Stateflow® Release Notes

MATLAB®&SIMULINK®

MathWorks®

How to Contact MathWorks

	Latest news:	www.mathworks.com
	Sales and services:	www.mathworks.com/sales_and_services
	User community:	www.mathworks.com/matlabcentral
	Technical support:	www.mathworks.com/support/contact_us
	Phone:	508-647-7000

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

Contents

R2021a

R2020b

R2019a

R2018b

R2018a

R2017b

R2017a

R2016b

R2016a

R2015aSP1

Bug Fixes

R2015b

R2015a

R2014b

R2014a

R2013b

R2013a

R2012b

R2012a

R2011b

R2011a

R2010bSP2

Bug Fixes

R2010bSP1

Bug Fixes

R2010b

R2010a

R2009bSP1

Bug Fixes

R2009b

R2009a

R2008b

R2008a+

Bug Fixes

R2008a

R2007b+

Bug Fixes

R2007b

R2007a+

Bug Fixes

R2007a

R2006b

R2006a+

No New Features or Changes

R2006a

R14SP3

R14SP2

R2022b

Version: 10.7

New Features

Bug Fixes

Compatibility Considerations

Generate more efficient code by using in-place data

In R2022b, you can improve the performance and decrease the memory footprint of the generated code for your Stateflow charts, truth tables, and state transition tables by using in-place data. You create in-place data when you use the same data name for a chart input and a chart output. When you generate code from the chart, the generated code treats the input and output data as a single in-place argument passed by reference. Using in-place data reduces the number of times that the generated code copies intermediate data, which results in more efficient code.

Enhancements to Stateflow programmatic interface

In R2022b, the Stateflow programmatic interface has new object functions and properties.

- `Stateflow.AtomicSubchart` and `Stateflow.AtomicBox` objects have new object functions that you can use to edit the mapping of subchart symbols:

  - `getMappingForSymbol`
  - `setMappingForSymbol`
  - `clearMappingForSymbol`
  - `disableMappingForSymbol`

- `Stateflow.StateTransitionTableChart` objects have a new object function, `exportAsStruct`, that exports the contents of state transition tables as structure arrays.

- `Stateflow.Box` objects have a new property, `ExecutionOrder`, that specifies the execution order for substates in parallel (AND) decomposition.

Functionality being removed or changed

New keyboard shortcut for state transition tables
Behavior change

The keyboard shortcut to append a transition column to a State Transition Table is now **Ctrl+K**. In previous releases, the shortcut was **Ctrl+M**.

R2022a

Version: 10.6

New Features

Bug Fixes

Compatibility Considerations

Use Symbols pane while writing MATLAB function code

Starting in R2022a, you can view the **Symbols** pane while you write MATLAB® function code. In previous releases, you could use the **Symbols** pane to edit MATLAB function variables only from the Stateflow Editor.

The **Symbols** pane displays the variables that you define in the signature label for the MATLAB function. After you make changes to the code, refresh the contents of the **Symbols** pane by saving the model, updating the model, returning to the chart canvas, or clicking a different pane.

Manage Stateflow breakpoints in the Simulink Breakpoints List pane

Starting in R2022a, you can use the new breakpoints list in the Simulink® Editor to manage breakpoints in your Stateflow chart. In previous releases, these breakpoints appear only in the Stateflow Breakpoints and Watch window. For more information, see Unified breakpoint list in the Simulink Editor for debugging.

Expanded string support for charts that use MATLAB as the action language

When using MATLAB as the action language, you can include:

- String types with the operators: `forward`, `hasChangedFrom`, and `hasChangedTo`.
- SDI with string type data. For more information about SDI, see Simulation Data Inspector (Simulink).
- String type data in Truth Table blocks.
- String type data in State Transition Table blocks.

Additionally, string truncation in charts that use MATLAB as the action language is on par with charts that use C as the action language.

Repeat data names at different levels of the chart hierarchy

Starting in R2022a, Stateflow charts that use MATLAB as the action language support data objects with the same name at different levels of the chart hierarchy. For example, you can now use the name of a chart symbol as an argument or return value for a graphical function, MATLAB function, or truth table. In previous releases, using the same data name in different levels of the chart hierarchy results in a compile-time error.

Export chart-level Simulink functions

Starting in R2022a, you can export chart-level Simulink functions in Stateflow charts by enabling the chart property **Export chart level functions**. In previous releases, enabling this property in a chart that contains chart-level Simulink functions results in a run-time error.

Convert states that contain supertransitions to atomic subcharts

You can now convert states to atomic subcharts, even when supertransitions cross the boundary of the state. Converting a state to an atomic subchart automatically replaces any supertransition into or

out of the state with a transition that connects to an entry or exit port. Entry and exit ports enable your chart to transition across boundaries in the Stateflow hierarchy while isolating the logic for entering and exiting the atomic subchart. For more information, see Convert a State or Normal Subchart to an Atomic Subchart, Create Entry and Exit Connections Across State Boundaries, and Isolate the Transition Logic for Entering and Exiting an Atomic Subchart.

Note that you cannot undo the conversion to an atomic subchart. You can convert the atomic subchart back to a normal subchart, as described in Convert an Atomic Subchart to a Normal Subchart, but this action does not replace the new entry and exit ports with supertransitions.

Convert machine-parented data to chart-parented data store memory

Use the Upgrade Advisor check Check for machine-parented data (Simulink) to convert machine-parented data to chart-parented data of scope `Data Store Memory`. For more information, see Consult the Upgrade Advisor (Simulink). Machine-parented data will no longer be supported in a future release.

Functionality being removed or changed

Use strong data typing with Simulink I/O chart property has been removed
Errors

In R2022a, the chart property **Use strong data typing with Simulink I/O** has been removed. Data types of input signals to charts and state transition tables must now match the type of the corresponding Stateflow data object. Otherwise, a type mismatch error occurs.

R2021b

Version: 10.5

New Features

Bug Fixes

Compatibility Considerations

Create entry and exit connections across hierarchy boundaries

In R2021b, you can now use ports and junctions to create connections across boundaries in the Stateflow hierarchy. Entry and exit ports improve componentization by isolating the transition logic for entering and exiting states. Unlike supertransitions, they can be used in atomic subcharts.

In the Stateflow Editor, entry and exit ports appear as arrows on the boundary of a state or subchart. Each port has a matching junction that marks the entry or exit point inside the state or subchart. The entry junction icon ● and the exit junction icon ⊗ indicate the junction. A transition path that leads to an entry port continues along the transition connected to the matching entry junction. Similarly, a transition path that leads to an exit junction continues along the transition connected to the matching exit port.

For example, in the model `sf_boiler`, the exit port labeled `turnOff` represents the exit connection out of the subchart `On`.
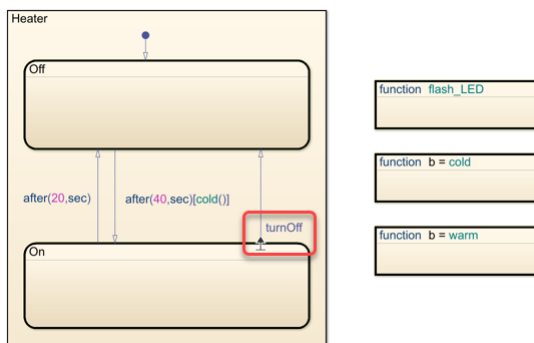
In the subchart, the transition path leading to the exit junction defines the logic for exiting the subchart. In this example, the function `warm` must evaluate to `true` on two consecutive time steps before the chart makes the transition out of the `On` state.
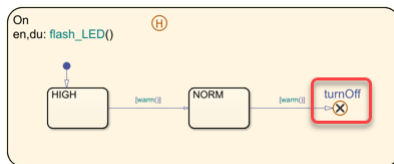
For more information on entry and exit ports, see Create Entry and Exit Connections Across State Boundaries. For more information about this example, see Model Bang-Bang Temperature Control System. For other examples that use entry and exit ports, see Isolate the Transition Logic for Entering and Exiting an Atomic Subchart and Model a Launch Abort System.

