATLAB function (default = `''`). Equivalent to entering a description in the **Description** field of the properties dialog box for this Embedded MATLAB function.
`Document`	String	RW	Document link to this Embedded MATLAB function. Equivalent to entering the **Document Link** field of the properties dialog box for this Embedded MATLAB function.
`FontSize`	Double	RW	Size of the (default = `12`) font of the label text for this Embedded MATLAB function. This property overrides the font size set for this Embedded MATLAB function at creation by the `StateFont.Size` property of the containing Chart object. Equivalent to selecting **Font Size** > *font size* in the context menu for this Embedded MATLAB function.

Property	Type	Access	Description
`Id`	Integer	RO	Unique identifier assigned to this Embedded MATLAB function to distinguish it from other objects in the model.
`LabelString`	String	RW	Full label for this Embedded MATLAB function (default = `'()'`) including its return, name, and arguments. Equivalent to typing the label for this Embedded MATLAB function in its label text field in the Stateflow Editor.
`Machine`	Machine	RO	Machine that contains this Embedded MATLAB function.
`Name`	String	RW	Name of this Embedded MATLAB function (default = `''`). Equivalent to typing a name for this Embedded MATLAB function into the label text field of the truth table box in the Stateflow Editor. Label syntax is *return* = Name (*arguments*).
`Position`	Rect	RW	Position and size of this Embedded MATLAB function in the Stateflow chart, given in the form of a 1-by-4 array (default is `[0 0 90 60]`) consisting of the following: • (x,y) coordinates for the box's left upper vertex relative to the upper left vertex of the Stateflow Editor workspace • Width and height of the box

`Script`	String	RW	String that holds the lines of code in an Embedded MATLAB function. For example, you can use the property to define code in this way:
			`fcn = Stateflow.EMFunction(c);`
			`str = sprintf('y=proc(x)\n y=x;');`
			`fcn.script = str;`

Property	Type	Access	Description
`Subviewer`	Chart or State	RO	State or chart in which you can graphically view this Embedded MATLAB function.
`Tag`	Any Type	RW	Holds data of any type (default = `[]`) for this Embedded MATLAB function.

Embedded MATLAB Function Methods

Embedded MATLAB Function objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Embedded MATLAB Function Properties" on page 2-32.

Method	Description
`delete`	Delete this Embedded MATLAB function from the Stateflow chart.
`dialog`	Display the properties dialog box of this Embedded MATLAB function.
`disp`	Display the property names and their settings for this Embedded MATLAB function object.
`find`	Find all objects that this Embedded MATLAB function contains that meet the specified criteria.
`fitToView`	Zoom in on this Embedded MATLAB function and highlight it in the Stateflow Editor.
`get`	Return the specified property settings for this Embedded MATLAB functions.
`help`	Display a list of properties for this Embedded MATLAB function with short descriptions.
`methods`	Display all nonglobal methods of this Embedded MATLAB object.
`set`	Set the specified property of this Embedded MATLAB object with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this Embedded MATLAB object.
`view`	Opens this Embedded MATLAB function in the Embedded MATLAB Editor.

Event Properties

Stateflow API objects of type Event have the properties shown below. See also "Event Methods" on page 2-39.

Property	Type	Access	Description
`Debug.Breakpoints.StartBroadcast`	Boolean	RW	If set to `true` (default = `false`), set a debugger breakpoint for the start of the broadcast of this event. Equivalent to selecting the **Start of broadcast** check box in the Event properties dialog box.
`Debug.Breakpoints.EndBroadcast`	Boolean	RW	If set to `true` (default = `false`), set a debugger breakpoint for the end of the broadcast of this event. Equivalent to selecting the **End of broadcast** check box in the Event properties dialog box.
`Description`	String	RW	Description of this event (default = `''`). Equivalent to entering a description in the **Description** field in the Event properties dialog box.
`Document`	String	RW	Document link to this event (default = `''`). Equivalent to entering the **Document Link** field in the Event properties dialog box.
`Id`	Integer	RO	Unique identifier assigned to this event to distinguish it from other objects in the model.
`Machine`	Machine	RO	Machine to which this event belongs.
`Name`	String	RW	Name of this event (default = `event`*n*, where *n* is a counter of events with the name root `event`). Equivalent to entering the name in the **Name** field of the Event properties dialog box.

Property	Type	Access	Description
`Port`	Integer	RO	Port index number for this event (default = 1).
`Scope`	Enum	RW	Scope of this event. Allowed values vary with the object containing this data. The following applies to any event: • `'Local'` The following apply to events for charts only: • `'input'` (**Input from Simulink** in properties dialog box) • `'Output'` (**Output to Simulink** in properties dialog box) The following apply to events for machines only: • `'Imported'` • `'Exported'`
`Tag`	Any Type	RW	Holds data of any type (default = `[]`) for this event.
`Trigger`	Enum	RW	Type of signal that triggers this chart input event. Also the type of trigger associated with this chart output event. Equivalent to the entries for the **Trigger** field in the Event properties dialog box. The following triggers apply to both chart input and output events: • `'Either'` (**Either Edge**)

Property	Type	Access	Description
			• `'Function call'` (**Function Call**) The following triggers apply only to chart input events: • `'Rising'` (**Rising Edge**) • `'Falling'` (**Falling Edge**)

Event Methods

Event objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Event Properties" on page 2-36.

Method	Description
`delete`	Delete this event.
`dialog`	Display the Event properties dialog box.
`disp`	Display the property names and their settings for this Event object.
`get`	Return the specified property settings for this event.
`help`	Display a list of properties for this Event object with short descriptions.
`methods`	Display all nonglobal methods of this Event object.
`set`	Set the specified property of this Event object with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this Event object.
`view`	Display this event in its properties dialog box.

Graphical Function Properties

Stateflow API objects of type Function have the properties shown below. See also "Graphical Function Methods" on page 2-43.

Property	Type	Access	Description
`ArrowSize`	Double	RW	Size of transition arrows coming into this graphical function (default = `8`).
`BadIntersection`	Boolean	RO	If `true`, this function graphically intersects a state, box, Embedded MATLAB function, truth table, or another graphical function.
`Chart`	Chart	RO	Chart object containing this function.
`Debug.Breakpoints.OnDuring`	Boolean	RW	If `true`, sets the during breakpoint for this graphical function. Equivalent to selecting the **Function Call** check box in the properties dialog box for this graphical function.
`Description`	String	RW	Description of this function (default = `''`). Equivalent to entering a description in the **Description** field of the properties dialog box for this graphical function.
`Document`	String	RW	Document link to this function. Equivalent to entering a link in the **Document Link** field of the properties dialog box for this graphical function.
`FontSize`	Double	RW	Size of the (default = `12`) font of the label text for this function. This property overrides the font size set for this function at creation by the `StateFont.Size` property of the containing Chart's object. Equivalent to selecting **Font Size** > **<font size>** in the context menu for this graphical function.

Property	Type	Access	Description
`Id`	Integer	RO	Unique identifier assigned to this function to distinguish it from other objects in the model.
`InlineOption`	String	RW	Determine how generated code for this graphical function appears. Possible settings are: • `'Inline'` — Call to function is replaced by code. • `'Function'` — Function becomes a C function. • `'Auto'` — Determines if the function is inlined or made a function through an internal calculation.
`IsGrouped`	Boolean	RW	If set to `true` (default = `false`), group this function.
`IsSubchart`	Boolean	RW	If set to `true` (default = `false`), make this function a subchart.
`LabelString`	String	RW	Label for this function (default = `'()'`). Equivalent to typing the label for this function in its label text field in the Stateflow Editor.
`Machine`	Machine	RO	Machine that contains this function.
`Name`	String	RW	Name of this function (default = `''`). Equivalent to typing this function's name into the beginning of the label text field after the word `'function'` in the Stateflow Editor.

Property	Type	Access	Description
`Position`	Rect	RW	Position and size of this function's box in the Stateflow chart, given in the form of a 1-by-4 array (default is `[0 0 90 60]`) consisting of the following: • (x,y) coordinates for the box's left upper vertex relative to the upper left vertex of the Stateflow Editor workspace • Width and height of the box
`Subviewer`	Chart or State	RO	State or chart in which this function can be graphically viewed.
`Tag`	Any Type	RW	Holds data of any type (default = `[]`) for this function.

Graphical Function Methods

Function objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Graphical Function Properties" on page 2-40.

Method	Description
`defaultTransitions`	Return the default transitions in this function at the top level of containment.
`delete`	Delete this function from the Stateflow chart.
`dialog`	Display the properties dialog box of this graphical function.
`disp`	Display the property names and their settings for this Function object.
`find`	Find all objects that this graphical function contains that meet the specified criteria.
`fitToView`	Zoom in on this graphical function and highlight it in the Stateflow Editor.
`get`	Return the specified property settings for this function.
`help`	Display a list of properties for this Function object with short descriptions.
`methods`	Display all nonglobal methods of this Function object.
`set`	Set the specified property of this Function object with the specified value.
`sourcedTransitions`	Return all inner and outer transitions whose source is this function.
`struct`	Return and display a MATLAB structure containing the property settings of this Function object.
`view`	Display this function's chart in the Stateflow Editor with this state highlighted.

Junction Properties

Stateflow API objects of type Junction have the properties shown below. See also "Junction Methods" on page 2-45.

Property	Type	Access	Description
`ArrowSize`	Double	RW	Size of transition arrows (default = `8`) coming into this junction.
`Chart`	Chart	RO	Chart that this junction resides in.
`Description`	String	RW	Description of this junction (default = `''`). Equivalent to entering a description in the **Description** field of the Junction properties dialog box.
`Document`	String	RW	Document link to this junction (default = `''`). Equivalent to entering the **Document Link** field of the Junction properties dialog box.
`Id`	Integer	RO	Unique identifier assigned to this junction to distinguish it from other objects in the model.
`Machine`	Machine	RO	Machine containing this junction.
`Position. Center`	Rect	RW	Position of the center of this junction (default = `[10 10]`) relative to the upper left corner of the parent chart or state as an `[x,y]` point array.
`Position. Radius`	Rect	RO	Radius of this junction (default = `10`).
`Subviewer`	Chart or State	RO	State or chart in which this junction can be graphically viewed.
`Tag`	Any Type	RW	Holds data of any type (default = `[]`) for this junction.
`Type`	Enum	RO	Type of this junction. For junctions, can be `'CONNECTIVE'` (default) or `'HISTORY'`

Junction Methods

Junction objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Junction Properties" on page 2-44.

Method	Description
`delete`	Delete this junction from the Stateflow chart.
`dialog`	Display the Junction properties dialog box.
`disp`	Display the property names and their settings for this Junction object.
`fitToView`	Zoom in on this junction and highlight it in the Stateflow Editor.
`get`	Return the specified property settings for this junction.
`help`	Display a list of properties for this Junction object with short descriptions.
`methods`	Display all nonglobal methods of this Junction object.
`set`	Set the specified property of this Junction object with the specified value.
`sourcedTransitions`	Return all inner and outer transitions whose source is this junction.
`struct`	Return and display a MATLAB structure containing the property settings of this Junction object.
`view`	Display this junction's chart in the Stateflow Editor with this junction highlighted.

Machine Properties

Stateflow API objects of type Machine have the properties shown below. See also "Machine Methods" on page 2-50.

Property	Type	Access	Description
`Created`	String	RO	Date of creation of this machine.
`Creator`	String	RW	Creator (default = `'Unknown'`) of this machine.

`Debug.`
`Animation.`
`Enabled`
	Boolean	RW	If set to `true` (default), animation (simulation) is enabled. If `false`, disabled. Equivalent to the **Enabled** or **Disabled** radio button of the Animation section of the Debugger window.

`Debug.`
`Animation.`
`Delay`
	Double	RW	Specify a value to delay (slow down) animation (default value = `0`). Equivalent to the **Delay (sec)** field in the Animation section of the Debugger window.

`Debug.`
`BreakOn.`
`ChartEntry`
	Boolean	RW	If set to `true` (default = `false`), set the chart entry breakpoint for all charts in this machine. Equivalent to the **Chart Entry** check box in the Debugger window.

`Debug.`
`BreakOn.`
`EventBroadcast`
	Boolean	RW	If set to `true` (default = `false`), set the event broadcast breakpoint for all charts in this machine. Equivalent to the **Event Broadcast** check box in the Debugger window.

`Debug.`
`BreakOn.`
`StateEntry`
	Boolean	RW	If set to `true` (default = `false`), set the state entry breakpoint for all charts in this machine. Equivalent to the **State Entry** check box in the Debugger window.

Property	Type	Access	Description
`Debug.DisableAllBreakpoints`	Boolean	RW	If set to `true` (default = `false`), disable the use of all breakpoints in this machine. Equivalent to the **Disable all** check box in the Debugger window.
`Debug.RunTimeCheck.CycleDetection`	Boolean	RW	If set to `true`, check for cyclical behavior errors during a debug session. Equivalent to the **Detect Cycles** check box in the Debugger window.
`Debug.RunTimeCheck.DataRangeChecks`	Boolean	RW	If set to `true` (default), check for data range violations during a debug session. Equivalent to the **Data Range** check box in the Debugger window.
`Debug.RunTimeCheck.StateInconsistencies`	Boolean	RW	If set to `true` (default), check for state inconsistencies during a debug session. Equivalent to the **State Inconsistency** check box in the Debugger window.
`Debug.RunTimeCheck.TransitionConflicts`	Boolean	RW	If set to `true` (default), check for transition conflicts during a debug session. Equivalent to the **Transition Conflict** check box in the Debugger window.
`Description`	String	RW	Description of this state (default = `''`). Equivalent to entering a description in the **Description** field of the properties dialog box for this machine.
`Dirty`	Boolean	RW	If `true` (default), this model has changed since it was opened or saved.