Detect rising and falling edges in data expressions

Stateflow charts in Simulink models can now detect rising and falling edges in data expressions. The new edge detection operators take scalar-valued expressions and return a Boolean output.

Operator	Syntax	Description	Example
crossing	`tf = crossing(expression)`	Returns 1 (`true`) if: • The previous value of `expression` was positive and its current value is zero or negative. • The previous value of `expression` was zero and its current value is nonzero. • The previous value of `expression` was negative and its current value is zero or positive. Otherwise, the operator returns 0 (`false`). This operator imitates the behavior of a Trigger (Simulink) block with **Trigger Type** set to `either`.	Transition out of state if the value of the input data `signal` crosses a threshold of 2.5. `[crossing(signal-2.5)]` The edge is detected when the value of the expression `signal-2.5` changes from positive to negative, from negative to positive, from zero to nonzero, or from nonzero to zero.
falling	`tf = falling(expression)`	Returns 1 (`true`) if: • The previous value of `expression` was positive and its current value is zero or negative. • The previous value of `expression` was zero and its current value is negative. Otherwise, the operator returns 0 (`false`). This operator imitates the behavior of a Trigger (Simulink) block with **Trigger Type** set to `falling`.	Transition out of state if the value of the input data `signal` falls below a threshold of 2.5. `[falling(signal-2.5)]` The falling edge is detected when the value of the expression `signal-2.5` changes from positive to negative, from positive to zero, or from zero to negative.

Operator	Syntax	Description	Example
rising	`tf = rising(expression)`	Returns 1 (`true`) if: • The previous value of `expression` was negative and its current value is zero or positive. • The previous value of `expression` was zero and its current value is positive. Otherwise, the operator returns 0 (`false`). This operator imitates the behavior of a Trigger (Simulink) block with **Trigger Type** set to `rising`.	Transition out of state if the value of the input data `signal` rises above a threshold of 2.5. `[rising(signal-2.5)]` The rising edge is detected when the value of the expression `signal-2.5` changes from negative to positive, from negative to zero, or from zero to positive.

Like the Trigger block, these operators detect a single edge when the `expression` argument changes value from positive to zero to negative or from negative to zero to positive at three consecutive time steps. The edge occurs when the value of the expression becomes zero.

The arguments to these operators can combine input data, constants, nontunable parameters, continuous-time local data, and state data from Simulink based states. Arguments can include addition, subtraction, and multiplication of scalar variables, elements of a matrix, fields in a structure, or any valid combination of structure fields and matrix indices. Indices can be numbers or expressions that evaluate to a constant integer. For more information, see Detect Changes in Data and Expression Values.

MATLAB Function block editor in Stateflow window

Starting in R2021b, the MATLAB Function Block Editor opens in the same Stateflow window as the parent chart of the MATLAB Function block. Previously, when you opened a function in the MATLAB Function Block Editor, the editor opened in the MATLAB window. The new MATLAB Function Block Editor enables you to:

- Easily navigate from the MATLAB function back to the parent chart in the same window.
- Access and edit properties of the chart while viewing the MATLAB function on the canvas.
- Run and step the chart while debugging the MATLAB function in the MATLAB Function block debugger.
- Edit your function using the tools that were previously available in the MATLAB Function Block Editor.
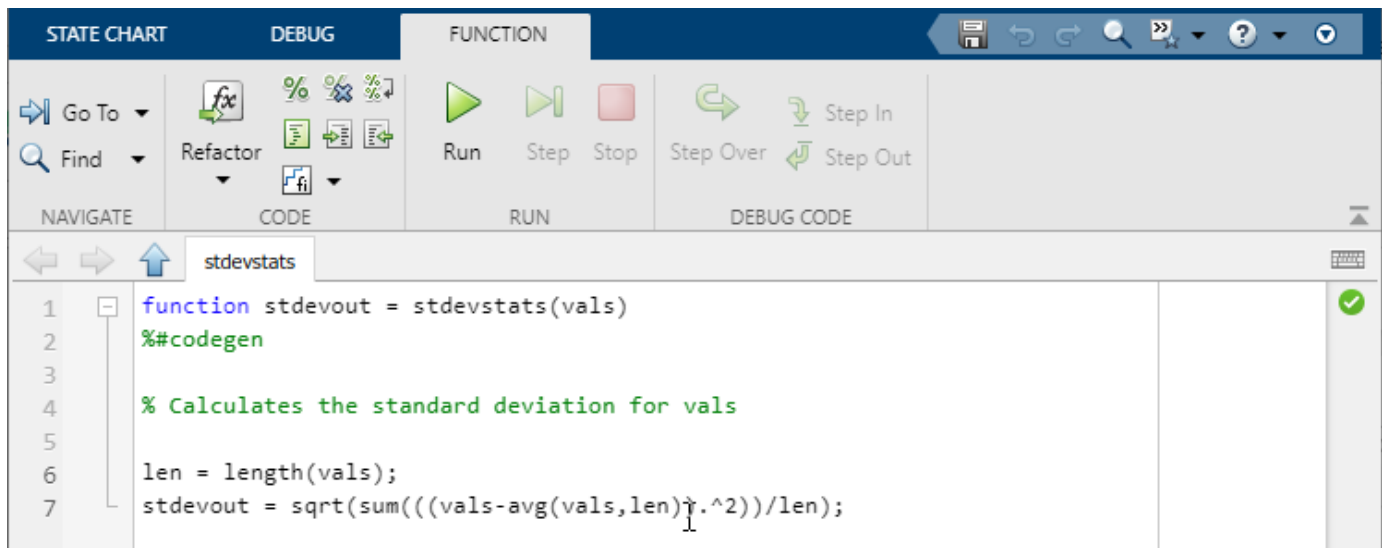
To open the editor, in a Stateflow chart, double-click a MATLAB Function block. The **Function** tab opens and the canvas shows the MATLAB function. To navigate back to the chart, use the navigation buttons ⬅ ➡ ⬆ .

Index and assign values to arrays of structures in C action language

Starting in R2021b, you can use arrays of structures in Stateflow charts and state transition tables that use C as the action language. To access and modify the contents of an array of structures, use dot notation and numeric indices. For example, this list illustrates how to access the elements of an array of structures by using zero-based indexing delimited by brackets:

- `structArray[0]` — First element of the array of structures `structArray`
- `structArray[0].a` — Field `a` of the structure `structArray[0]`
- `structArray[0].a.b` — Field `b` of the substructure `structArray[0].a`
- `structArray[0].a.b[2][3]` — Element in the third row, fourth column of the field `b` of substructure `structArray[0].a`

For more information on accessing and modifying the contents of a structure or an array of structures, see Index and Assign Values to Stateflow Structures.

String support for charts that use MATLAB as the action language

You can create and manipulate string data in a Stateflow chart that uses MATLAB as the action language. The string data type is compatible with strings in MATLAB and Simulink. For more information about using string data, see Manage Textual Information by Using Strings.

Navigate using the Stateflow miniature map

When you zoom in or out of your Stateflow chart, a miniature map appears in the bottom left hand corner of the Stateflow editor. As you zoom or navigate your Stateflow chart, the blue square will move on the mini map, indicating your location relative to the entire chart.
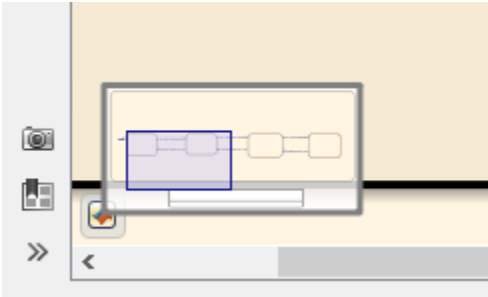
For more information, see Zoom and Navigate with the Miniature Map.