Property	Type	Access	Description
`Document`	String	RW	Document link to this machine (default = `''`). Equivalent to entering the **Document Link** field of the properties dialog box for this machine.
`EnableBitOps`	Boolean	RW	If true, recognize C bitwise operators (~, &, |, ^, >>, and so on) in action language statements for all Stateflow charts in the model and encode them as C bitwise operations.
`FullFileName`	String	RO	Full path name of file (default value = `''`) under which this machine (model) is stored.
`Iced`	Boolean	RO	Equivalent to property `Locked` (default = `false`) except that this property is used internally to lock this model from being changed during activities such as simulation.
`Id`	Integer	RO	Unique identifier assigned to this machine to distinguish it from other objects loaded in memory.
`isLibrary`	Boolean	RO	If `true` (default = `false`), specifies that the current model builds a library and not an application.
`Locked`	Boolean	RW	If set to `true` (default = `false`), prevents user from changing any Stateflow chart in this model.
`Machine`	Machine	RO	A handle to the Machine object for this Machine object, that is, this Machine object.

Property	Type	Access	Description
`Modified`	String	RW	Comment area (default = `''`) for entering date and name of modification to this model.
`Name`	String	RO	Name of this model (default = `'untitled'`) set when saved to disk.
`SfVersion`	Double	RO	Full version number for current Stateflow software. For example, the string `'41112101'` appears for Stateflow software version 4.1.1 and MATLAB software version 12.1. The remaining `'01'` is for internal use.
`Tag`	Any Type	RW	A field you can use to hold data of any type for this machine (default = `[]`).
`Version`	String	RW	Comment string (default = `'none'`) for recording the version of this model.

Machine Methods

Machine objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Machine Properties" on page 2-46.

Method	Description
`dialog`	Display the properties dialog box of this machine.
`disp`	Display the property names and their settings for this Machine object.
`find`	Find all objects that this machine contains that meet the specified criteria. **Note** Do not use the `-depth` switch with the `find` method for a machine object.
`get`	Return the specified property settings for this machine.
`help`	Display a list of properties for this Machine object with short descriptions.
`methods`	Display all nonglobal methods of this Machine object.
`parse`	Parse all the charts in this machine.
`set`	Set the specified property of this Machine object with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this Machine object.

Note Properties

Stateflow API objects of type Note have the properties shown below. See also "Note Methods" on page 2-53.

Property	Type	Access	Description
`Alignment`	Enum	RW	Alignment of text in note box. Can be `'LEFT'` (default), `'CENTER'`, or `'RIGHT'`.
`Chart`	Chart	RO	Chart object containing this note.
`Description`	String	RW	Description of this note (default = `''`). Equivalent to entering a description in the **Description** field of the Note properties dialog box.
`Document`	String	RW	Document link to this note (default = `''`). Equivalent to entering a link in the **Document Link** field of the Note properties dialog box.
`Font.Name`	String	RO	Name of the font (default = `'Helvetica'`) for the text in this note. This property is read-only (RO) and set by the `StateFont.Name` property of the Chart object containing this note.
`Font.Angle`	String	RW	Style of the font for the text in this note. Can be `'ITALIC'` or `'NORMAL'` (default). This property overrides the default style for this note, which is set by the `StateFont.Angle` property of the Chart object containing this note.
`Font.Size`	Double	RW	Size of the font (default = `12`) for the label text for this note. This property overrides the font size set for this note at creation by the `StateFont.Size` property of the containing Chart's object. Equivalent to selecting **Font Size > <font size>** in the context menu for this note.
`Font.Weight`	String	RW	Weight of the font for the label text for this note. Can be `'BOLD'` or `'NORMAL'` (default). This property overrides the default weight for the text in this note, which is set by the `StateFont.Weight` property of the Chart object containing this note.

Property	Type	Access	Description
`Id`	Integer	RO	Unique identifier assigned to this note to distinguish it from other objects in the model.
`Interpretation`	Enum	RW	How the text in this note is interpreted for text processing. Can be `'NORMAL'` (default) or `'TEX'`.
`Machine`	Machine	RO	Machine that contains this note.
`Position`	Rect	RW	Position and size of this note's box in the Stateflow chart, given in the form of a 1-by-4 array (default is `[0 0 25 25]`) consisting of the following: • (x,y) coordinates for the box's left upper vertex relative to the upper left vertex of the Stateflow Editor workspace • Width and height of the box
`Subviewer`	Chart or State	RO	State or chart in which this note can be graphically viewed.
`Tag`	Any Type	RW	Holds data of any type (default = `[]`) for this note.
`Text`	String	RW	Label for this note (default = `'?'`). The text content for this note that you enter directly into the note in the Stateflow Editor or in the **Label** field of the Note properties dialog box.

Note Methods

Note objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Note Properties" on page 2-51.

Method	Description
`delete`	Delete this note from the Stateflow chart.
`dialog`	Display the Note properties dialog box.
`disp`	Display the property names and their settings for this Note object.
`fitToView`	Zoom in on this note and highlight it in the Stateflow Editor.
`get`	Return the specified property settings for this note.
`help`	Display a list of properties for this Note object with short descriptions.
`methods`	Display all nonglobal methods of this Note object.
`set`	Set the specified property of this Note object with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this Note object.
`view`	Display this note's chart in the Stateflow Editor with this note highlighted.

Root Methods

The Root object has the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

Method	Description
`find`	Find all objects that this Root object contains that meet the specified criteria.
`get`	Return the specified property settings for the Root object.
`help`	Display a list of properties for the Root object with short descriptions.
`methods`	Display all nonglobal methods of this Root object.
`set`	Set the specified property of this Root object with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this Root object.

Simulink Function Properties

Stateflow API objects of type Simulink Function have the properties shown below. See also "Simulink Function Methods" on page 2-57.

Property	Type	Access	Description
`ArrowSize`	Double	RW	Size of transition arrows coming into this Simulink function (default = `8`).
`BadIntersection`	Boolean	RO	If `true`, this function graphically intersects a state, box, Embedded MATLAB function, graphical function, truth table, or another Simulink function.
`Chart`	Chart	RO	Chart object containing this function.
`Description`	String	RW	Description of this function (default = `''`).
`Document`	String	RW	Document link to this note.
`FontSize`	Double	RW	Size of the font (default = `12`) of the label text for this function. This property overrides the font size set for this function at creation by the `StateFont.Size` property of the containing Chart's object. Equivalent to selecting **Font Size** > **<font size>** in the context menu for this Simulink function.
`Id`	Integer	RO	Unique identifier assigned to this function to distinguish it from other objects in the model.
`LabelString`	String	RW	Label for this function. Equivalent to typing the label for this function in its label text field in the Stateflow Editor.
`Machine`	Machine	RO	Machine that contains this function.

Property	Type	Access	Description
`Name`	String	RW	Name of this function (default = `'simfcn'`). Equivalent to typing the name of this function in its label text field in the Stateflow Editor.
`Position`	Rect	RW	Position and size of this function box in the Stateflow chart, given in the form of a 1-by-4 array (default is `[0 0 90 60]`) that consists of the following: • (x,y) coordinates for the box's left upper vertex relative to the upper left vertex of the Stateflow Editor workspace • Width and height of the box
`Subviewer`	Chart	RO	Chart in which this function can be graphically viewed.
`Tag`	Any Type	RW	A field you can use to hold data of any type for this function.

Simulink Function Methods

Simulink Function objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Simulink Function Properties" on page 2-55.

Method	Description
`delete`	Delete this function from the Stateflow chart.
`disp`	Display the property names and their settings for this Simulink Function object.
`find`	Find all objects that this Simulink function contains that meet the specified criteria.
`fitToView`	Zoom in on this Simulink function and highlight it in the Stateflow Editor.
`get`	Return the specified property settings for this function.
`help`	Display a list of properties for this Simulink Function object with short descriptions.
`methods`	Display all nonglobal methods of this Simulink Function object.
`set`	Set the specified property of this Simulink Function object with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this Simulink Function object.
`view`	Display the contents of the subsystem inside this Simulink function.

State Properties

Stateflow API objects of type State have the properties shown below. See also "State Methods" on page 2-62.

Property	Type	Access	Description
`ArrowSize`	Double	RW	Size of transition arrows coming into this state (default = 8). Equivalent to selecting **Arrowhead Size** from the context menu for this state.
`BadIntersection`	Boolean	RO	If `true`, this state graphically intersects a box, Embedded MATLAB function, graphical function, truth table, or another state.
`Chart`	Chart	RO	Chart object containing this state.
`Debug.Breakpoints.OnDuring`	Boolean	RW	If set to `true` (default = `false`), set the state during breakpoint for this chart. Equivalent to selecting the **State During** check box in the State properties dialog box.
`Debug.Breakpoints.OnEntry`	Boolean	RW	If set to `true` (default = `false`), set the state entry breakpoint for this chart. Equivalent to selecting the **State Entry** check box in the State properties dialog box.
`Debug.Breakpoints.OnExit`	Boolean	RW	If set to `true` (default = `false`), set the state exit breakpoint for this chart. Equivalent to selecting the **State Exit** check box in the State properties dialog box.
`Decomposition`	Enum	RW	Set this property to `'EXCLUSIVE_OR'` (default) to specify exclusive (OR) decomposition for the states at the first level of containment in this state. Set to `'PARALLEL_AND'` to specify parallel (AND) decomposition for these states. Equivalent to the **Decomposition** selection in the context menu for the state.

Property	Type	Access	Description
`Description`	String	RW	Description of this state (default = `''`). Equivalent to entering a description in the **Description** field of the State properties dialog box.
`Document`	String	RW	Document link to this state (default = `''`). Equivalent to entering a link in the **Document Link** field of the State properties dialog box.
`ExecutionOrder`	Integer	RW	Specifies the order in which this state wakes up for parallel (AND) decomposition. Equivalent to the **Execution Order** selection in the context menu for the state. The `UserSpecifiedStateTransitionExecutionOrder` property of the parent chart must be true. Otherwise, this property does not apply.
`FontSize`	Double	RW	Size of the font (default = `12`) for the label text for this state. This property overrides the font size set for this state at creation by the `StateFont.Size` property of the containing Chart's object. Equivalent to selecting **Font Size** > **<font size>** in the context menu for this state.
`HasOutputData`	Boolean	RW	If set to true (default = `false`), create a data output port on the Stateflow block for this state with its activity status. If the state is active, the output value is 1. If the state is inactive, the output is 0. Equivalent to selecting the **Output State Activity** check box in the State properties dialog box.
`Id`	Integer	RO	Unique identifier assigned to this state to distinguish it from other objects in the model.

Property	Type	Access	Description
`InlineOption`	String	RW	Determine how generated code for this state appears. Possible settings are: • `'Inline'` — Call to function is replaced by code. • `'Function'` — Function becomes a C function. • `'Auto'` — Determines if the function is inlined or made a function through an internal calculation. For guidelines on controlling inlining of state functions, see "Controlling Inlining of State Functions in Generated Code" in the Stateflow and Stateflow Coder User's Guide.
`IsGrouped`	Boolean	RW	If set to `true` (default = `false`), group this state. Nothing can change inside a grouped state. This property is also useful for copying states to a new location. See "Copying by Grouping (Recommended)" on page 1-33.
`IsSubchart`	Boolean	RW	If set to `true` (default = `false`), make this state a subchart.
`LabelString`	String	RW	Label for this state (default = `'?'`). Equivalent to typing the label for this state in its label text field in the Stateflow Editor.
`Machine`	Machine	RO	Machine containing this state.

Property	Type	Access	Description
`Name`	String	RW	Name of this state (default = `''`). Equivalent to typing this state's name into the beginning of the label text field for this state in the Stateflow Editor. `Name` is separated from the remainder of this state's label text by a forward slash (/) character.
`Position`	Rect	RW	Position and size of this state's box in the Stateflow chart, given in the form of a 1-by-4 array (default is `[0 0 90 60]`) consisting of the following: • (x,y) coordinates for the box's left upper vertex relative to the upper left vertex of the Stateflow Editor workspace • Width and height of the box
`Subviewer`	Chart or State	RO	Chart or state in which you can graphically view this state.
`Tag`	Any Type	RW	Holds data of any type (default = `[]`) for this state.
`TestPoint`	Integer	RW	If set to `1` (default = `0`), sets this state as a Stateflow test point. See "Monitoring Test Points in Stateflow Charts" in the Stateflow and Stateflow Coder User's Guide.
`Type`	Enum	RO	Type of this state (`default = 'OR'`). Can be `'OR'` (exclusive) or `'AND'` (parallel). The type of this state is determined by the parent's `Decomposition` property.

State Methods

State objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "State Properties" on page 2-58.

Method	Description
`defaultTransitions`	Return the default transitions in this state at the top level of containment.
`delete`	Delete this state.
`dialog`	Display the State properties dialog box.
`disp`	Display the property names and their settings for this State object.
`find`	Find all objects that this state contains that meet the specified criteria.
`fitToView`	Zoom in on this state and highlight it in the Stateflow Editor.
`get`	Return the specified property settings for this state.
`help`	Display a list of properties for this State object with short descriptions.
`innerTransitions`	Return the inner transitions that originate with this state and terminate on a contained object.
`methods`	Display all nonglobal methods of this State object.
`outerTransitions`	Return an array of transitions that exit the outer edge of this state and terminate on an object outside the containment of this state.
`outputData`	Output the activity status of this state to the Simulink base workspace via a data output port on the Stateflow block of this state.
`set`	Set the specified property of this State object with the specified value.
`sourcedTransitions`	Return all inner and outer transitions whose source is this state.

Method	Description
`struct`	Return and display a MATLAB structure containing the property settings of this State object.
`view`	Display this state's chart in the Stateflow Editor with this state highlighted.

Target Properties

Stateflow API objects of type Target have the properties shown below. See also “Target Methods” on page 2-69.

Property	Type	Access	Description
`ApplyToAllLibs`	Boolean	RW	If set to `true` (default), use settings in this custom target for all libraries. Equivalent to selecting the **Use these custom code settings for all libraries** check box in the Custom Target dialog box.
`CodeFlagsInfo`	Array	RO	A MATLAB vector of structures containing information on the code flag settings for this custom target. See “CodeFlagsInfo Property of Targets” on page 2-66 for more information.
`CodegenDirectory`	String	RW	Folder to store generated code for a custom target (default = `''`). Equivalent to the entry in the **Generated Code Directory** field of the Custom Target dialog box.
`CustomCode`	String	RW	Custom code included at the top of the generated header file for a custom target (default = `''`). Equivalent to the entry in the **Include Code** field of the Custom Target dialog box.
`Description`	String	RW	Description of this custom target (default = `''`). Equivalent to entering a description in the **Description** field of the Custom Target dialog box.