Functionality being removed or changed

Stateflow charts no longer support creating machine-parented data
Errors

Starting in R2021b, Stateflow charts do not support creating machine-parented data. In the Model Explorer, you cannot select the Add Data ▣ button or the **Add** > **Data** menu option at the Stateflow machine level. In addition, calling the `Stateflow.Data` function with arguments of type `Stateflow.Machine` now results in an error.

The presence of machine-parented data in a model prevents the reuse of generated code and other code optimizations. This type of data is also incompatible with many Simulink and Stateflow features. To make Stateflow data accessible to other charts and blocks in a model, use data store memory. For more information, see Best Practices for Using Data in Charts and Access Data Store Memory from a Chart.

Machine-parented data will no longer be supported in a future release.

Stateflow truth tables no longer generate content

Starting in R2021b, you no longer need to generate content for Stateflow truth tables to set breakpoints and debug. For more information on setting break points in a truth table, see Debug Errors in a Truth Table.

R2021a

Version: 10.4

New Features

Bug Fixes

Edit the color of data syntax highlighting

In Stateflow charts, truth tables, and state transition tables, you can edit the color of data syntax highlighting based on its scope. This highlighting can be applied to:

- Local data
- Constant data
- Inputs
- Outputs
- Parameters
- Data Store Memory data

To change the color of a specific data, in the **Format** tab, select **Style > Syntax Highlighting**. For more information, see Stateflow Editor Operations.

Add Stateflow chart behavior to an architecture component

In R2021a, you can add Stateflow chart behavior to a component in a System Composer™ architecture model. Describe component behavior using state charts with Stateflow to represent modes of operation.

- Add a Stateflow chart behavior to your component.
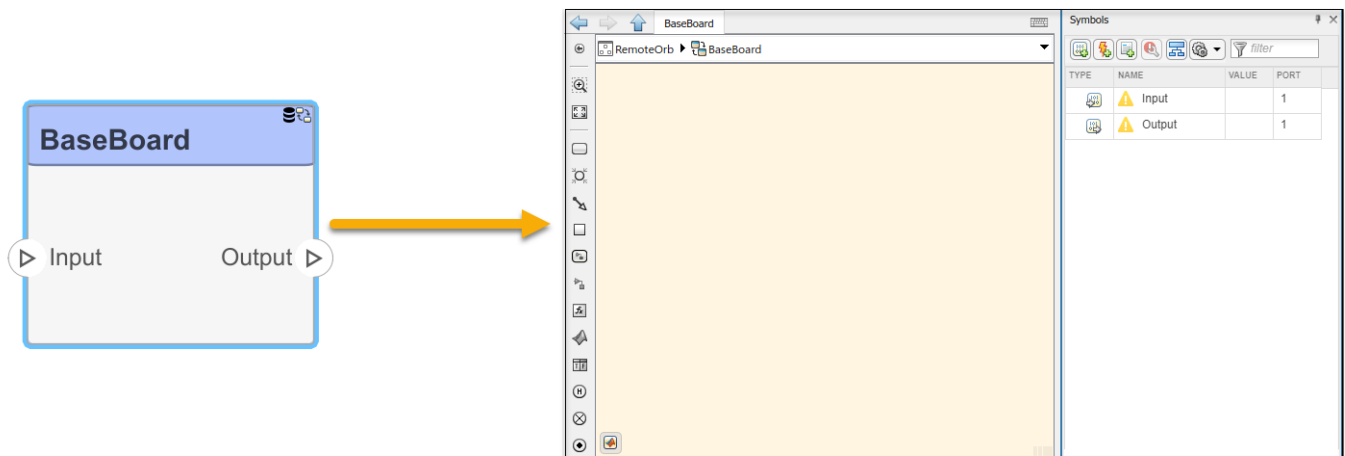
- The new Stateflow chart behavior for a component is embedded within the same `.slx` file as the parent architecture model, reducing the need for multiple model files.

For more information, see Add Stateflow Chart Behavior to Architecture Component (System Composer).

64-bit integer type support for parameters

Parameters in Stateflow charts can now be set as 64-bit integer data. For more information, see Differences Between MATLAB and C as Action Language Syntax.

Half-precision data type support

Stateflow charts that use MATLAB as the action language can now use half-precision type data. A half-precision data type occupies 16 bits of memory, but its floating-point representation enables it to handle wider dynamic ranges than integer or fixed-point data types of the same size. For more information, see The Half-Precision Data Type in Simulink (Fixed-Point Designer).

Multidimensional variable support for row-major arrays

In charts that use C as the action language, you can use multidimensional custom code variables. To implement row-major as the default array layout for functions, open the Configuration Parameters dialog box. In the **Simulation Target** pane, click **Import custom code**. In the **Code Generation > Interface** pane, under the **Data exchange interface** section, ensure that **Array layout** is set to `Row-major`.

Reverse transitions

If a Stateflow transition can be reversed, you can reverse the transition without deleting your transition and attached conditions or actions. To reverse a transition in a Stateflow chart, hover over the transition. Move the cursor over the ellipses that appear above the transition and click the

reverse transition button ⟲ .
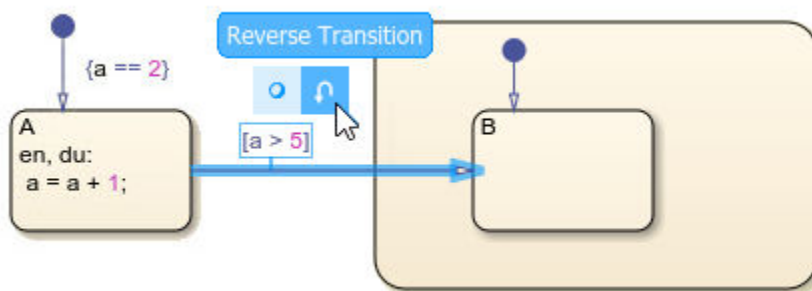
Insert components without leaving the Stateflow canvas

In R2021a, you can use quick insert to add Stateflow components to the Stateflow canvas. To insert a component, double-click the Stateflow canvas and type the name of the component you want to add. A list appears with possible components to add to your Stateflow chart.

4-4
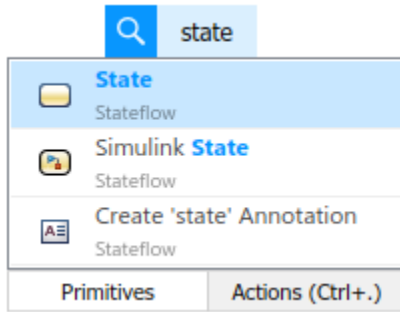
R2020b

Version: 10.3

New Features

Bug Fixes

Compatibility Considerations

Visualize chart behavior with the Activity Profiler

The Activity Profiler visually represent the usage of states, transitions, and functions in your Stateflow chart. With the Activity Profiler enabled, the states, transitions, and functions in your chart are highlighted after simulation to show how often Stateflow accessed the object during simulation.
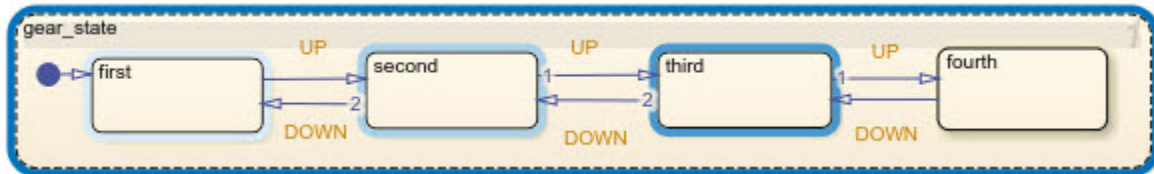
You can also view this data in a table by using the Activity Profiler pane, which appears when you enable heat maps.

To enable the Activity Profiler, in the Stateflow Editor, in the **Debug** tab, click **Activity Profiler**. For more information, see Visualize Chart Execution with the Activity Profiler.