Property	Type	Access	Description
`Document`	String	RW	Document link to this custom target (default = `''`). Equivalent to entering the **Document Link** field of the Custom Target dialog box.
`Id`	Integer	RO	Unique identifier assigned to this Target object to distinguish it from other objects loaded in memory.
`Machine`	Machine	RO	Stateflow machine containing this custom target.
`Name`	String	RW	Name of this custom target (default = `'untitled'`). Equivalent to naming or renaming this target in the Model Explorer.
`ReservedNames`	String	RW	Comma- or space-separated list of names to not use in Stateflow generated code for a custom target. Equivalent to the entry in the **Reserved Names** field of the Custom Target dialog box. **Note** This property applies only to nonlibrary models. You cannot specify this property for library models.
`Tag`	Any Type	RW	Holds data of any type (default = `[]`) for this target.

Property	Type	Access	Description
`UserIncludeDirs`	String	RW	Space-separated list of custom include folder paths for a custom target (default = `''`). Equivalent to the entry in the **Include Paths** field of the Custom Target dialog box. **Note** If your list includes any Windows® path strings that contain spaces, each instance must be enclosed in double quotes within the argument string, for example, `'C:\Project "C:\Custom Files"'`
`UserLibraries`	String	RW	Space-separated list of custom libraries for a custom target (default = `''`). Equivalent to the entry in the **Libraries** field of the Custom Target dialog box.
`UserSources`	String	RW	Space-separated list of custom source files for a custom target (default = `''`). Equivalent to the entry in the **Source Files** field of the Custom Target dialog box.

CodeFlagsInfo Property of Targets

The `CodeFlagsInfo` property of a Target object is a read-only MATLAB vector of structures containing information on the code flag settings for a custom target. Each element in the vector has the following MATLAB structure of information about a particular code flag:

Element	Type	Description
`name`	String	Short name for this flag
`type`	String	The type of the code flag
`description`	String	A description of this code flag
`defaultValue`	Boolean	The default value of this code flag upon creation of its target
`visible`	Boolean	Whether or not this flag is visible
`enable`	Boolean	Whether or not to enable this flag
`value`	Boolean	The value of the flag

The first element of each structure is a shorthand name for the individual flag that you set in the Custom Target dialog box. For example, the name `'comments'` actually refers to the dialog box setting **User Comments in generated code**. While the `CodeFlagsInfo` property is informational only, you can use these shorthand flag names in the methods `getCodeFlag` and `setCodeFlag` to access and change the values of a flag. (See Chapter 4, "API Method Reference" for more information.)

The names of possible code flags in the `CodeFlagsInfo` property and the name of the flag as it appears in the **General** pane of the Custom Target dialog box are as follows:

Name in CodeFlagsInfo	Name in Custom Target Dialog Box	Default Value
`comments`	**User Comments in generated code**	Enabled
`autocomments`	**Auto-generated Comments in generated code**	Disabled
`emitdescriptions`	**State/Transition Descriptions in generated code**	Enabled
`statebitsets`	**Use bitsets for storing state configuration**	Enabled
`databitsets`	**Use bitsets for storing boolean data**	Enabled

Name in CodeFlagsInfo	Name in Custom Target Dialog Box	Default Value
`emitlogicalops`	**Compact nested if-else using logical AND/OR operators**	Enabled
`elseifdetection`	**Recognize if-elseif-else in nested if-else statements**	Enabled
`constantfolding`	**Replace constant expressions by a single constant**	Enabled
`redundantloadelimination`	**Minimize array reads using temporary variables**	Disabled
`exportcharts`	**Use chart names with no mangling**	Disabled
`ioformat`	**I/O data format:** Can be one of these values: • 0 = **Use global input/output data** • 1 = **Pack input/output data into structures**	0
`initializer`	**Generate chart initializer function**	Disabled
`multiinstanced`	**Multi-instance capable code**	Disabled

Target Methods

Target objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Target Properties" on page 2-64.

Method	Description
`build`	Build this custom target incrementally only for those portions of the target's charts that have changed since the last build.
`delete`	Delete this custom target.
`dialog`	Display the Custom Target dialog box.
`disp`	Display the property names and their settings for this Target object.
`get`	Return the specified property settings for this custom target.
`getCodeFlag`	Return the value of the specified code flag for this custom target.
`help`	Display a list of properties for this Target object with short descriptions.
`make`	Compile this custom target incrementally only for those portions of generated code that have changed since the last compilation.
`methods`	Display all nonglobal methods of this Target object.
`rebuildAll`	Completely rebuild this custom target.
`regenerateAll`	Completely regenerate code for this custom target.
`set`	Set the specified property of this Target object with the specified value.
`setCodeFlag`	Set the specified code flag for this custom target with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this Target object.
`view`	Display this custom target in the Model Explorer.

Transition Properties

Stateflow API objects of type Transition have the properties shown below. See also "Transition Methods" on page 2-74.

Property	Type	Access	Description
`ArrowSize`	Double	RW	Size of the arrow (default = `10`) for this transition.
`Chart`	Chart	RO	Stateflow chart object containing this transition.
`Debug.Breakpoints.WhenTested`	Boolean	RW	If set to `true` (default = `false`), set a debugging breakpoint to occur when this transition is tested to see whether it is a valid transition or not. Equivalent to selecting the **When Tested** check box in the properties dialog box of this transition.
`Debug.Breakpoints.WhenValid`	Boolean	RW	If set to `true` (default = `false`), set a debugging breakpoint to occur when this transition has tested as valid. Equivalent to selecting the **When Valid** check box in the properties dialog box of this transition.
`Description`	String	RW	Description of this transition (default = `''`). Equivalent to entering a description in the **Description** field of the properties dialog box for this transition.
`Destination`	State or Junction	RW	Destination state or junction (default = `[]`) of this transition. Assign `Destination` the destination object for this transition. You can also use the property `Destination` to detach the destination end of a transition, through the command `t.Destination = []` where `t` is the Transition object.
`DestinationOClock`	Double	RW	Location of transition destination connection on state (default = `0`). Varies from 0 to 12 for full clock cycle location. Its value is taken as modulus 12 of its assigned value.

Property	Type	Access	Description
`Document`	String	RW	Document link to this transition (default = `''`). Equivalent to entering the **Document Link** field of the properties dialog box for this transition.
`DrawStyle`	Enum	RW	Drawing style for this transition. Set to `'SMART'` (default) for smart transitions or `'STATIC'` for static transitions. Equivalent to selecting **Smart** from the context menu for this transition to toggle between settings. **Note** Transition must be connected to effect a change in the `DrawStyle` property. Otherwise, an error occurs.
`ExecutionOrder`	Integer	RW	Specifies the number for this transition in the execution order for its source (see "Evaluation Order for Outgoing Transitions" in the Stateflow and Stateflow Coder User's Guide). The `UserSpecifiedStateTransitionExecutionOrder` property of the parent chart has to be set to `true`, otherwise, this transition property will be ignored. `ExecutionOrder` has to be an integer between 1 and *`m`*, where *`m`* is the total number of transitions originating from the source.
`FontSize`	Double	RW	Size of the font (default = `12`) for the label text for this box. This property overrides the default size for this box, which is set by the `TransitionFont.Size` property of the Chart object containing this box. Equivalent to selecting **Font Size** > **<font size>** in the context menu for this box.
`Id`	Integer	RO	Unique identifier assigned to this transition to distinguish it from other objects in the model.

Property	Type	Access	Description
`LabelPosition`	Rect	RW	Position and size of this note's box in the Stateflow chart, given in the form of a 1-by-4 array (default is `[0 0 8 14]`) consisting of the following: • (x,y) coordinates for the box's left upper vertex relative to the upper left vertex of the Stateflow Editor workspace • Width and height of the box
`LabelString`	String	RW	Label for this transition (default = `'?'`). Equivalent to typing the label for this transition in its label text field in the Stateflow Editor.
`Machine`	Machine	RO	Machine containing this transition.
`MidPoint`	Rect	RW	Position of the midpoint of this transition relative to the upper left corner of the Stateflow Editor workspace in an `[x y]` point array (default = `[0 0]`).
`Source`	State or Junction	RW	Source state or junction of this transition (default = `[]`). Assign `Source` the source object for this transition. You can also use the property `Source` to detach the source end of a transition, through the command `t.Source = []` where `t` is the Transition object.
`SourceEndPoint`	Rect	RO*	`[x y]` spatial coordinates for the endpoint of a transition (default = `[2 2]`). This property is RW (read/write) only for default transitions. For all other transitions it is RO (read-only).
`SourceOClock`	Double	RW	Location of transition source connection on state (default = `0`). Varies from 0 to 12 for full clock cycle location. The value taken for this property is the modulus 12 of the entered value.

Property	Type	Access	Description
`Subviewer`	Chart or State	RO	State or chart in which this transition can be graphically viewed.
`Tag`	Any Type	RW	Holds data of any type (default = `[]`) for this transition.

Transition Methods

Transition objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Transition Properties" on page 2-70.

Method	Description
`delete`	Delete this transition from the Stateflow chart.
`dialog`	Display the properties dialog box of this transition.
`disp`	Display the property names and their settings for this Transition object.
`fitToView`	Zoom in on this transition and highlight it in the Stateflow Editor.
`get`	Return the specified property settings for this transition.
`help`	Display a list of properties for this Transition object with short descriptions.
`methods`	Display all nonglobal methods of this Transition object.
`set`	Set the specified property of this Transition object with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this Transition object.
`view`	Display this transition's chart in the Stateflow Editor with this transition highlighted.

Truth Table Properties

Stateflow API objects of type TruthTable have the properties shown below. See also "Truth Table Methods" on page 2-78.

Property	Type	Access	Description
`ActionTable`	Cell Array	RW	A cell array of strings containing the contents of the Action Table for this truth table.
`ArrowSize`	Double	RW	Size of transition arrows coming into the truth table function in the Stateflow chart (default = `8`).
`BadIntersection`	Boolean	RO	If `true`, this truth table graphically intersects a state, box, Embedded MATLAB function, graphical function, or another truth table.
`Chart`	Chart	RO	Chart object containing this truth table.
`ConditionTable`	Cell Array	RW	A cell array of strings containing the contents of the Condition Table for this truth table, including the **Actions** row.
`Debug.Breakpoints.OnDuring`	Boolean	RW	If `true`, sets the during breakpoint for this truth table. Equivalent to selecting the **Function Call** check box in the Truth Table properties dialog box.
`Description`	String	RW	Description of this truth table (default = `''`). Equivalent to entering a description in the **Description** field of the Truth Table properties dialog box.
`Document`	String	RW	Document link to this truth table. Equivalent to entering a link in the **Document Link** field of the Truth Table properties dialog box.

Property	Type	Access	Description
`FontSize`	Double	RW	Size of the (default = `12`) font of the label text for this truth table. This property overrides the font size set for this truth table at creation by the `StateFont.Size` property of the containing Chart's object. Equivalent to selecting **Font Size** > *font size* in the context menu for this truth table.
`Id`	Integer	RO	Unique identifier assigned to this truth table to distinguish it from other objects in the model.
`LabelString`	String	RW	Full label for this truth table (default = '`()`') including its return, name, and arguments. Equivalent to typing the label for this truth table in its label text field in the Stateflow Editor.
`Machine`	Machine	RO	Machine that contains this truth table.
`Name`	String	RW	Name of this truth table (default = `''`). Equivalent to typing a name for this truth table into the label text field of the truth table box in the Stateflow Editor. Label syntax is *return* = Name (*arguments*).
`OverSpecDiagnostic`	String	RW	Interprets the error diagnosis of this truth table as overspecified according to the possible values 'Error', 'Warning', or 'None'. In the Truth Table Editor, the value of this property is assigned by selecting **Overspecified** from the **Diagnostics** menu item and then selecting one of the three values.

Property	Type	Access	Description
`Position`	Rect	RW	Position and size of this truth table's box in the Stateflow chart, given in the form of a 1-by-4 array (default is `[0 0 90 60]`) consisting of the following: • (x,y) coordinates for the box's left upper vertex relative to the upper left vertex of the Stateflow Editor workspace • Width and height of the box
`Subviewer`	Chart or State	RO	State or chart in which this truth table can be graphically viewed.
`Tag`	Any Type	RW	Holds data of any type (default = `[]`) for this truth table.
`UnderSpecDiagnostic`	String	RW	Interprets the error diagnosis of this truth table as underspecified according to the possible values 'Error', 'Warning', or 'None'. In the Truth Table Editor, the value of this property is assigned by selecting **Underspecified** from the **Diagnostics** menu item and then selecting one of the three values.

Truth Table Methods

Truth table objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Truth Table Properties" on page 2-75.

Method	Description
`delete`	Delete this truth table from the Stateflow chart.
`dialog`	Display the Truth Table properties dialog box.
`disp`	Display the property names and their settings for this truth table object.
`find`	Find all objects that this graphical truth table contains that meet the specified criteria.
`fitToView`	Zoom in on this truth table and highlight it in the Stateflow Editor.
`get`	Return the specified property settings for this truth table.
`help`	Display a list of properties for this truth table object with short descriptions.
`methods`	Display all nonglobal methods of this truth table object.
`set`	Set the specified property of this truth table object with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this truth table object.
`view`	Display this truth table's chart in the Stateflow Editor with this truth table highlighted.

Truth Table Chart Properties

Stateflow API objects of type TruthTableChart have the properties shown below. See also “Truth Table Chart Methods” on page 2-82.