Design state machines to control MATLAB apps

By using the new keyword `this`, you can establish a bidirectional connection between an external MATLAB function or app and a Stateflow chart in a Simulink model. With this connection, you can use a chart to define the behavior of a MATLAB app created in App Designer. The chart monitors your interactions with the app and enables or disables app widgets accordingly. For examples that illustrate this workflow, see Model a Power Window Controller and Simulate a Media Player.

Compatibility Considerations

If you have functions or variables named `this`, change their name to avoid conflicting with the keyword.

Connect dashboard blocks to Stateflow

In R2020b, you can connect Dashboard blocks to Stateflow, including:

- Charts
- States
- Simulink based states
- Subcharts
- Atomic subcharts

Dashboard blocks can observe the self, child, and leaf activity of the connected state or chart. Additionally, you can connect Dashboard blocks to Stateflow local and output data by selecting states or transitions from the Stateflow Editor. To connect a chart to Dashboard blocks, double-click the block in the Simulink Editor and select the Stateflow chart. For more information on using Dashboard blocks, see Control Simulations with Interactive Displays (Simulink).

Execute standalone charts saved in earlier versions of Stateflow

You can now share a Stateflow standalone chart that was saved in R2020a with collaborators that have R2020b. If your collaborators have the same or a later version of MATLAB than you have, they

can execute your standalone charts as MATLAB objects without opening the Stateflow Editor. For more information, see Share Standalone Charts.

Compatibility Considerations

To run standalone charts that you saved in R2019a or R2019b, your collaborators must have the same version of MATLAB.

Programmatically extract actions from states and transitions

The API objects `Stateflow.State` and `Stateflow.Transition` now have read-only properties that help you extract the text of state and transition actions.

API Object	Property	Type	Description
Stateflow. State	DuringAction	Character vector	Text in the `during` action in this state. This property is not supported in Moore charts.
	EntryAction	Character vector	Text in the `entry` action in this state. This property is not supported in Moore charts.
	ExitAction	Character vector	Text in the `exit` action in this state. This property is not supported in Moore charts.
	MooreAction	Character vector	Text in the action in this state. This property is supported only in Moore charts.
	OnAction	Cell array of character vectors	Text in the `on` actions in this state, parsed as a cell array of this form: `{'trigger1','action1',...,'triggerN','actionN'}` This property is not supported in Moore charts.
Stateflow. Transition	Condition	Character vector	Text in the condition on this transition.
	ConditionAction	Character vector	Text in the condition action on this transition.
	TransitionAction	Character vector	Text in the transition action on this transition.
	Trigger	Character vector	Text in the trigger on this transition.

The values of these properties are set by the `LabelString` property for the state or transition. For more information, see Specify Labels in States and Transitions Programmatically.

Multidimensional custom code function support for row-major

In charts that use C as the action language, you can include data and message inputs for multidimensional custom code functions with row-major as the array layout. To implement row-major as the default array layout for inputs in functions, open the Configuration Parameters dialog box. In the **Simulation Target** pane, click **Import custom code** and set **Default function array layout** to `Row-major`.

You can also specify individual functions for row-major array layout. In the **Simulation Target** pane, click **Specify by function**. From this window, you can add or remove functions and specify their individual array layout.

Use the Sequence Viewer in the toolstrip to visualize message flow, function calls, and state transitions

The Sequence Viewer in the toolstrip allows you to visualize messages, functions calls, and state transitions without using the Sequence Viewer block in your model.

To activate logging, on the **Simulation** tab, in the **Prepare** section, select **Log Events**.
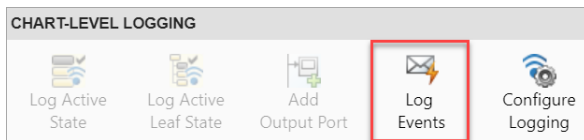
To visualize the simulation results, go to the **Review Results** section and select the Sequence Viewer.
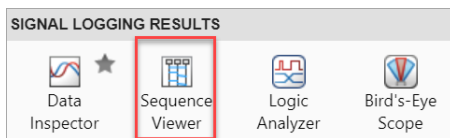
Generated default switch cases determined alphabetically

In code generated from Stateflow charts that include exclusive (OR) states, the default `case` of `switch` expressions corresponds to the child state whose name is last in alphabetical order.

R2020a

Version: 10.2

New Features

Bug Fixes

Compatibility Considerations

Generate code for variant software configurations

With variant transitions, you can create Stateflow charts in Simulink models that generate code that may be used in a variety of different software situations. Variant transitions use chart parameters in a condition and attach to states within your chart that are variations from the core chart configuration. To change a transition to a variant transition, click the transition that you want to change. In the **Transition** tab, select **Variant Transition**. The transition appears on the chart with a # symbol, which indicates that the transition is a variant transition.
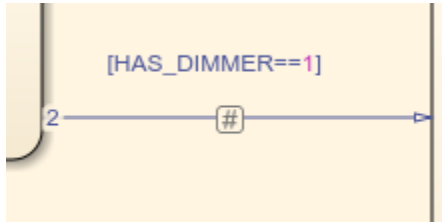
Once a variant transition is implemented in your chart, the code that you generate may include either:

- The portions of the code that are currently enabled.
- Preprocessor conditional statements that guard different configurations.

To generate the preprocessor conditional statement, open the Property Inspector. In the Stateflow editor, on the **Modeling** tab, under **Design**, select **Property Inspector**. Under **Advanced**, select **Generate preprocessor conditionals**.

64-bit integer type support for charts that use MATLAB as the action language

Stateflow charts that use MATLAB as the action language now support 64-bit integer data. Charts implement `int64` and `uint64` data types as fixed-point numbers with a word length of 64 bits and a fraction length of 0. For more information, see Differences Between MATLAB and C as Action Language Syntax .

Cache and report compilation warnings

In R2020a, Stateflow caches chart warnings and displays them when you update an unmodified chart.

Multidimensional array indexing for constant, Data Store Memory, and message data

If you have Embedded Coder®, you can generate code that preserves the multidimensionality of Stateflow constant data, Data Store Memory data, and messages without flattening the data as one-dimensional arrays. For example, consider this matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

By default, with row-major layout, the code generator flattens the matrix to a one-dimensional array:

```
{1, 2, 3, 4, 5, 6}
```

To preserve the dimensions of your array, use the Embedded Coder Code Mappings editor. For more information, see Preserve Dimensions of Multidimensional Arrays in Generated Code (Embedded Coder).

Now you can implement the matrix to a two-dimensional array:

```
{{1, 2, 3}, {4, 5, 6}}
```

To preserve Stateflow data array dimensions, you must first select row-major layout. For more information, see Select Array Layout for Matrices in Generated Code.

Absolute-time temporal logic operators for standalone charts in MATLAB

In standalone Stateflow charts in MATLAB, you can now use the operators `after`, `at`, and `every` to wake up the chart based on absolute-time conditions. Standalone charts define absolute-time temporal logic in terms of wall-clock time, which is limited to 1 millisecond precision. For more information, see Control Chart Execution by Using Temporal Logic.

Event queuing semantics in standalone charts in MATLAB

In R2020a, standalone Stateflow charts in MATLAB use new queuing semantics when a chart is busy processing another operation. If a chart is busy when it receives an input event or an implicit event associated with an absolute-time temporal logic operator, the chart queues the event. The event is executed when the current step is completed. You can specify the size of the event queue by setting the configuration option `-eventQueueSize` when you create the chart object. For more information, see Events in Standalone Charts.

Compatibility Considerations

Execution behavior has changed when a standalone chart receives an event while it is processing another operation.

Before R2020a	R2020a
New events interrupt the current activity of the chart. When the chart finishes executing the new event, it returns to the activity that was taking place before the interruption. The results of processing the new event can conflict with the action that was taking place before the event was generated, leading to unexpected behavior. For example, a chart can finish executing the interrupted `during` actions of a state after the state becomes inactive.	New events are queued and do not interrupt the current activity of the chart. The chart executes events in the order they are received.

Export standalone Stateflow charts for execution in earlier versions of MATLAB

You can now export a standalone chart to a format used by an earlier version of Stateflow so that collaborators with earlier versions of the software can use your charts. You can only export back to

R2019a and later releases. To complete the export process, you need access to the versions of Stateflow from which and to which you are exporting. Using the later version of Stateflow, open the standalone chart and select **Save > Previous Version**. Alternatively, you can export the chart by calling the function `Stateflow.exportToVersion`. Then, using the earlier version of Stateflow, open and resave the exported chart. For more information, see Share Standalone Charts.

Functionality being removed or changed

Behavior in charts with 64-bit fixed-point type inputs could change
Behavior change

The Simulink data type `fixdt(1,64,0)` was previously shown as `sfix64` on the Simulink canvas. When inputting this signal into a Stateflow chart that uses MATLAB as the action language it was treated as an `embedded.fi` object internally regardless of the **Treat these inherited Simulink signal types as fi objects** setting.

In R2020a, Stateflow now treats these inputs as either `int64` or an `embedded.fi` object depending on the setting. When **Treat these inherited Simulink signal types as fi objects** is set to `Fixed-point`, `fixdt(1,64,0)` is treated as an int64 data type. When **Treat these inherited Simulink signal types as fi objects** is set to `Fixed-point & Integer`, `fixdt(1,64,0)` is treated as an `embedded.fi` object.

This change may lead to a difference in chart behavior when set to `Fixed-point`.

To preserve the behavior of previous releases in your Stateflow chart, use the function `cast64BitIntToFi`.
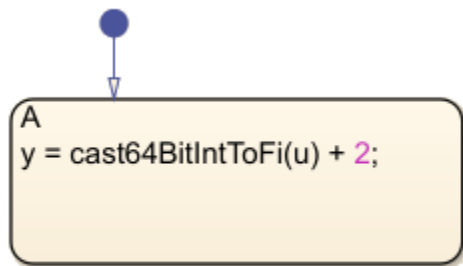
This behavior also applies to `ufix64` data types.

R2019b

Version: 10.1

New Features

Bug Fixes
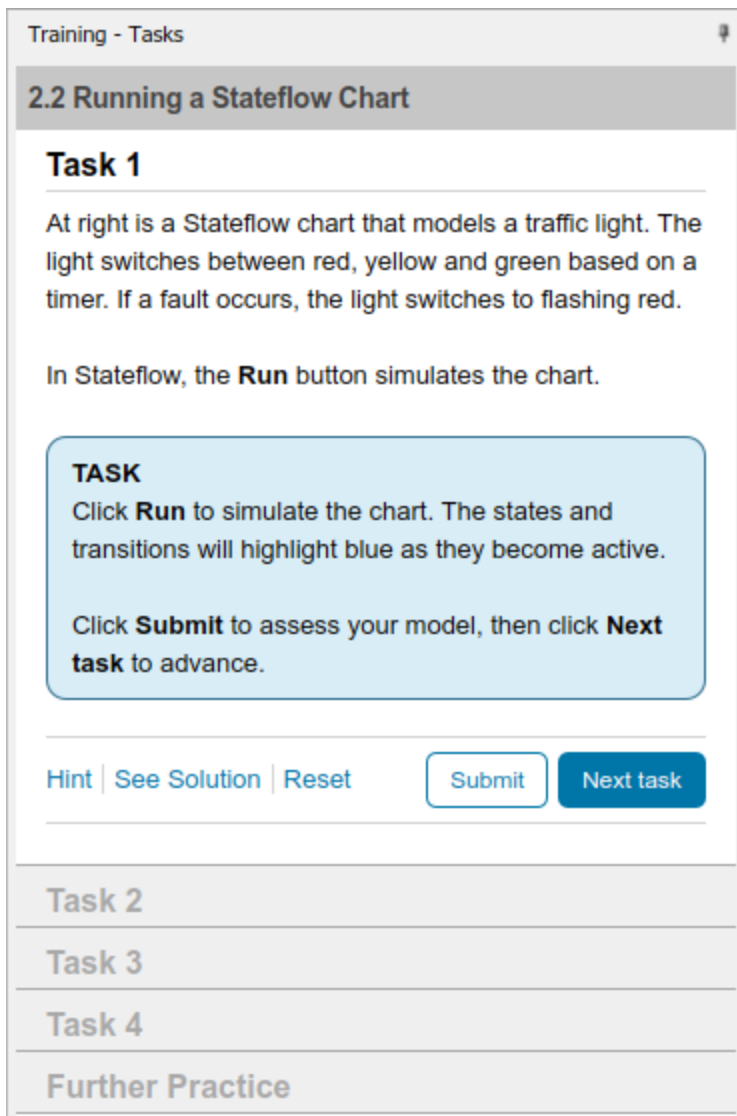
Compatibility Considerations

Stateflow Onramp: Self-paced, interactive tutorial for getting started with Stateflow

To help you get started quickly with Stateflow basics, Stateflow Onramp provides a self-paced, interactive tutorial. After completing Stateflow Onramp, you will be able to use the Stateflow environment and build Stateflow charts based on real-world examples.

To open Stateflow Onramp, on the Simulink Start Page, you can select the ⬚ **Stateflow Onramp** button under **Learn**.

To teach concepts incrementally, Stateflow Onramp uses hands-on exercises.

You receive automated assessments and feedback after submitting tasks.
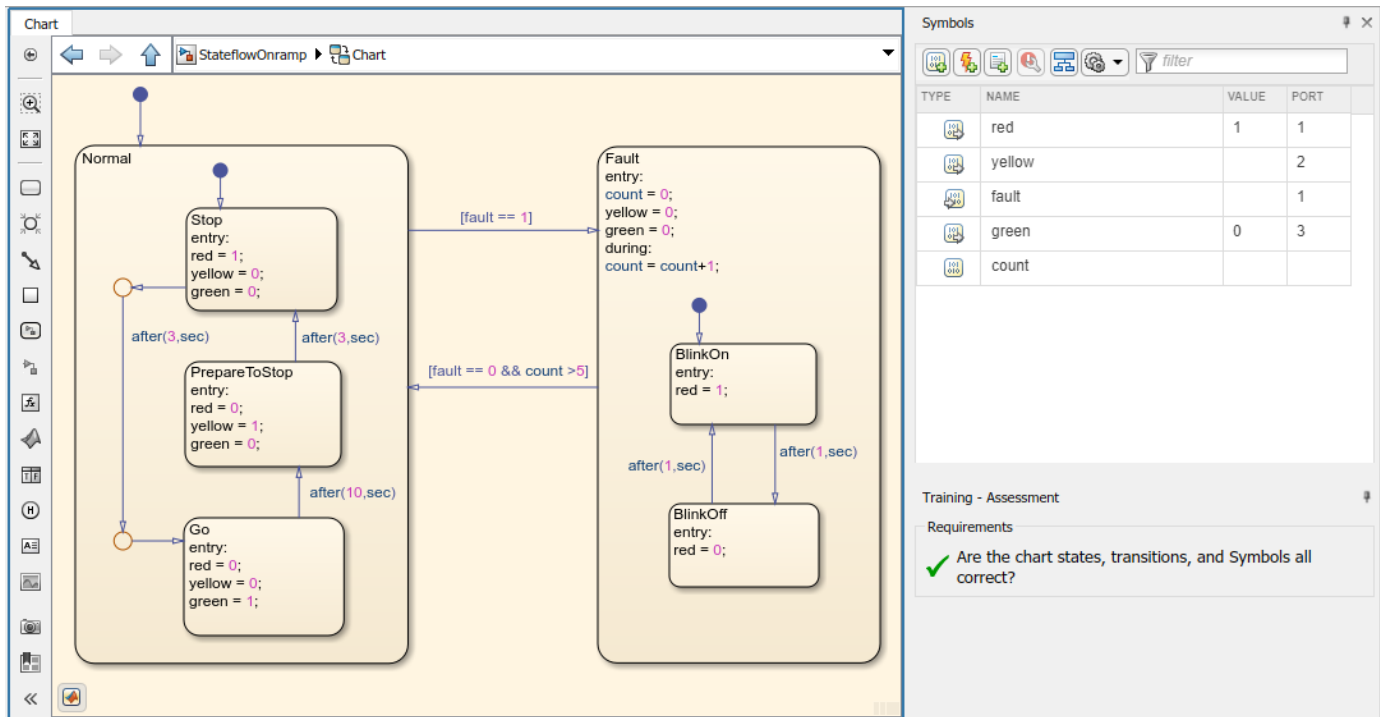
Your progress is saved if you exit the application, so you can complete the training in multiple sessions.