Property	Type	Access	Description
`ActionTable`	Cell Array	RW	A cell array of strings containing the contents of the Action Table for this truth table block.
`ChartUpdate`	Enum	RW	Activation method of this chart. Can be 'INHERITED' (default), 'DISCRETE', or 'CONTINUOUS'.
`ConditionTable`	Cell Array	RW	A cell array of strings containing the contents of the Condition Table for this truth table block, including the **Actions** row.
`Description`	String	RW	Description of this truth table block (default = `''`). Equivalent to entering a description in the **Description** field of the properties dialog box for this truth table block.
`Dirty`	Boolean	RW	If set to true (default = false), this chart has changed since being opened or saved.
`Document`	String	RW	Document link to this truth table block.
`Iced`	Boolean	RO	Equivalent to property Locked (default = false) except that this property is used internally to lock this block from change during activities such as simulation.
`Id`	Integer	RO	Unique identifier assigned to this truth table block to distinguish it from other objects in the model.

Property	Type	Access	Description
`InputFimath`	embedded.fimath object	RW	The embedded.fimath object that will be associated with inputs from Simulink blocks.
`LabelString`	String	RW	Full label for this truth table (default = '()') including its return, name, and arguments. Equivalent to typing the label for this truth table in its label text field in the Stateflow Editor.
`Locked`	Boolean	RW	If set to true (default = false), mark this block as read-only and prohibit any write operations on it.
`Machine`	Machine	RO	Machine that contains this truth table block.
`Name`	String	RW	Name of this truth table block. (default = ''). Equivalent to typing a name for this truth table into the label text field of the truth table box in the Stateflow Editor. Label syntax is *return* = Name (*arguments*).
`OverSpecDiagnostic`	String	RW	Interprets the error diagnosis of this truth table as overspecified according to the possible values 'Error', 'Warning', or 'None'. In the Truth Table Editor, the value of this property is assigned by selecting **Overspecified** from the **Settings** menu item and then selecting one of the three values.
`Path`	String	RW	Path to the block.
`SampleTime`	String	RW	Sample time for activating this chart (default = ' ').

Property	Type	Access	Description
`Tag`	Any Type	RW	Holds data of any type (default = [ ]) for this truth table block.
`TreatInheritedIntegersAs`	String	RW	Determines how inherited integer signals are treated in Embedded MATLAB functions.The two choices are `Integers` and `Fixed-point`. A Simulink model does not distinguish between a fixed-point signal with zero fraction length and an integer signal. However, the MATLAB workspace has two different classes for these two kinds of data: Integers (uint8, int16, etc...) and embedded.fi. You can specify the type for any given input signal to be either `Integer` or `fixed-point` and override this default.
`UnderSpecDiagnostic`	String	RW	Interprets the error diagnosis of this truth table as underspecified according to the possible values 'Error', 'Warning', or 'None'. In the Truth Table Editor, the value of this property is assigned by selecting **Underspecified** from the **Settings** menu item and then selecting one of the three values.

Truth Table Chart Methods

Truth Table Chart objects have the methods in the table below. For details on each method, see Chapter 4, "API Method Reference".

See also "Truth Table Chart Properties" on page 2-79.

Method	Description
`defaultTransitions`	Return the default transitions in this object at the top level of containment.
`delete`	Delete this truth table from the Stateflow chart.
`dialog`	Display the properties dialog box of this truth table.
`disp`	Display the property names and their settings for this truth table object.
`find`	Find all objects that this graphical truth table contains that meet the specified criteria.
`get`	Return the specified property settings for this truth table.
`help`	Display a list of properties for this truth table object with short descriptions.
`methods`	Display all nonglobal methods of this truth table object.
`set`	Set the specified property of this truth table object with the specified value.
`struct`	Return and display a MATLAB structure containing the property settings of this truth table object.
`view`	Display this truth table's chart in the Stateflow Editor with this truth table highlighted.

3

API Object Properties and Methods

Reference Table Column Descriptions

Reference tables for Stateflow API properties and methods have these columns:

- **Name** — The name for the property or method. Each property or method has a name that you use in dot notation along with a Stateflow object to set or obtain the property's value or call the method.
- **Type** — A data type for the property. Some types are other Stateflow API objects, such as the `Machine` property, which is the Machine object that contains this object.
- **Access** — An access type for the property. Properties that are listed as RW (read/write) can be read and changed. For example, the `Name` and `Description` properties of particular objects are RW. However, some properties are RO (read-only) because they are set by the MATLAB workspace itself.
- **Description** — A description for the property or method. For some properties, the equivalent GUI operations for setting it are also given.
- **Objects** — The types of objects that have this property or method. The object types are listed as follows: Root (R), Machine (M), Chart (C), State (S), Box (B), Graphical Function (F), Truth Table (TT), Embedded MATLAB Function (EM), Simulink Function (SLF), Note (N), Transition (T), Junction (J), Event (E), Data (D), Target (X), Editor (ED), and Clipboard (CB).

Access Methods

The following methods find, get, and set objects and their properties.

Method	Description	Objects
`defaultTransitions`	Return the default transitions in this chart at the top level of containment.	C S B F
`disp`	Display the property names and their settings for this object.	C S B F N T J D E X TT EM SLF
`find`	Find all objects that this object contains that meet the criteria specified by the arguments.	All
`get`	Return the specified property settings for this object.	All
`help`	Display a list of properties for this object with short descriptions. Used with all objects except the Root and Machine object.	All
`innerTransitions`	Return the inner transitions that originate with this object and terminate on a contained object.	S B
`methods`	Return the methods of this object.	All
`outerTransitions`	Return an array of transitions that exit the outer edge of this object and terminate on an object outside the containment of this object.	S B
`set`	Set the specified property of this object with a specified value. Used with all objects except the Root object.	All
`sourcedTransitions`	Return all inner and outer transitions whose source is this object.	S B F J
`struct`	Return a MATLAB structure containing the property settings of this object.	C S B F N T J D E X TT EM SLF

Code Generation and Target Building

In this section...

Code Generation and Build Methods

The following methods control parsing, code generation, and building of custom targets.

Method	Description	Objects
`build`	Build this custom target only for those portions of the target's charts that have changed since the last build (i.e., incrementally). See also the methods `rebuildAll`, `generate`, `rebuildAll`, and `make`.	X
`generate`	Generate code for this custom target only for those portions of this target's charts that have changed since the last code generation (i.e., incrementally). See also the methods `build`, `rebuildAll`, `regenerateAll`, and `make`.	X
`getCodeFlag`	Return the value of the specified code flag for this custom target.	X
`make`	Compile this custom target incrementally, only for those portions of this target's charts that have changed since the last compilation. See also the methods `build`, `rebuildAll`, `generate`, and `regenerateAll`.	X
`parse`	Parses all the charts in this machine (model) or just this chart.	M C

Method	Description	Objects
`rebuildAll`	Completely rebuild this custom target. See also the methods `build`, `generate`, `regenerateAll`, and `make`.	X
`regenerateAll`	Completely regenerate code for this custom target. See also the methods `build`, `rebuildAll`, `generate`, and `make`.	X
`setCodeFlag`	Set the value of the specified code flag for this custom target.	X

Code Generation Properties

The following properties control the code generated from the Stateflow charts in a model.

Property	Type	Access	Description	Objects
`ApplyToAllLibs`	Boolean	RW	If set to true, use settings in this custom target for all libraries. Equivalent to selecting the **Use these custom code settings for all libraries** check box in the Custom Target dialog box.	X

`CodeFlagsInfo`	Array	RO	A MATLAB vector of structures containing information on code flag settings for this custom target. Each element in the vector is a MATLAB structure with information about a specific code flag. Each flag corresponds to a selection in the properties dialog box for this target.
			If you want to see information about the first flag for a Target object `x`, use these commands:
			`cfi = x.CodeFlagsInfo`
			`disp(cfi(1))`
				X

Property	Type	Access	Description	Objects
			If you want to see the names of all the flags, use this command: `cfi.name` The `Name` member of the `CodeFlagsInfo` structure is shorthand for a longer expression in the properties dialog box. For example, the name `'comments'` refers to the dialog box setting **User Comments in generated code**. You use the name of a code flag to get and set the code flag value with the methods `getCodeFlag` and `setCodeFlag`. Changing the vector returned by `CodeFlagsInfo` does not change an actual flag.	
`EnableBitOps`	Boolean	RW	If set to true, enable C-like bit operations in generated code for this chart. Equivalent to selecting the **Enable C-bit operations** check box in the Chart properties dialog box.	C

Custom Code Properties

The following properties control the custom code that you include with a Stateflow chart.

Property	Type	Access	Description	Objects
`CodegenDirectory`	String	RW	Folder to store generated code for a custom target (default = `''`). Equivalent to the entry in the **Generated Code Directory** field of the custom code settings in the Custom Target dialog box.	X
`CustomCode`	String	RW	Custom code included at the top of the generated header file for a custom target (default = `''`). Equivalent to the entry in the **Include Code** field of the custom code settings in the Custom Target dialog box.	X
`UserIncludeDirs`	String	RW	Space-separated list of custom include folder paths for a custom target (default = `''`). Equivalent to the entry in the **Include Paths** field of the custom code settings in the Custom Target dialog box. **Note** If your list includes any Windows path strings that contain spaces, each instance must be enclosed in double quotes within the argument string, for example, `'C:\Project "C:\Custom Files"'`	X
`UserLibraries`	String	RW	Space-separated list of custom libraries for a custom target (default = `''`). Equivalent to the entry in the **Libraries** field of the custom code settings in the Custom Target dialog box.	X

Property	Type	Access	Description	Objects
`UserSources`	String	RW	Space-separated list of custom source files for a custom target (default = `''`). Equivalent to the entry in the **Source Files** field of the custom code settings in the Custom Target dialog box.	X
`ReservedNames`	String	RW	Comma- or space-separated list of names to not use in Stateflow generated code for a custom target. Equivalent to the entry in the **Reserved Names** field of the custom code settings in the Custom Target dialog box. **Note** This property applies only to nonlibrary models. You cannot specify this property for library models.	X

Containment

The following properties control how one Stateflow object contains another Stateflow object.

Property	Type	Access	Description	Objects
`Chart`	Chart	RO	Chart object containing this object.	S B F N T J TT EM SLF
`Decomposition`	Enum	RW	Set this property to `'EXCLUSIVE_OR'` to specify exclusive (OR) decomposition for the states at the first level of containment in this chart or state. Set to `'PARALLEL_AND'` to specify parallel (AND) decomposition for these states. Equivalent to selecting the **Decomposition** in the context menu for the chart or state.	C S
`IsGrouped`	Boolean	RW	If set to true, group this object. Nothing is allowed to change inside a grouped object. You must first ungroup the object before you can change its contents. This property is also useful for copying states and their contents to a new location. See "Copying by Grouping (Recommended)" on page 1-33.	S B F
`IsSubchart`	Boolean	RW	If set to true, makes this state, box, or graphical function a subchart.	S B F
`Machine`	Machine	RO	Machine that contains this object. A machine object contains all of the Chart objects in a Model.	C S B F N T J D E X TT EM SLF

Creating and Deleting Objects

Use the following methods to create and delete Stateflow objects.

Method	Description	Objects
`copy`	Copy the specified array of objects to the clipboard for pasting. See also the `pasteTo` method.	CB
`delete`	Delete this object.	All but R M C CB ED
`pasteTo`	Paste the objects in the Clipboard to the specified container object. See also `copy` method.	CB
`Stateflow.Box`	Create a box for a parent chart, state, box, or function.	NA
`Stateflow.Data`	Create a data for a parent machine, chart, state, box, or function.	NA
`Stateflow.EMFunction`	Create an Embedded MATLAB function for a parent chart or state.	NA
`Stateflow.Event`	Create an event for a parent machine, chart, state, box, or function.	NA
`Stateflow.Function`	Create a graphical function for a parent chart, state, box, or function.	NA
`Stateflow.Junction`	Create a junction for a parent chart, state, box, or function.	NA
`Stateflow.Note`	Create a note for a parent chart or state.	NA
`Stateflow.SLFunction`	Create a Simulink function for a parent chart or state.	NA
`Stateflow.State`	Create a state for a parent chart, state, box, or function.	NA
`Stateflow.Target`	Create a custom target for a parent machine.	NA

Method	Description	Objects
`Stateflow.Transition`	Create a transition for a parent chart, state, box, or function.	NA
`Stateflow.TruthTable`	Create a truth table function for a parent chart or state.	NA

Data Definition Properties

The following properties control the type, size, and value of data in Stateflow charts.

Property	Type	Access	Description	Objects
`DataType`	Enum	RW	Data type of this data. Can have one of the following possible values: `'boolean'`, `'uint8'`, `'int8'`, `'uint16'`, `'int16'`, `'uint32'`, `'int32'`, `'single'`, `'double'` and `'fixpt'`. Equivalent to an entry in the **Type** column for this data in the Model Explorer or the **Type** field in the Data properties dialog box.	D
`FixptType.Bias`	Double	RW	The Bias value for this fixed-point type.	D
`FixptType.FractionalSlope`	Double	RW	The Fractional Slope value for this fixed-point type.	D
`FixptType.RadixPoint`	Integer	RW	The power of two specifying the binary point location for this fixed-point type.	D
`FixptType.BaseType`	Enum	RW	The size and sign of the base for the quantized integer, Q, of this fixed-point type.	D
`ParsedInfo.Array.Size`	Integer	RO	Numeric equivalent of string Data property `Props.Array.Size`.	D
`ParsedInfo.Array.FirstIndex`	Integer	RO	Numeric equivalent of string Data property `Props.Range.FirstIndex`.	D
`ParsedInfo.InitialValue`	Double	RO	Numeric equivalent of string Data property `Props.InitialValue`.	D
`ParsedInfo.Range.Maximum`	Double	RO	Numeric equivalent of string Data property `Props.Range.Maximum`.	D

Property	Type	Access	Description	Objects
`ParsedInfo.Range.Minimum`	Double	RO	Numeric equivalent of string Data property `Props.Range.Minimum`.	D
`Port`	Integer	RW	Port index number for this input or output data or event (default = `1`).	D E
`Props.Array.Size`	String	RW	Specifying a positive value for this property specifies that this data is an array of specified size. Equivalent to entering a positive value in the **Size** column for this data in the Model Explorer or in the **Size** field of the Data properties dialog box.	D
`Props.Array.FirstIndex`	String	RW	Index of the first element of this data if it is an array (`Props.Array.Size >= 1`). Equivalent to entering a value of zero or greater in the **First index** field of the Data properties dialog box.	D
`Props.InitialValue`	String	RW	If the source of the initial value for this data is the Stateflow hierarchy, this is the value used. Equivalent to entering this value in the **InitVal** column for this data in the Model Explorer or similar field in the Data properties dialog box.	D
`Props.Range.Maximum`	String	RW	Maximum value that this data can have during execution or simulation of the state machine. Equivalent to entering value in **Max** column for this data in the Model Explorer or the **Maximum** field in the Data properties dialog box.	D

Property	Type	Access	Description	Objects

`Props.`
`Range.`
`Minimum`
	String	RW	Minimum value that this data can have during execution or simulation of the state machine. Equivalent to entering value in the **Min** column for this data in the Model Explorer or in the **Minimum** field in the Data properties dialog box.	D

`SaveToWorkspace`	Boolean	RW	If set to true, this data is saved to the MATLAB workspace. Equivalent to selecting the **ToWS** column entry for this data in the Model Explorer or selecting the **Save final value to base workspace** check box in the Data properties dialog box.	D

Debugging Properties

The following properties control values used in debugging Stateflow chart applications with the Stateflow Debugger.