Stateflow Onramp covers these topics:

- State machines
- Creating state charts
- Stateflow symbols and data
- Chart actions
- Chart execution
- Flow charts
- Functions in Stateflow
- Chart hierarchy

Stateflow Onramp helps you practice what you learn with these projects:

- Robotic Vacuum
- Robotic Vacuum Driving Modes

Simulink Toolstrip: Access Stateflow capabilities by using contextual tabs

In R2019b, the Simulink Toolstrip replaces Simulink menu bar. For more details, see Simulink Toolstrip: Access and discover Simulink capabilities when you need them (Simulink). The location of several Stateflow features and buttons are located in contextual tabs. The Simulink Toolstrip contextual tabs appear only when you need them.

- To access the **State Chart** tab, click a Stateflow chart in a Simulink model.
- To access the **State** tab, click a state in a Stateflow chart.

Stateflow Editor Changes

- "Mapping from Simulink Editor to the Simulink Toolstrip" on page 7-4
- "Chart Menu" on page 7-4

Mapping from Simulink Editor to the Simulink Toolstrip

The following tables list the new Simulink Toolstrip items that are different from the Stateflow Editor **Chart** menu bar items. Many of the features and options that were previously hidden within the toolbar menus are now directly available from tabs on the Simulink Toolstrip. For a complete mapping of all Simulink Editor menu bar items, see Simulink Editor Changes.

Chart Menu

Menu Bar Item	Toolstrip Equivalent
Parse Chart	**Debug > Update Model > Update Chart**
Refresh Blocks	(Ctrl+K)
Group & Subchart > • **Group** • **Subchart** • **Atomic Subchart**	Select a state. **State >** • **Group** • **Subchart** • **Atomic Subchart**
Add Inputs & Outputs > • **Data Input From Simulink** • **Data Output To Simulink** • **Event Input From Simulink** • **Event Output To Simulink** • **Message Input From Simulink** • **Message Output To Simulink**	**Modeling > Design Data** gallery **>** • **Data Input** • **Data Output** • **Event Input** • **Event Output** • **Message Input** • **Message Output**
Add Other Elements > • **Parameter** • **Local Data** • **Constant** • **Data Store Memory** • **Local Event** • **Local Message**	**Modeling > Design Data** gallery **>** • **Parameter** • **Local Data** • **Constant** • **Data Store** • **Event** • **Message**
Add Pattern In Chart	**Modeling > Pattern**
Save Pattern	**Modeling > Pattern > Save As Pattern**
Create Subchart from Selection	Select a state. **Modeling > Subchart Selection**
Create Superstate from Selection	Select a state. **Modeling > Add Superstate**

Menu Bar Item	Toolstrip Equivalent
Create Subcharted Box from Selection	Select a state. **Modeling > Subcharted Box**
Create Box from Selection	Select a state. **Modeling > Add Box**
Format > Font Style	**Format**
Format > Font Size	**Format**
Format > Text Alignment	Double-click the text. Use the pop-up menu.
Format > • **Enable TeX Commands** • **Shadow** • **Arrowhead Size** • **Junction Size** • **.Content Preview**	**Format >** • Select annotation. **Enable Equations** Σ • Select text. **Shadow** • Select transition. **Arrowhead** drop-down. • Select junction. **Junction** drop-down. • Select subchart or atomic subchart. **Content Preview**.
Arrange	Select the states to be arranged.**Format >** • **Align** • **Distribute** • **Match**
Library Link > • **Go to Library Block** • **Disable Link** • **Resolve Link** • **View Changes**	Select library state. **State >** • **Go to Library** • **Disable Link** • **Restore Link** • **View Changes**
Decomposition > • **Exclusive** • **Parallel**	**Modeling > Decomposition** • **Exclusive** • **Parallel**
Execution Order	Select parallel state. **Subchart > Execution Order**
Subchart Mappings	Select atomic subchart. **Atomic Subchart > Mappings**
Properties	**Modeling > State Properties**

Flow Charts from MATLAB: Visualize MATLAB scripts and functions as Stateflow flow charts

In standalone Stateflow charts, you can use the Pattern Wizard to transform your MATLAB code into flow charts and graphical functions. Supported patterns for conversion include:

- `if`, `if-else`, and other nested decision statements
- `for` and `while` loops
- `switch` statements

For more information, see Convert MATLAB Code into Stateflow Flow Charts.

64-bit integer types int64 and uint64

Stateflow charts that use C as the action language now support 64-bit integer data. Charts implement `int64` and `uint64` data types as fixed-point numbers with a word length of 64 bits and a fraction length of 0.

- `int64` is an alias type for `fixdt(1,64,0)`.
- `uint64` is an alias type for `fixdt(0,64,0)`.

For more information, see Fixed-Point Data in Stateflow Charts.

Change detection in standalone Stateflow charts

Standalone Stateflow charts now support the operators `hasChanged`, `hasChangedFrom`, and `hasChangedTo`. You can use these operators to detect changes in the values of local data when you execute a chart in MATLAB.

Debugging enhancements for standalone Stateflow charts in MATLAB

While debugging standalone charts, you can now select different configurations of breakpoint types:

- States support `On State Entry`, `During State`, and `On State Exit` breakpoints.
- Transitions support `When Transition is Tested` and `When Transition is Valid` breakpoints.

Conditional breakpoints are also supported. To specify a condition for the breakpoint, use a valid MATLAB expression that combines numerical values and Stateflow data objects that are in scope at the breakpoint. For more information, see Debug a Standalone Stateflow Chart.

Enhanced support of row-major data in Stateflow blocks

Use row-major array layout in Stateflow blocks to more easily and efficiently integrate your Simulink model with row-major data and algorithms. Row-major layout is now supported in Stateflow charts, state transition tables, and truth table blocks, including:

- Charts that use MATLAB as the action language.
- Charts that contain truth table functions and MATLAB functions.

- Charts that use custom C code where all custom variables and arguments to custom functions are scalars, vectors, or structures of scalars and vectors. Specify the size of an *n*-element vector as *n*, and not as [*n* 1] or [1 *n*].

For more information, see Select Array Layout for Matrices in Generated Code.

External receiving queues for input messages

Input messages can now connect to receiving queues that you configure outside of a Stateflow chart. When you disable the **Use Internal Queue** property for an input message, you can connect the message input port to:

- A Queue block that manages an external queue in your Simulink model
- A root-level Inport block that enables messages to cross the model boundary

For more information, see Use Internal Queue.

Message delivery in debugging mode

While debugging a Stateflow chart, you can test the design of the chart by sending local and output messages. For example, suppose that M is a local message. While the simulation is paused at a breakpoint, you can change the value of the data field and send the message to its local queue. In the MATLAB Command Window, enter:

```
M = 5;
send(M);
```

To see the effects of sending the message, advance to the next step of the simulation.

Follow these rules when sending messages while debugging a chart:

- To read or write to the message data field of a valid message, use the name of the message object. Do not use dot notation syntax.
- You can send a message from the debugging prompt only when the chart explicitly sends the message by calling the send operator.
- You cannot send input messages from the debugging prompt.

For more information, see Send Messages by Using the Debugging Prompt.