Property	Type	Access	Description	Objects
`Debug.Animation.Delay`	Double	RW	Specify a delay (slow down) value for animation. Equivalent to setting the **Delay (sec)** field in the Animation section of the Debugger window.	M
`Debug.Animation.Enabled`	Boolean	RW	If true, animation (simulation) is enabled. If false (=0), disabled. Equivalent to selecting the **Enabled** or **Disabled** radio button of the Animation section of the Debugger window.	M
`Debug.BreakOn.ChartEntry`	Boolean	RW	If true, sets the chart entry breakpoint for all charts in this machine. Equivalent to selecting the **Chart Entry** check box on the Debugger window.	M
`Debug.BreakOn.EventBroadcast`	Boolean	RW	If true, sets the event broadcast breakpoint for all charts in this machine. Equivalent to selecting the **Event Broadcast** check box on the Debugger window.	M
`Debug.BreakOn.StateEntry`	Boolean	RW	If true, sets the state entry breakpoint for all states in this machine. Equivalent to selecting the **State Entry** check box on the Debugger window.	M
`Debug.Breakpoints.EndBroadcast`	Boolean	RW	If true, sets a debugger breakpoint for the end of the broadcast of this event. Equivalent to selecting the **End of broadcast** check box in the Event properties dialog box.	E

Property	Type	Access	Description	Objects
`Debug.Breakpoints.StartBroadcast`	Boolean	RW	If true, sets a debugger breakpoint for the start of the broadcast of this event. Equivalent to selecting the **Start of broadcast** check box in the Event properties dialog box.	E
`Debug.Breakpoints.OnDuring`	Boolean	RW	If true, sets the during breakpoint for this object. Equivalent to selecting the **State During** check box in the properties dialog box for this state or selecting the **Function Call** check box in the properties dialog box for this graphical function or truth table.	F S TT
`Debug.Breakpoints.OnEntry`	Boolean	RW	If true, sets the entry breakpoint for this object. Equivalent to selecting the **Chart Entry** check box in the properties dialog box for this chart, or selecting the **State Entry** check box in the properties dialog box for this state.	C S
`Debug.Breakpoints.OnExit`	Boolean	RW	If true, sets the exit breakpoint for this object. Equivalent to selecting the **State Exit** check box in the properties dialog box for this state.	S
`Debug.Breakpoints.WhenTested`	Boolean	RW	If true, sets a debugging breakpoint to occur when this transition is tested to see if it is a valid transition. Equivalent to selecting the **When Tested** check box in the properties dialog box of this transition.	T
`Debug.Breakpoints.WhenValid`	Boolean	RW	If true, sets a debugging breakpoint to occur when this transition has tested as valid. Equivalent to selecting the **When Valid** check box in the properties dialog box of this transition.	T

Property	Type	Access	Description	Objects
`Debug.DisableAllBreakpoints`	Boolean	RW	If true, disables the use of all breakpoints in this machine. Equivalent to selecting the **Disable all** check box in the Debugger window.	M
`Debug.StateRunTimeCheck.Inconsistencies`	Boolean	RW	If true, checks for state inconsistencies during a debug session. Equivalent to selecting the **State Inconsistency** check box in the Debugger window.	M
`Debug.RunTimeCheck.TransitionConflicts`	Boolean	RW	If true, checks for transition conflicts during a debug session. Equivalent to selecting the **Transition Conflict** check box in the Debugger window.	M
`Debug.RunTimeCheck.CycleDetection`	Boolean	RW	If true, checks for cyclical behavior errors during a debug session. Equivalent to selecting the **Detect Cycles** check box in the Debugger window.	M
`Debug.RunTimeCheck.DataRangeChecks`	Boolean	RW	If true, checks for data range violations during a debug session. Equivalent to selecting the **Data Range** check box in the Debugger window.	M
`Debug.Watch`	Boolean	RW	If true, causes the Debugger window to halt execution if this data is modified. Equivalent to selecting the **Watch** column entry for this data in the Model Explorer or selecting the **Watch in debugger** check box in the Data properties dialog box.	D
`TestPoint`	Boolean	RW	If true (default = false), sets this data or state as a Stateflow test point. You can monitor Stateflow test points with a floating scope during simulation. You can also log test point values into MATLAB workspace objects. See “Monitoring Test Points in Stateflow	D S

Property	Type	Access	Description	Objects
			Charts" in the Stateflow and Stateflow Coder User's Guide.	

Display Control

In this section...

Display Methods

The following methods control the current display.

Method	Description	Objects
`dialog`	Display the properties dialog box of this object.	M C S B F N T J D E X TT EM
`fitToView`	Zoom in on this object and highlight it in the Stateflow Editor.	C S B F N T J TT EM SLF
`view`	Make this object visible for editing.	C S B F N T J D E X TT EM SLF
`zoomIn and zoomOut`	Causes the Stateflow Editor to zoom in or zoom out on this chart.	ED

Display Properties

The following properties affect the display of the current Stateflow chart.

Property	Type	Access	Description	Objects
`Visible`	Boolean	RO	If true, indicates that this object is currently displayed in a Stateflow Editor window.	C
`Subviewer`	Chart or State	RO	State or chart in which you can view this object graphically.	S B F N T J TT EM SLF
`ZoomFactor`	Double	RW	View magnification level (zoom factor) of this chart in the Stateflow Editor.	ED

Graphical Appearance

In this section...

Color Properties

The following properties set colors for the graphical objects in Stateflow charts.

Property	Type	Access	Description	Objects
`ChartColor`	[R,G,B]	RW	Background color of this chart in a 1-by-3 RGB array with each value normalized on a scale of 0 to 1.	C
`ErrorColor`	[R,G,B]	RW	Set the RGB color for errors in the Stateflow Editor in a 1-by-3 RGB array with each value normalized on a scale of 0 to 1. Equivalent to changing **Error** color in the Colors & Fonts dialog box under **Edit > Style**.	C
`JunctionColor`	[R,G,B]	RW	Set the RGB color for junctions in the Stateflow Editor in a 1-by-3 RGB array with each value normalized on a scale of 0 to 1. Equivalent to changing the **Junction** color in the Colors & Fonts dialog box under **Edit > Style**.	C
`SelectionColor`	[R,G,B]	RW	Color of selected items for this chart in a 1-by-3 RGB array with each value normalized on a scale of 0 to 1. Equivalent to changing the **Selection** color in the Colors & Fonts dialog box under **Edit > Style**.	C

Property	Type	Access	Description	Objects
`StateColor`	[R,G,B]	RW	Color of the state box in a 1-by-3 RGB array with each value normalized on a scale of 0 to 1. Equivalent to changing the **State/Frame** color in the Colors & Fonts dialog box under **Edit > Style**.	C
`StateLabelColor`	[R,G,B]	RW	Color of the state labels for this chart in 1-by-3 RGB array with each value normalized on a scale of 0 to 1. Equivalent to changing the label color of **StateLabel** in the Colors & Fonts dialog box under **Edit > Style**.	C
`TransitionColor`	[R,G,B]	RW	Set the RGB color for transitions in the Stateflow Editor in a 1-by-3 RGB array with each value normalized on a scale of 0 to 1. Equivalent to changing the **Transition** color in the Colors & Fonts dialog box under **Edit > Style**.	C
`Transition LabelColor`	[R,G,B]	RW	Color of the transition labels for this chart in 1-by-3 RGB array with each value normalized on a scale of 0 to 1. Equivalent to changing the label color of **TransitionLabel** in the Colors & Fonts dialog box under **Edit > Style**.	C

Drawing Properties

The following properties control how Stateflow objects are drawn in their charts.

Property	Type	Access	Description	Objects
`ArrowSize`	Double	RW	Size of transition arrows coming into this object. Equivalent to selecting **Arrowhead Size** from the context menu for this state.	S B F T J TT EM SLF
`DrawStyle`	String	RW	Drawing style for this transition. Set to `'SMART'` (default) for smart transitions or `'STATIC'` for static transitions. Equivalent to selecting **Smart** from the context menu for this transition to toggle between settings. **Note** Transition must be connected to effect a change in the `DrawStyle` property. Otherwise, an error occurs.	T
`Editor`	Editor	RO	Editor object for this chart.	C

Font Properties

The following properties change the font used for text in a Stateflow chart.

Property	Type	Access	Description	Objects
`Font.Angle`	Enum	RW	Style of the font for the text in this note. Can be `'ITALIC'` or `'NORMAL'`. This property overrides the default style for this note, which is set by the `StateFont.Angle` property of the Chart object containing this note.	N
`Font.Name`	String	RO	Name of the font for the text in this note. This property is read-only (RO) and set by the `StateFont.Name` property of the Chart object containing this note.	N

Property	Type	Access	Description	Objects
`Font.Size`	Double	RW	Size of the font for the label text for this note. This property overrides the default size for this note, which is set by the `StateFont.Size` property of the Chart object containing this note. Equivalent to selecting **Font Size** > **<font size>** in the context menu for this note.	N
`Font.Weight`	Enum	RW	Weight of the font for the label text for this note. Can be `'BOLD'` or `'NORMAL'`. This property overrides the default weight for the text in this note, which is set by the `StateFont.Weight` property of the Chart object containing this note.	N
`FontSize`	Double	RW	Size of the font for the label text for this object. This property overrides the default size for this object, which is set by the `StateFont.Size` property (`TransitionFont.Size` for transitions) of the Chart object containing this object. Equivalent to selecting **Font Size** > **<font size>** in the context menu for this object.	S B F T TT EM SLF
`StateFont.Angle`	Enum	RW	Font angle for the labels of State, Box, Function, and Note objects. Can be `'ITALIC'` or `'NORMAL'`. Equivalent to **Italic** and **Regular** settings when changing the font style of the **StateLabel** in the Colors & Fonts dialog box under **Edit** > **Style**. Use with property `StateFont.Weight` to achieve Bold Italic style. You can individually override this property with the `Font.Angle` property for Note objects.	C

Property	Type	Access	Description	Objects
`StateFont.Name`	String	RW	Font style used for the labels of State, Box, Function, and Note objects. Enter a string for the font name -- no selectable values. Font remains set to previous font for unrecognized font strings. Equivalent to changing the font of **StateLabel** in the Colors & Fonts dialog box under **Edit** > **Style**.	C
`StateFont.Size`	Integer	RW	Font size for the labels of State, Box, Function, and Note objects. Equivalent to changing the font size of **StateLabel** in the Colors & Fonts dialog box under **Edit** > **Style**. You can individually override this property with the `FontSize` property for State, Box, and Function objects and with the `Font.Size` property for Note objects.	C
`StateFont.Weight`	Enum	RW	Font weight for state labels. Can be `'BOLD'` or `'NORMAL'`. Equivalent to **Bold** and **Regular** settings of **StateLabel** in the Colors & Fonts dialog box under **Edit** > **Style**. Use with the property `StateFont.Angle` to achieve Bold Italic style. You can individually override this property with the `Font.Weight` property for Note objects.	C
`TransitionFont.Angle`	Enum	RW	Font angle for state labels. Can be `'ITALIC'` or `'NORMAL'`. Equivalent to **Italic** and **Regular** settings when changing font style of **TransitionLabel** in the Colors & Fonts dialog box under **Edit** > **Style**. Use with property `StateFont.Weight` to achieve Bold Italic style.	C

Property	Type	Access	Description	Objects
`TransitionFont.Name`	String	RW	Font used for transition labels. Enter string for font name (no selectable values). Font remains set to previous font for unrecognized font strings. Equivalent to changing the font of **TransitionLabel** in the Colors & Fonts dialog box under **Edit** > **Style**.	C
`TransitionFont.Size`	Integer	RW	Default font size for transition labels. Truncated to closest whole number less than or equal to entered value. Equivalent to changing font size of **TransitionLabel** in the Colors & Fonts dialog box under **Edit** > **Style**.	C
`TransitionFont.Weight`	Enum	RW	Font weight for transition labels. Can be `'BOLD'` or `'NORMAL'`. Equivalent to **Bold** and **Regular** settings when changing font style of **TransitionLabel** in the Colors & Fonts dialog box under **Edit** > **Style**. Use with property `StateFont.Angle` to achieve Bold Italic style.	C

Position Properties

The following properties control the position of Stateflow objects in a Stateflow chart.

Property	Type	Access	Description	Objects
`BadIntersection`	Boolean	RO	If true, this object graphically intersects another state, box, or function in an invalid way.	S B F TT EM SLF
`Destination`	State or Junction	RW	Destination state or junction of this transition. Assign `Destination` the destination object for this transition. You can also use the property `Destination` to detach the destination end of a transition through the command `t.Destination = []` where `t` is the Transition object.	T
`Destination OClock`	Double	RW	Location of transition destination connection on state. Varies from 0 to 12 for full clock cycle location. Value taken as modulus 12 of entered value.	T
`LabelPosition`	Rect	RW	Position and size of this transition's label in the Stateflow chart, given in the form of a 1-by-4 array consisting of the following: • (x,y) coordinates for the label's left upper vertex relative to the upper left vertex of the Stateflow Editor workspace • Width and height of the label	T
`MidPoint`	Rect	RW	Position of the midpoint of this transition relative to the upper left corner of the Stateflow Editor workspace, in a 1-by-2 (x,y) point array.	T

Property	Type	Access	Description	Objects
`Position`	Rect	RW	Position and size of box-like objects in the Stateflow chart, given in the form of a 1-by-4 array consisting of the following: • (x,y) coordinates for the object's left upper vertex relative to the upper left vertex of the Stateflow Editor workspace • Width and height of the box	S B F TT EM SLF N
`Position.Center`	Rect	RW	(x,y) position of junction relative to the upper left vertex of the parent chart or state.	J
`Position.Radius`	Double	RW	Radius of this junction.	J
`Source`	State or Junction	RW	Source state or junction of this transition. Assign `Source` the source object for this transition. You can also use the property `Source` to detach the source end of a transition with the command `t.Source = []` where `t` is the Transition object.	T
`SourceEndPoint`	Rect	RO*	x,y spatial coordinates for the endpoint of a transition. *This property can be changed only for default transitions. For all other transitions it is RO (read only).	T

Property	Type	Access	Description	Objects
SourceOClock	Double	RW	Location of transition source connection on state. Varies from 0 to 12 for full clock cycle location. Value taken as modulus 12 of entered value.	T
WindowPosition	Rect	RW	Position and size of this chart given in the form of a 1-by-4 array consisting of the following: • (x,y) coordinates for the window's left bottom vertex relative to the lower left corner of the screen • Width and height of the box	ED

Text Properties

The following properties control the text and text appearance apart from font and color in Stateflow charts.

Property	Type	Access	Description	Objects
Alignment	Enum	RW	Alignment of text in note box. Can be 'LEFT', 'CENTER', or 'RIGHT'.	N
Interpretation	Enum	RW	How the text in this note is interpreted for text processing. Can be 'NORMAL' or 'TEX'.	N
LabelString	String	RW	Label for this object. Equivalent to typing the label for this object in its label text field in the Stateflow Editor.	S B F T TT EM SLF
Text	String	RW	Label for this note. The text content for this note that you enter directly into the note in the Stateflow Editor or in the **Label** field of the Note properties dialog box.	N

Identifiers

The following properties identify objects for the version of Stateflow software.

Property	Type	Access	Description	Objects
`Description`	String	RW	Description of this object. Equivalent to entering a description in the **Description** field of the properties dialog box for this object (except for Simulink functions).	M C S B F N T J D E X TT EM SLF
`Document`	String	RW	Document link to this note. Equivalent to entering a link in the **Document Link** field of the properties dialog box for this object (except for Simulink functions).	M C S B F N T J D E X TT EM SLF
`Id`	Integer	RO	Unique identifier assigned to this object to distinguish it from other objects loaded in memory.	M C S B F N T J D E X TT EM SLF
`Name`	String	RW	Name of this object. This property is RW except for the name of Machine object, which is RO.	M C S B F D E X TT EM SLF
`SfVersion`	Double	RO	Full version number for current Stateflow software. For example, the string `'41112101'` appears for Stateflow software version 4.1.1 and MATLAB software version 12.1. The remaining `'01'` is for internal use.	M

Property	Type	Access	Description	Objects
`Tag`	Any Type	RW	A field you can use to hold data of any type for this object.	M C S B F T J D E X TT EM SLF
`Type`	Enum	RO	Type of this state or junction. For states, can be one of the following: • `'OR'` (inclusive) • `'AND'` (parallel) The type of a state is determined by the parent's `Decomposition` property. For junctions, can be one of the following: • `'CONNECTIVE'` • `'HISTORY'`	S J

Interface to Simulink Model

The following properties (and methods) control how data and events are input from and output to the Simulink model for a Stateflow chart.

Property (Method)	Type	Access	Description	Objects
`ChartUpdate`	Enum	RW	Activation method of this chart. Can be one of the following: • `'INHERITED'` (Inherited) • `'DISCRETE'` (Discrete) • `'CONTINUOUS'` (Continuous) These preceding entries are equivalent to the parenthetical entries for the **Update method** field in the Chart properties dialog box.	C
`ExecuteAtInitialization`	Boolean	RW	If set to true, initialize this chart's state configuration at time zero instead of at first input event. Equivalent to selecting the **Execute (enter) Chart At Initialization** check box in the Chart properties dialog box.	C
`ExportChartFunctions`	Boolean	RW	If set to `true` (default = `false`), graphical functions at chart level are made global. Equivalent to selecting the **Export Chart Level Graphical Functions (Make Global)** check box in the Chart properties dialog box.	C

Property (Method)	Type	Access	Description	Objects
`InitializeOutput`	Boolean	RW	Applies the initial value of outputs every time a chart wakes up, not only at time 0. See "Setting Properties for a Single Chart".	C
(`outputData`)	No Return	NA	Output the activity status of this state to the Simulink base workspace via a data output port on the Chart block of this state.	S
`Port`	Integer	RW	Port index number for this input or output data or event (default = `1`).	D E
`SampleTime`	String	RW	Sample time for activating this chart. Applies only when the `ChartUpdate` property for this chart is set to `'DISCRETE'` (= **Discrete** in the **Update method** field in the Chart properties dialog box).	C
`SaveToWorkspace`	Boolean	RW	If set to true, this data is saved to the MATLAB workspace. Equivalent to selecting the **SaveToWorkspace** column entry for this data in the Model Explorer or selecting the **Save final value to base workspace** field in the Data properties dialog box.	D

Property (Method)	Type	Access	Description	Objects
`Scope`	Enum	RW	Scope of this data. Allowed values vary with the object containing this data, which are as follows: • `'Local'` • `'Constant'` • `'Imported'` (machine objects only) • `'Exported'` (machine objects only) • `'Input'` (chart objects only) • `'Output'` (chart objects only) • `'Temporary'` (function objects only) • `'Function input'` (function objects only) • `'Function output'` (function objects only) Above values correspond to entries in the **Scope** field of the Data or Event properties dialog box.	D E

Property (Method)	Type	Access	Description	Objects
`StrongDataTyping WithSimulink`	Boolean	RW	If set to true, set strong data typing with Simulink I/O. Equivalent to selecting the **Use Strong Data Typing with Simulink I/O** check box in the Chart properties dialog box.	C
`Trigger`	Enum	RW	Type of signal that triggers this chart input event. Also the type of trigger associated with this chart output event. The following triggers apply to both chart input and output events: • `'Either'` (**Either Edge**) • `'Function call'` (**Function Call**) The following triggers apply only to chart input events: • `'Rising'` (**Rising Edge**) • `'Falling'` (**Falling Edge**) The preceding entries are equivalent to the entries in parentheses for the **Trigger** field in the Event properties dialog box.	E

Machine (Model) Identifier Properties

The following properties identify parts of the Simulink model containing a Stateflow chart.

Property	Type	Access	Description	Objects
`Created`	String	RO	Date of creation of this machine.	M
`Creator`	String	RW	Creator of this machine.	M
`Dirty`	Boolean	RW	If true, this object has changed since it was opened or saved.	M C
`FullFileName`	String	RO	Full path name of file under which this machine (model) is stored.	M
`Iced`	Boolean	RO	Equivalent to property `Locked` except that this property is used internally to lock this object from being changed during activities such as simulation.	M C
`IsLibrary`	Boolean	RO	If true, specifies that the current model builds a library and not an application.	M
`Locked`	Boolean	RW	If set to true, prevents user from changing any Stateflow chart in this machine or chart.	M C
`Modified`	String	RW	Comment area for entering date and name of modification to this machine (model).	M
`Version`	String	RW	Comment string for recording version of this model.	M

Truth Table Construction Properties

The following properties control the definition of a truth table.

Property	Type	Access	Description	Objects
`ActionTable`	Cell Array	RW	A cell array of strings containing the contents of the Action Table for this truth table.	TT
`ConditionTable`	Cell Array	RW	A cell array of strings containing the contents of the Action Table for this truth table.	TT
`OverSpec Diagnostic`	String	RW	Interprets the error diagnosis of this truth table as overspecified according to the possible values `'Error'`, `'Warning'`, or `'None'`. In the Truth Table Editor, the value of this property is assigned by selecting **Overspecified** from the **Diagnostics** menu item and then selecting one of the three values.	TT
`UnderSpec Diagnostic`	String	RW	Interprets the error diagnosis of this truth table as underspecified according to the possible values `'Error'`, `'Warning'`, or `'None'`. In the Truth Table Editor, the value of this property is assigned by selecting **Underspecified** from the **Diagnostics** menu item and then selecting one of the three values.	TT

4

API Method Reference

build

Purpose

Build custom target incrementally

Syntax

```
thisCustomTarget.build
```

Description

The `build` method incrementally builds this custom target and performs these actions:

- Parses all charts completely.
- Generates code for charts incrementally.

If a complete build has already taken place, this method builds only those portions of the target corresponding to charts that have changed since the last build.

Arguments

`thisCustomTarget`	The custom Target object to build.

Returns

None

Example

If `t` is a custom Target object, the command `t.build` builds the target for the Stateflow charts that have changed in the target's model since the last build or code generation.

See Also

The methods `rebuildAll`, `generate`, `regenerateAll`, and `make`

Purpose

Provide handle to schema class of object type

Syntax

```
handle = thisObject.classhandle
```

Description

The `classhandle` method returns a read-only handle to the schema class of this object's type. You can use the `classhandle` method to provide information about the structure of each object type.

Arguments

`thisObject`	The object for which to return a handle. Can be any Stateflow object.

Returns

`handle`	Handle to schema class of this object.

Example

If `j` is a Junction object, the class handle of a Junction object is `j.classhandle`. You can see the class schema for a Junction object by using the following `get` command:

```
j.classhandle.get
```

Two member arrays of the displayed class schema are `Properties` and `Methods`. These two members are members of the schema class for every object.

List the class schema for Properties with the following command:

```
j.classhandle.Properties.get
```

Two displayed members of the Properties schema are `Name` and `DataType`. Finally, using the class handle for a junction, you can display the properties of a Junction object along with their data types with the following command:

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

copy

Purpose

Copy specified array of objects to clipboard

Syntax

```
cbObj.copy(objArray)
```

Description

The `copy` method copies the specified objects to the clipboard. Objects to copy are specified through a single argument array of objects.

Later, complete the copy operation by invoking the `pasteTo` method.

Arguments

cbObj	The Clipboard object to copy to.
`objArray`	Array of Stateflow objects to copy. These objects must conform to the following: • The objects copied must be all graphical (states, boxes, functions, transitions, junctions) or all nongraphical (data, events, targets). • If all objects are graphical, they must all be seen in the same subviewer.

Returns

None

Example

See “Copying Objects” on page 1-32.

Purpose

Return default transitions in object at top level of containment

Syntax

```
defaultTransitions = thisObject.defaultTransitions
```

Description

The `defaultTransitions` method returns the default transitions in this object at the top level of containment.

Arguments

`thisObject`	The object for which to return default transitions. Can be an object of type Chart, State, Box, or Function.

Returns

`defaultTransitions`	Array of default transitions in this object at the top level of containment.

Example

If state `A` contains state `A1`, and state `A1` contains state `A11`, and states `A1` and `A11` have default transitions attached to them, the `defaultTransitions` method of state `A` returns the default transition attached to state `A1`.

delete

Purpose

Delete object

Syntax

```
thisObject.delete
```

Description

The `delete` method deletes this object from the model. This is true for all but objects of type Root, Chart, Clipboard, and Editor.

Arguments

`thisObject`	The object to delete. Can be an object of type Machine, State, Box, Function, Truth Table, Note, Transition, Junction, Data, Event, or Target.

Returns

None

Example

If a state `A` is represented by the State object `sA`, the command `sA.delete` deletes state `A`.

Purpose

Open properties dialog box of object

Syntax

```
thisObject.dialog
```

Description

The `dialog` method opens the Properties dialog box of its object.

Arguments

`thisObject`	The object for which to open the Properties dialog box.

Returns

None

Example

If state `A` is represented by State object `sA`, the MATLAB command statement `sA.dialog` opens the Properties dialog box for state `A`.

disp

Purpose

Display properties and settings for object

Syntax

```
thisObject.disp
```

Description

The `disp` method displays the properties and settings for this object. This is true for all but objects of type Root and Clipboard.

Arguments

`thisObject`	The object for which to display properties and settings.

Returns

None

Example

If a state `A` is represented by the State object `sA`, the command `sA.disp` displays the property names and their settings for state `A`.

Purpose

Return specified objects

Syntax

```
objArray = thisObject.find(Specifier,Value, ...)
```

Note You can also nest specifications using braces ({}).

Description

Using combinations of specifier-value argument pairs, the `find` method returns objects in this object that match the specified criteria. The specifier-value pairs can be property based or based on other attributes of the object such as its depth of containment. Specifiers can also be logical operators (`-and`, `-or`, etc.) that combine other specifier-value pairs.

By default, the `find` command finds objects at all depths of containment within an object. You can specify the maximum depth of search with the `-depth` specifier. However, the zeroth level of containment, i.e., the searched object itself, is always included if it happens to satisfy the search criteria.

If no arguments are specified, the `find` command returns all objects of this object at all levels of containment.

Arguments

`thisObject`	The object for which to find contained objects. Can be an object of type Root, Machine, State, Box, Function, or Truth Table.
`'-and'`	No value is paired to this specifier. Instead, this specifier relates a previous specifier-value pair to a following specifier-value pair in an AND relation.