Propagation of symbolic dimensions for Stateflow data

When you select the model configuration parameter **Allow symbolic dimension specification**, charts that use C as the action language can propagate the symbolic dimensions of Stateflow data throughout the model. If you have Embedded Coder, the symbolic dimensions go into the generated code for ERT targets. Specify the size of the symbolic dimensions by using Simulink parameters with one of these storage classes:

- Define or ImportedDefine with a specified header file
- CompilerFlag
- A user-defined custom storage class that defines data as a macro in a specified header file

For more information, see Propagate Symbolic Dimensions of Stateflow Data.

Compatibility Considerations

Behavior for ERT code generation has changed when you specify the size of a Stateflow data object by using a Simulink parameter with a storage class that is not supported for symbolic dimensions.

Before R2019b	R2019b
In the generated code, the symbolic dimensions are replaced by their constant values. No error or warning occurs.	Code generation results in an error. To resolve the error: • Change the storage class for the Simulink parameter. • In the Model Configuration Parameters dialog box, clear the **Allow symbolic dimension specification** check box.

Stateflow cache file support for code generation and Simulink

Simulink cache files now support code generation and Stateflow artifacts. The cached artifacts can reduce the time required for successive simulation and code generation. Caching occurs automatically when you simulate models in accelerator or rapid accelerator mode, or generate code for models. When Simulink cache files are available in the Simulation cache folder (Simulink), simulation and code generation automatically extract the relevant artifacts from the Simulink cache file. To share Simulink cache files with team members, you can store them in a network location.

For more information on Simulink cache files, see Share Build Artifacts for Faster Simulation and Code Generation (Simulink).

Zoom in Truth Tables

To zoom in on a truth table, press **Ctrl++ (Command++)**.

To zoom out on a truth table, press **Ctrl+- (Command+-)**.

To return to normal view (100%) on a truth table, press **Ctrl+0 (Command+0)**.

Functionality being removed or changed

Use dot notation to access message data in MATLAB functions and truth tables
Behavior change

In R2019b, use dot notation syntax to read or write to the message data field in a MATLAB function or a truth table function.

Before R2019b	R2019b

In MATLAB functions or truth table functions, use the name of a message to access the data field for the message. For example to read the data for message M and store it as the variable y, enter:
 	
`y = M;`
 	
To change the value of the data field and send the message, enter:
 	
`M = u;`
`send(M);`
 	
If the data for M is a structure with fields a and b, store the values of these fields by entering:
 	
`ya = M.a;`
`yb = M.b;`
	Use dot notation to access the data field for the message. For example, to read the data for message M and store it as the variable y, enter:
	
	`y = M.data;`
	
	To change the value of the data field and send the message, enter:
	
	`M.data = u;`
	`send(M);`
	
	If the data for M is a structure with fields a and b, store the values of these fields by entering:
	
	`ya = M.data.a;`
	`yb = M.data.b;`

For more information, see Control Message Activity in Stateflow Charts.

Transition execution order is always visible
Behavior change

In R2019b, transition execution order numbers are always visible in a Stateflow chart.

Log multiple signals
Behavior change

In R2019b, the Stateflow Signal Logging dialog box is no longer available. To log multiple signals from your Stateflow chart, press and hold shift to select the states for logging. In the **Simulation** tab, under **Prepare**, select **Log Self Activity**.

Search

CHART-LEVEL LOGGING

Log Active State · Log Active Leaf State · Add Output Port · Configure Logging

STATE LOGGING

Log Self Activity · Log Child Activity · Log Leaf States

Log Self Activity - Log activity of selected state

CONFIG...

Model Settings · Symbols Pane · Property Inspector · Chart Properties · Update Chart · Update Model

CODE GENERATION FOR STATES

Test Point

Step Forward · Stop · Data Inspector · Logic Analyzer · Bird's-Eye Scope

REVIEW RESULTS

UP · third · UP · fourth

DOWN · DOWN · DOWN

selection_state
during: [down_th,up_th] = calc_th(gear,throttle);

[speed < down_th] · steady_state · [speed > up_th]

[speed > down_th] · [speed < up_th]

downshifting · upshifting

after(TWAIT,tick)
[speed <= down_th]
{send(gear_state.DOWN)}

after(TWAIT,tick)
[speed >= up_th]
{send(gear_state.UP)}

Simulink Function
[down_th,up_th] = calc_th(gear,throttle)

Opening Stateflow
Behavior change in future release

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

R2019a

Version: 10.0

New Features

Bug Fixes

Stateflow Charts in MATLAB: Graphically program, debug, and execute standalone state machines as MATLAB objects

Create a Stateflow chart outside of a Simulink model. Save the standalone chart with the new extension `.sfx` and execute it as a MATLAB object. With standalone charts, you can create MATLAB applications such as:

- MATLAB App Designer user interfaces that use mode logic to manage the behavior of widgets.
- Communication protocols and data stream processing applications that use sequential logic.
- Data Acquisition Toolbox™ or Instrument Control Toolbox™ applications that use timer-based logic to monitor and control external tasks.

These applications can be shared and executed without requiring a Stateflow license. For more information, see Create Stateflow Charts for Execution as MATLAB Objects.

Truth Table Breakpoints: Check Truth Table logic by setting breakpoints and stepping though Truth Table simulation

Set breakpoints in a Stateflow Truth Table to check decision logic when Stateflow:

- Tests a condition.
- Tests a decision.
- Marks a decision as valid.
- Executes an action.

To set a breakpoint, right-click the corresponding area where the breakpoint is located. For example, to set a breakpoint when a condition is tested, right-click the number of the condition. In the contest menu select **Set Breakpoint**. Check the current data values by hovering over a condition or action cell.

Truth Table breakpoint conditions allow breakpoint to pause simulation once the condition is met. To set a breakpoint condition, click the breakpoint and add a condition under **Breakpoint condition**.

To disable a breakpoint, click the breakpoint and clear the **Enable Breakpoint** check box. To disable all breakpoints, right click a cell in the Truth Table and select **Configure Breakpoints > Disable All Breakpoints**.

To clear a breakpoint, click the breakpoint and clear the **Set Breakpoint** check box. To clear all breakpoints, right click a cell in the Truth Table and select **Configure Breakpoints > Clear All Breakpoints**.

Custom Code Symbols: Examine values when debugging a chart

While debugging your Stateflow charts, you can now view the values of your custom code symbols. When simulation pauses, if you point at a state or transition in the chart, a tooltip displays the value of custom variables that the object uses. The tooltip also displays chart data, messages, and temporal logic expressions. For more information, see Watch Data in the Stateflow Chart.

Change detection for buses and matrices

Stateflow charts can detect changes in value for:

- Scalar variables
- Matrices or elements of a matrix
- Structures or fields in a structure

For more information, see Detect Changes in Data Values.

Enhanced subchart mapping capabilities

When mapping variables in an atomic subchart or atomic box, you can type an expression that specifies:

- A field of a Stateflow structure
- An element of a vector or matrix

When referring to elements of a vector or matrix, regardless of the action language of the chart, use:

- One-based indexing delimited by parentheses and commas. For example, `A(4,5)`.
- Zero-based indexing delimited by brackets. For example, `A[3][4]`.

Indices can be numbers or parameters in the chart. Other expressions are not supported as indices. For more information, see Map Variables for Atomic Subcharts and Boxes.

Optimized counters for temporal logic

Temporal logic operators produce integer or fixed-point type counters in generated code. The size of the counter is optimized based on the operator and the type of threshold.

Relaxed restrictions on Moore charts

Transitions in Moore charts can contain condition and transition actions if these actions do not introduce a dependency between output values and input values. For more information, see Design Considerations for Moore Charts.

State machine logic control by using the count operator

The `count(C)` operator returns a double value equivalent to the number of ticks after the conditional expression `C` becomes `true`. The `count` operator is reset if the conditional expression becomes `false`. If the `count` operator is used within a state, it is reset when the state that contains it is entered. If the `count` operator is used on a transition, it is reset when the source state for that transition is entered. For more information, see count and Control Chart Execution by Using Temporal Logic.