`'-class'` String class name of the class to search for. Use this option to find all objects whose class exactly matches a given class. To allow matches for subclasses of a given class, use the `-isa` specifier. Classes are specified as the string name (e.g., `'Stateflow.State'`, `'Stateflow.Transition'`, etc.) or as a handle to the class (see the method `classhandle`).

`'-depth'` Integer depth to search, which can be 0, 1, 2,..., infinite. The default search depth is infinite.

Note For a machine object, using the `'-depth'` switch with the `find` method is not supported.

`'-function'` Handle to a function that evaluates each object visited in the search. The function must always return a logical scalar value that indicates whether or not the value is a match. If no property is specified, the function is passed the handle of the current object in the search. If a property is specified, the function is passed the value of that property.

In the following example, a function with handle `f` (defined in first line) is used to filter a find to return only those objects of type 'andState':

```
f = @(h) (strcmp(get(h,'type'), 'andState'));
objArray = thisObject.find('-function', f);
```

`'-isa'` Name of the type of objects to search for. Object types are specified as a string name (e.g., `'Stateflow.State'`, `'Stateflow.Transition'`, etc.) or as a handle to the object type (see method `classhandle`).

`'-method'`	String that specifies the name of a method belonging to the objects to search for.
`'-not'`	No value is paired to this specifier. Instead, this specifier searches for the negative of the following specifier-value pair.
`'-or'`	No value is paired to this specifier. Instead, this specifier relates the previous specifier-value pair to the following specifier-value pair in an OR relation.

Note If no logical operator is specified, `-or` is assumed.

`'`*`property`*`'`	The specifier takes on the name of the property. Value is the string value of the specified property for the objects you want to find.
`'-property'`	String name of the property that belongs to the objects you want to find.
`'-xor'`	No value is paired to this specifier. Instead, this specifier relates the previous specifier-value pair to the following specifier-value pair in an XOR relation.
`'-regexp'`	No value follows this specifier. Instead, this specifier indicates that the value of the following specifier-value pair contains a regular expression.

Returns

`objArray`	Array of objects found matching the criteria specified.

Example

If a Chart object `c` represents a Stateflow chart, the command `states=c.find('-isa','Stateflow.State')` returns an array, `states`, of all the states in the chart, and the command `states=c.find('Name','A')` returns an array of all objects whose `Name` property is `'A'`.

If state A, which is represented by State object sA, contains two states, A1 and A2, and you specify a find command that finds all the states in A as follows,

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

then the above command finds three states: A, A1, and A2.

Purpose

Zoom in on graphical Stateflow object

Syntax

```
thisObject.fitToView
```

Description

The `fitToView` method zooms in on this Stateflow object and highlights it in the Stateflow Editor.

Arguments

`thisObject` The object on which to zoom.

Returns

None

Example

If `myState` is a State object, the command `myState.fitToView` zooms in on that state and highlights it in the Stateflow Editor.

See Also

The methods `view` and `zoomIn and zoomOut`

generate

Purpose

Generate code incrementally for custom target

Syntax

```
thisCustomTarget.generate
```

Description

The `generate` method generates code incrementally for this custom target. If a complete code generation has already taken place, it performs an incremental generation only for those portions of the target corresponding to charts that have changed since the last code generation.

Arguments

`thisCustomTarget`	The custom Target object for which to generate code.

Returns

None

Example

If `t` is a custom Target object, the command `t.generate` generates code for the Stateflow charts that have changed in the target's model since the last code generation.

See Also

The methods `build`, `rebuildAll`, `regenerateAll`, and `make`

Purpose

Return MATLAB structure containing property settings of object or array of objects

Syntax

```
propList = thisObject.get(prop)
```

Description

The `get` method returns and displays a MATLAB structure containing the settings for the specified property of this object. If no property is specified, the settings for all properties are returned.

The `get` method is also vectorized so that it returns an `m`-by-`n` cell array of values for an array of `m` objects and an array of `n` properties.

Arguments

`thisObject`	The object for which to get specified property.
`prop`	String name of property (e.g., `'FontSize'`) to get value for. Can also be an array of properties (see return `propList` below). If no property is specified, a list of all properties is returned.

Returns

`propList`	MATLAB structure listing the properties of this object. Can also be an `m` by `n` cell array of values if `thisObject` is an array of `m` objects and `prop` is an array of `n` properties.

Example

State `A` is represented by the State object `sA`.

The following command lists the properties of state `A`:

```
sA.get
```

The following command returns a handle to a MATLAB structure of the properties of state `A` to the workspace variable `Aprops`:

```
Aprops = sA.get
```

getCodeFlag

Purpose

Return specified code flag for custom target

Syntax

```
thisCustomTarget.getCodeFlag(name)
```

Description

The `getCodeFlag` method returns the value of a code flag whose name you specify.

Arguments

`thisCustomTarget`	The custom Target object for which to get the code flag value.
`name`	The short string name of the code flag for which to get the value. See "CodeFlagsInfo Property of Targets" on page 2-66 for a list of these names.

Returns

None

Example

Assume that the Target object `x` represents the custom target for the loaded model. If `m` is the Stateflow machine object for this model, you can obtain `x` with this command:

```
x = m.find('-isa','Stateflow.Target','Name','ctarg')
```

A custom target has thirteen code flags. You can verify this by looking at the `CodeFlagsInfo` property of `x` with this command:

```
x.CodeFlagsInfo.name
```

You enable or disable the `comments` code flag using the **User Comments in generated code** check box in the properties dialog box for this target. By default, this flag is turned on (== 1) for the custom target. You can verify this setting with the following command:

```
x.getCodeFlag('comments')
```

Similarly, you can get the values of the `statebitsets` and `databitsets` code flags, which you enable or disable using check boxes in the same dialog box. You can verify the settings with these commands:

```
x.getCodeFlag('statebitsets')
x.getCodeFlag('databitsets')
```

See Also

The method `setCodeFlag`

help

Purpose

Display list of properties for object with accompanying descriptions

Syntax

```
thisObject.help
```

Description

The `help` method returns a list of properties for any object. To the right of this list appear simple descriptions for each property. Some properties do not have descriptions because their names are descriptive in themselves.

Arguments

None

Returns

None

Example

If `j` is an API handle to a Stateflow junction, the command `j.help` returns a list of the property names and descriptions for a Stateflow API object of type Junction.

Purpose

Return inner transitions that originate with chart or state and terminate on contained object

Syntax

```
transitions = thisObject.innerTransitions
```

Description

The `innerTransitions` method returns the inner transitions that originate with this object and terminate on a contained object.

Arguments

None

Returns

`thisObject`	Object for which to get inner transitions. Can be of type State or Box.
`transitions`	Array of inner transitions originating with this object and terminating on a contained state or junction.

Example

State `A` contains state `A1`, and state `A1` contains state `A11`. State `A` has two transitions, each originating from its inside edge and terminating inside it. These are inner transitions. One transition terminates with state `A1` and the other terminates with state `A11`. The `innerTransitions` method of state `A` returns both of these transitions.

make

Purpose

Incrementally compile custom target with no code generation

Syntax

```
thisCustomTarget.make
```

Description

This method incrementally compiles a custom target, but does not generate code. It compiles only those portions of generated code that have changed since the last compilation.

Arguments

`thisCustomTarget`	The custom Target object for which to do make.

Returns

None

Example

Suppose you define `t` as a custom Target object for the Simulink model `mytest`, which contains a Stateflow chart. The command `t.make` incrementally compiles generated code for that target.

See Also

The methods `build`, `rebuildAll`, `generate`, and `regenerateAll`

Purpose

List methods belonging to object

Syntax

```
thisObject.methods
```

Description

The `methods` method lists the names of the methods belonging to this object.

Note The `methods` method for this object displays some internal methods that do not apply to chart use, and are not documented. Unsupported methods include: `areChildrenOrdered`, `evalDialogParams`, `getChildren`, `getCurrentDialogPrompts`, `getDialogInterface`, `getDialogProxy`, `getDialogSchema`, `getDisplayClass`, `getDisplayIcon`, `getDisplayLabel`, `getFullName`, `getHierarchicalChildren`, `getInstanceProperties`, `getParent`, `getPreferredProperties`, `isHierarchical`, `isLibrary`, `isLinked`, `isMasked`, `isModelReference`, `isTunableProperty`, `isValidProperty`.

Arguments

`thisObject`	Object for which to list methods. Can be of any Stateflow object type.

Returns

None

Example

If state `A` is represented by State object `sA`, the command `sA.methods` lists the methods of state `A`.

outerTransitions

Purpose

Return array of outer transitions for object

Syntax

```
transitions = thisObject.outerTransitions
```

Description

The `outerTransitions` method returns an array of transitions that exit the outer edge of this object and terminate on objects outside the containment of this object.

Arguments

None

Returns

`thisObject`	The object for which to find outer transitions. Can be of object type State or Box.
`transitions`	An array of transitions exiting the outer edge of this state.

Example

A chart contains three states, `A`, `B`, and `C`. State `A` is connected to state `B` through a transition from state `A` to state `B`. State B is connected to state `C` through a transition from state `B` to state `C`. And state `C` is connected to state `A` through a transition from state `C` to state `A`. If state `A` is represented by State object handle `sA`, the command `sA.outerTransitions` returns the transition from state `A` to state `B`.

Purpose

Create, retrieve, or delete data output to Simulink base workspace of state activity status

Syntax

```
StateData = thisState.outputData(action)
```

Description

The `outputData` method of this state creates, retrieves, or deletes a special data object of type `State`. This data is attached internally to an output port on this state's Stateflow block in a Simulink model to output the activity status of this state to the Simulink base workspace during run-time.

Note You cannot use the Model Explorer to create Data objects of type `State`.

Arguments

`thisState` The state object for which to add a special port.

`action` This string value can be one of the following:

- `'create'` — Returns a new data object of type `State` and attaches it internally to a new state activity output port on this state's Stateflow block.
- `'get'` — Returns this state's existing data object of type `State` attached internally to an existing state activity output port on this state's Stateflow block.
- `'delete'` — Deletes this state's data object of type `State` and the state activity output port on its Stateflow block to which it is attached.

Returns

`StateData` The data object of type `State` for this state

Example

If state `A` is represented by State object `sA`, the following command creates a new data object of type State, which is output to the Simulink base workspace and contains state `A`'s activity:

```
s.outputData('create')
```

The Chart block in the Simulink model that contains state `A` now has an output port labeled `A`, the name of state `A`. In the Model Explorer, state `A` now contains a data object of type `State` whose scope is `Output to Simulink`.

The following command returns a Data object, `d`, for the data output to the Simulink base workspace containing state `A`'s activity:

```
s.outputData('get')
```

The following command deletes the data output to the Simulink base workspace containing state `A`'s activity:

```
s.outputData('delete')
```

Purpose

Parse single chart or all charts in model

Syntax

```
thisChart.parse
thisMachine.parse
```

Description

For Chart objects, the `parse` method parses this Stateflow chart. This is equivalent to selecting **Parse** from the **Tools** menu of the Stateflow Editor window for this chart.

For Machine objects, the `parse` method parses all the charts in this machine.

Arguments

`thisChart`	The chart to parse.
`thisMachine`	The machine containing charts to parse.

Returns

None

Example

If `ch` is a handle to an API object representing a chart, then the command `ch.parse` parses the chart.

pasteTo

Purpose

Paste objects in clipboard to specified container object

Syntax

```
clipboard.pasteTo(newContainer)
```

Description

The `paste` method pastes the contents of the Clipboard to the specified container object. The receiving container is specified through a single argument. Use of this method assumes that you placed objects in the Clipboard with the `copy` method.

Arguments

`newContainer`	The Stateflow object to receive a copy of the contents of the Clipboard object. If the objects in the Clipboard are all graphical (states, boxes, functions, notes, transitions, junctions), this object must be a chart or subchart.

Returns

None

Example

See the section “Copying Objects” on page 1-32.

Purpose

Completely rebuild custom target

Syntax

```
thisCustomTarget.rebuildAll
```

Description

The `rebuildAll` method completely rebuilds this custom target with the following actions:

- Parses all charts completely.
- Regenerates code for all charts completely.

Arguments

`thisCustomTarget`	The custom Target object to rebuild.

Returns

None

Example

If `t` is a custom Target object, the command `t.rebuildAll` completely rebuilds that target.

See Also

The methods `build`, `generate`, `regenerateAll`, and `make`

regenerateAll

Purpose

Completely regenerate code for custom target

Syntax

```
thisCustomTarget.regenerateAll
```

Description

The `regenerateAll` method regenerates this custom target. Regardless of previous code generations, it regenerates code for all charts in this target's model.

Arguments

`thisCustomTarget`	The custom Target object for which to regenerate code.

Returns

None

Example

If `t` is a custom Target object, the command `t.regenerateAll` completely regenerates code for the Stateflow charts in that target's model.

See Also

The methods `build`, `rebuildAll`, `generate`, and `make`

Purpose

Set properties with specified values

Syntax

```
thisObject.set(propName,value,...)
```

Note Arguments can consist of an indefinite number of property (name, value) pairs.

Description

The `set` method sets the value of a specified property or sets the values of a set of specified properties for this object. You specify properties and values through pairs of property (name, value) arguments.

The `get` method is also vectorized so that it sets an `m`-by-`n` cell array of values for an array of `m` objects and an array of `n` properties.

Arguments

`thisObject`	The object for which the specified property is set. Can be any Stateflow object.
`propName`	String name of the property to set (e.g., `'FontSize'`). Can also be a cell array of `m` property names.
`value`	New value for the specified property. Can be a cell array of `m`-by-`n` values if `thisObject` is an array of `m` objects and `propName` is an array of `n` property names.

Returns

None

Example

The following command sets the `Name` and `Description` properties of the State object `s`:

```
s.set('Name', 'Kentucky', 'Description', 'Bluegrass State')
```

The following command sets the `Position` property of the State object `s`:

```
s.set('Position',[200,119,90,60])
```

Purpose

Set code flag for custom target to value you specify

Syntax

```
thisCustomTarget.setCodeFlag(name,value)
```

Description

The `setCodeFlag` method sets the value of a code flag whose name you specify.

Arguments

`thisCustomTarget`	The custom Target object for which to set the code flag.
`name`	Short string name of the code flag. See "CodeFlagsInfo Property of Targets" on page 2-66 for a list of these names.
`value`	Value of code flag. Can be of any type.

Flag values can vary in type. Use the property `CodeFlagsInfo` to obtain the type for a specific flag.

Returns

None

Example

Assume that the Target object `x` represents the custom target for the loaded model. If `m` is the Stateflow machine object for this model, you can obtain `x` with the following command:

```
x = m.find('-isa','Stateflow.Target','Name','ctarg')
```

A custom target has thirteen code flags. You can verify this by looking at the `CodeFlagsInfo` property of `x` with this command:

```
x.CodeFlagsInfo.name
```

You enable or disable the `comments` code flag using the **User Comments in generated code** check box in the properties dialog box

for this target. By default, this flag is turned on (== 1) for the custom target. You can verify this setting with the following command:

```
x.getCodeFlag('comments')
```

If you want to disable user comments, enter this command:

```
x.setCodeFlag('comments',0)
```

See Also

The method `getCodeFlag`

Purpose

Return transitions that have object as source

Syntax

```
transitions = thisObject.sourcedTransitions
```

Description

The `sourcedTransitions` method returns all inner and outer transitions that have their source in this object.

Arguments

`transitions`	Source object of the returned transitions. Can be of type State, Box, Function, or Junction.

Returns

`transitions`	Array of all transitions whose source is this object

Example

Suppose that a chart contains three states, `A`, `B`, and state `A1`, which is contained by state `A`. The chart also has three transitions: one from `A` to `B` labeled `AtoB`, one from `B` to `A` labeled `BtoA`, and one from the inner edge of `A` to its state `A1` (inner transition) labeled `AtoA1`. If State object `sA` represents state `A`, the command `sA.sourcedTransitions` returns two transitions: the outer transition labeled `AtoB` and the inner transition labeled `AtoA1`.

Stateflow.Box

Purpose

Constructor for creating box

Syntax

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

Description

The `Stateflow.Box` method is a constructor method for creating boxes in a parent chart, state, box, or function, that returns a handle to an Event object for the new function.

Arguments

`parent`	Handle to an object for the parent chart, state, box, or function of the new box

Returns

`box_new`	Handle to the Box object for the new box

Example

If `sA` is a handle to a State object for an existing state `A`, the following command creates a new box parented (contained by) state `A`:

```
box_new = Stateflow.Box(sA)
```

The new box is unnamed and appears in the upper left corner inside state `A`. `box_new` is a handle to a Box object for the new box.

Purpose

Constructor for creating data

Syntax

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

Description

The `Stateflow.Data` method is a constructor method for creating data for a parent machine, chart, state, box, or function, that returns a handle to the Data object for the new data.

Arguments

`parent`	Handle to an object for the parent machine, chart, state, box, or function of the new data

Returns

`data_new`	Handle to the Data object for the new data

Example

If `sA` is a handle to a State object for an existing state `A`, the following command creates a new data parented (contained by) state `A`:

```
data_new = Stateflow.Data(sA)
```

The new data is named `'data'` with an incremented integer suffix to distinguish additional creations. `data_new` is a handle to the Data object for the new data.

Stateflow.EMFunction

Purpose

Constructor for creating Embedded MATLAB function

Syntax

```
efunction_new = Stateflow.EMFunction(parent)
```

Description

The `Stateflow.EMFunction` method is a constructor method for creating an Embedded MATLAB function in a parent Stateflow chart, state, box, or graphical function. It returns a handle to the EMFunction object for the new Embedded MATLAB function.

Arguments

`parent`	Handle to parent chart or state of the new Embedded MATLAB function

Returns

`efunction_new`	Handle to a Function object for the new Embedded MATLAB function

Example

If `sA` is a handle to a State object for the existing state `A`, the following command creates a new Embedded MATLAB function parented (contained by) state `A`:

```
efunction_new = Stateflow.EMFunction(sA)
```

The new Embedded MATLAB function is unnamed and appears in the upper left corner inside of state `A` in the Stateflow Editor. `efunction_new` is a handle to the EMFunction object, which you use to rename the function, set its properties, and execute its methods.

Purpose

Constructor for creating event

Syntax

```
event_new = Stateflow.Event(parent)
```

Description

The `Stateflow.Event` method is a constructor method for creating an event for a parent machine, chart, state, box, or function, that returns a handle to an Event object for the new event.

Arguments

`parent`	Handle to parent machine, chart, state, box, or function of new event

Returns

`event_new`	Handle to the Event object for the new event

Example

If `sA` is a handle to a State object for an existing state `A`, the following command creates a new event parented (contained by) state `A`:

```
event_new = Stateflow.Event(sA)
```

The new event is named `'event'` with an incremented suffix to distinguish additional creations . `event_new` is a handle to an Event object for the new event that you use to rename the event, set its properties, and execute Event methods for the event.

Stateflow.Function

Purpose

Constructor for creating graphical function

Syntax

```
function_new = Stateflow.Function(parent)
```

Description

The `Stateflow.Function` method is a constructor method for creating graphical functions in a parent Stateflow chart or state that returns a handle to a Function object for the new graphical function.

Arguments

`parent`	Handle to parent chart or state of the new graphical function

Returns

`function_new`	Handle to a Function object for the new graphical function

Example

If `sA` is a handle to a State object for the existing state `A`, the following command creates a new graphical function parented (or contained) by state `A`:

```
function_new = Stateflow.Function(sA)
```

The new graphical function is unnamed and appears in the upper left corner inside state `A` in the Stateflow Editor. `function_new` is a handle to the Function object for the new graphical function that you use to rename the function, set its properties, and execute its methods.

Purpose

Constructor for creating junction

Syntax

```
junc_new = Stateflow.Junction(parent)
```

Description

The `Stateflow.Junction` method is a constructor method for creating a junction in a parent Stateflow chart, state, box, or function, that returns a handle to the Junction object for the new junction.

Arguments

`parent`	Handle to the object for the parent chart, state, box, or function of the new junction

Returns

`junc_new`	Handle to the Junction object for new junction

Example

If `sA` is a handle to a State object for the existing state `A`, the following command creates a new junction parented (contained by) state `A`:

```
junc_new = Stateflow.Junction(sA)
```

The new junction appears in the middle of state `A` in the Stateflow Editor. `junc_new` is a handle to the Junction object for the new junction that you use to set its properties, and execute its methods.

Stateflow.Note

Purpose

Constructor for creating note

Syntax

```
note_new = Stateflow.Note(parent)
```

Description

The `Stateflow.Note` method is a constructor method for creating notes for a parent Stateflow chart, state, box, or function, that returns a handle to the Note object for the new note.

Arguments

`parent`	Handle to the object for the parent chart, or subchart for the new note

Returns

`note_new`	Handle to the Note object for the newly created note

Example

If `sA` is a handle to a State object for the existing state `A`, the following command creates a new note parented (contained by) state `A`:

```
note_new = Stateflow.Note(sA)
```

The new note appears in the upper left corner of state `A` in the Stateflow Editor, but is invisible because it has no text content. `note_new` is a handle to the Note object for the new note, that you use to set its text content with a command like the following:

```
note_new.Text = 'This is a note'
```

Purpose

Constructor for creating Simulink function

Syntax

```
sl_function = Stateflow.SLFunction(parent)
```

Description

The `Stateflow.SLFunction` method is a constructor method for creating a Simulink function for a parent Stateflow chart or state that returns a handle to the new Simulink Function object.

Arguments

`parent`	Handle to the object for the parent chart or state for the new Simulink Function object

Returns

`sl_function`	Handle to the newly created Simulink Function object

Example

If `sA` is a handle to a State object for the existing state `A`, the following command creates a new Simulink function parented (contained) by state `A`:

```
sl_function = Stateflow.SLFunction(sA)
```

The new Simulink function appears in the upper left corner of state `A` in the Stateflow Editor. `sl_function` is a handle to the new Simulink function that you can use to rename the function, set its properties, and execute its methods.

Stateflow.State

Purpose

Constructor for creating state

Syntax

```
state_new = Stateflow.State(parent)
```

Description

The `Stateflow.State` method is a constructor method for creating a state for a parent Stateflow chart, state, box, or function, that returns a handle to the State object for the new state.

Arguments

`parent`	Handle to the object for the parent chart, state, box, or function for the new state

Returns

`state_new`	Handle to State object for newly created state

Example

If `sA` is a handle to a State object for the existing state `A`, the following command creates a new state parented (contained) by state `A`:

```
state_new = Stateflow.State(sA)
```

The new state appears in the upper left corner of state `A` in the Stateflow Editor. `state_new` is a handle to the State object for the new state that you use to rename the state, set its properties, and execute its methods.

Purpose

Constructor for creating custom target

Syntax

```
target_new = Stateflow.Target(parent_m)
```

Description

The `Stateflow.Target` method is a constructor method for creating a custom target for a parent machine, that returns a handle to the Target object for the new target.

Arguments

`parent_m`	Handle to object for the parent machine of the new custom target

Returns

`target_new`	Handle to the Target object for the newly created custom target

Example

The following command creates a new custom target for the machine with the Machine object whose handle is `pm`:

```
target_new = Stateflow.Target(pm)
```

The preceding command creates a custom target with the name `untitled`. `target_new` is a handle to the Target object, which you can use to rename and set properties for the custom target. This command renames the new target to `ctarg`:

```
target_new.Name = 'ctarg'
```

Stateflow.Transition

Purpose

Constructor for creating transition

Syntax

```
transition_new = Stateflow.Transition(parent)
```

Description

The `Stateflow.Transition` method is a constructor method for creating transitions in a parent Stateflow chart, state, box, or function that returns a handle to a Transition object for the new transition.

Arguments

`parent`	Handle to parent chart, state, box, or function of new transition

Returns

`transition_new`	Handle to Transition object for the new transition

Example

If `sA` is a handle to a State object for the existing state `A`, the following command creates a new transition parented by state `A`:

```
transition_new = Stateflow.Transition(sA)
```

The new transition is unlabeled and appears in the upper left corner of the chart in the Stateflow Editor. `transition_new` is a handle to the Transition object for the new transition that you use to rename the transition, set its properties, and execute its methods.

Purpose

Constructor for creating truth table

Syntax

```
truth_table_new = Stateflow.TruthTable(parent)
```

Description

The `Stateflow.TruthTable` method is a constructor method for creating truth tables in a parent Stateflow chart, state, box, or function, that returns a handle to a Truth Table object for the new truth table.

Arguments

`parent`	Handle to parent chart or state of new truth table

Returns

`truth_table_new`	Handle to Truth Table object for new truth table

Example

If `sA` is a handle to a State object for the existing state `A`, the following command creates a new truth table parented (contained by) state `A`:

```
truth_table_new = Stateflow.TruthTable(sA)
```

The new truth table is unnamed and appears in the upper left corner inside of state `A` in the Stateflow Editor. `truth_table_new` is a handle to the Truth Table object for the new truth table that you use to rename the truth table, set its properties, and execute its methods.

Purpose

Return MATLAB structure containing property settings of object

Syntax

```
propList = thisObject.struct
```

Description

The `struct` method returns and displays a MATLAB structure containing the property settings of this object.

Note You can change the values of the properties in this structure just as you would a property of the object. However, the MATLAB structure is not a Stateflow object and changing it does not affect the model.

Arguments

`transitions`	The object for which to display property settings. Can be any Stateflow object type.

Returns

`propList`	MATLAB structure listing the properties of this object

Example

If State object `sA` represents a state `A`, the command `x = sA.struct` returns a MATLAB structure `x`. You can use dot notation on `x` to report properties or set the values of other variables. For example, the command `y=x.Name` sets the MATLAB variable `y` to the value of the `Name` property of state `A`, which is `'A'`. The command `x.Name = 'Kansas'` sets the `Name` property of `x` to `'Kansas'` but does not change the `Name` property of state `A`.

Purpose

Return parent of object

Syntax

```
parentObject = thisObject.up
```

Description

The `up` method returns a handle to the parent of this object.

Arguments

`thisObject`	Object for which to return parent (containing) object

Returns

`parentObject`	Object containing `thisObject`

Example

Assume that a Stateflow chart has two states, `A` and `B`, and state `A` contains state `B`. If the object `sB` represents the state `B`, then the command

```
p = sB.up
```

returns a handle `p` to the parent of `B`, which is state `A`.

view

Purpose

Make object visible for editing

Syntax

```
thisObject.view
```

Description

The `view` method opens the Stateflow object in its appropriate editing environment as follows:

- For Chart objects, the `view` method opens the chart in the Stateflow Editor, if it is not already open, and brings it to the foreground.
- For State, Box, Function, Note, Junction, and Transition objects, the `view` method does the following:
  1. Opens the chart containing the object in the Stateflow Editor if it is not already open.
  2. Highlights the object.
  3. Zooms the object's Stateflow Editor window to the level of full expanse of the object's containing state or chart.
  4. Brings the Stateflow Editor window for this object to the foreground.
- For Truth Table objects, the `view` method opens the Truth Table Editor for this truth table.
- For Embedded MATLAB Function objects, the `view` method opens the Embedded MATLAB Editor for this function.
- For Simulink Function objects, the `view` method shows the contents of the function-call subsystem.
- For Event, Data, and Target objects, the `view` method opens the Model Explorer.

Arguments

`thisObject` Object for which to display editing environment.

Returns

None

zoomIn and zoomOut

Purpose

Zoom in or out on Stateflow chart

Syntax

```
thisEditor.zoomIn
thisEditor.zoomOut
```

Description

The methods `zoomIn` and `zoomOut` cause the Stateflow Editor for a chart to zoom in or zoom out, respectively, by 20 percentage points.

Note The `zoomIn` and `zoomOut` methods do not open or give focus to the Stateflow Editor for the chart.

Arguments

`thisEditor` Editor object on which to zoom in or out.

Returns

None

Example

If the Editor object `ed` represents a Stateflow Editor for a chart with the zoom level at 100%, the command `ed.zoomIn` raises the zoom level to 120%.

Index

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

Z