Stateflow contextual tabs in the Simulink Toolstrip

In R2019a, you have the option to turn on the Simulink Toolstrip. See Simulink Toolstrip Tech Preview replaces menus and toolbars in the Simulink Desktop for more details.

The Simulink Toolstrip includes contextual tabs — they appear only when you need them. The Stateflow contextual tabs include options for completing actions that apply only to Stateflow.

- To access the **State Chart** tab, click a Stateflow chart within a Simulink model.
- To access the **State** tab, click a State within a Stateflow chart.

R2018b

Version: 9.2.0.0

New Features

Bug Fixes

Compatibility Considerations

Simulation Debugger: Check chart logic with simplified breakpoint management, statement-by-statement stepping, and in-canvas visualization of data and time

- **Simplified management of breakpoints:** Now there is a single menu option for setting breakpoints on state or transitions. For states, the default breakpoint is `On State Entry` and `During State`. For transitions, the default breakpoint is `When Transition is Valid`. For more information, see Set a Breakpoint for a Stateflow Object.

- **Statement by statement debugging:** In state or transition actions containing more than one statement, you can now step through the individual statements one at a time by selecting **Step Over**. Stateflow highlights each statement before executing it. To execute a group of statements together, right-click the last statement in the group and select **Run To Cursor**. For more information, see Control Chart Execution After a Breakpoint.

- **Data values in the Symbols window:** When in debugging mode, the values of each data are displayed in the **VALUE** column of the Symbols pane. When the debugger is stopped at a breakpoint, you can change the value of a symbol in either the command prompt or the Symbols pane. The value column highlights changes to data values as the changes occur.

  In the Symbols pane multidimensional arrays appear as the data type and size of the array. If the array does not exceed more than 100 elements, hover over the symbol to view the elements. For arrays that contain more than 100 elements, view the elements by using the command prompt.

  For other non-scalar objects, the size and data type appear. To see these values, use the Watch window. See Watch Stateflow Data Values and Manage Stateflow Breakpoints and Watch Data.

- **Introspection of temporal operators:** When simulation pauses, if you point at a state or transition in the chart, a tooltip now displays the value of the temporal logic expressions that the selected object uses. For more information, see Watch Data in the Stateflow Chart.

External C Code: Fully integrate external C code in Stateflow charts with change synchronization, error checking, and analysis by Simulink Coverage and Simulink Design Verifier

Use your custom C code in Stateflow charts that use MATLAB as the action language. Directly call functions and variables from custom C code without using `coder.ceval` or `coder.opaque`. Access your custom code while using Just-In-Time Compilation mode. When you make changes to your custom code, Stateflow charts automatically recompile. Errors messages display for incorrect custom code symbols usage, such as when calling a function with the wrong number of variables. Custom codes symbols appear in the **Explore** menu for quick navigation. For more information, see Custom Code Algorithm.

Row-Major Array Layout: Define the array layout as row-major to simplify integration with external C/C++ functions, tools, and libraries

When generating code from a C action language chart, you can specify the array layout for matrices. For example, consider this matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

By default, the code generator uses column-major layout to store the matrix in memory with this arrangement:

{1, 4, 2, 5, 3, 6}

Now, you can select row-major layout to store the matrix in memory with this arrangement:

{1, 2, 3, 4, 5, 6}

Row-major layout is not supported in:

- Charts and state transition table blocks that use MATLAB as the action language.
- Charts that contain truth table functions that use MATLAB as the action language.
- Charts that contain MATLAB functions.
- Charts that use custom C code.
- Truth table blocks.

For more information, see Select Array Layout for Matrices in Generated Code.

Strings: Design embedded systems with native support of strings

Now, you can create and manipulate string data in a Stateflow chart that uses C as the action language. The new string data type is compatible with strings in MATLAB and Simulink.

To manipulate string data in a chart, use the operators listed in this table.

Operator	Syntax	Description	Example
strcpy	dest = src	Assigns string `src` to `dest`.	Assigns string data to `s1` and `s2`: s1 = 'hello'; s2 = "good bye";

	strcpy(dest,src)	An alternative way to execute `dest = src`.	Assigns string data to `s3` and `s4`:
			
			strcpy(s3,'howdy');
			strcpy(s4,"so long");

strcat	dest = strcat(s1,...,sN)	Concatenates strings `s1,...,sN`.	Concatenates strings to form "Stateflow": s1 = "State"; s2 = "flow"; dest = strcat(s1,s2);
substr	dest = substr(str,i,n)	Returns the substring of length `n` starting at the `i`-th character of string `str`. Use zero-based indexing.	Extracts substring "Stateflow" from a longer string: str = "Stateflow rule the waves"; dest = substr(str,0,9);

Operator	Syntax	Description	Example

tostring	dest = tostring(X)	Converts numerical, Boolean, or enumerated data to string.	Converts numerical value to string "1.2345":
			
			dest = tostring(1.2345);
			
			Converts Boolean value to string "true":
			
			dest = tostring(1==1);
			
			Converts enumerated value to string "RED":
			
			dest = tostring(RED);

strcmp	tf = strcmp(s1,s2)	Compares strings s1 and s2. Returns 0 if the two are identical. Otherwise returns a nonzero integer that depends on the input strings and the compiler that you use.
		
		Strings are considered identical when they have the same size and content.
		
		This operator is consistent with the C library function strcmp. The operator behaves differently than the function strcmp in MATLAB.
			Returns a value of 0 (strings are equal):
			
			tf = strcmp("abc","abc");
			
			Returns a nonzero value (strings are not equal):
			
			tf = strcmp("abc","abcd");

	s1 == s2	An alternative way to execute strcmp(s1,s2) == 0.	Returns a value of true: "abc" == "abc";
	s1 != s2	An alternative way to execute strcmp(s1,s2) != 0.	Returns a value of true: "abc" != "abcd";
	tf = strcmp(s1,s2,n)	Returns 0 if the first n characters in s1 and s2 are identical.	Returns a value of 0 (substrings are equal): tf = strcmp("abc","abcd",3);
strlen	L = strlen(str)	Returns the number of characters in the string str.	Returns a value of 9: L = strlen("Stateflow");

Operator	Syntax	Description	Example
str2double	X = str2double(str)	Converts the text in string str to a double-precision value. str contains text that represents a number. Text that represents a number can contain: • Digits • A decimal point • A leading + or - sign • An e preceding a power of 10 scale factor If str2double cannot convert text to a number, then it returns a NaN value.	Returns a value of -12.345: X = str2double("-12.345"); Returns a value of 123400: X = str2double("1.234e5");
str2ascii	A = str2ascii(str,n)	Returns array of type uint8 containing ASCII values for the first n characters in str, where n is a positive integer. Use of variables or expressions for n is not supported.	Returns uint8 array {72,101,108,108,111}: A = str2double("Hello",5);
ascii2str	dest = ascii2str(A)	Converts ASCII values in array A of type uint8 to string.	Returns string "Hi!": A[0] = 72; A[1] = 105; A[2] = 33; dest = ascii2str(A);

For more information, see Manage Textual Information by Using Strings.

Messages: Produce strictly typed, readable, and MISRA-C Mandatory and Required check compliant code from messages

In R2018b, messages, queues, and related data structures are strictly typed to improve code quality and enable verification and validation workflows. Generated code is more compact and readable. For Stateflow charts sending or receiving messages, Mandatory and Required MISRA-C checks have zero violations.

C action language in state transition tables

State transition tables support using C as the action language. For more information about the differences between these action languages, see Differences Between MATLAB and C as Action Language Syntax.

To set C as the action language, in the Stateflow editor, select **Chart > Properties**.

Under **Action Language**, select C from the drop-down list.

Custom code headers for enumerated data and buses

For imported enumerated data and buses, the header file is not required in the **Simulation Target** pane in the Configuration Parameters dialog box.

Multidimensional array indexing in generated code

If you have Embedded Coder, you can generate code that preserves the multidimensionality of Stateflow local data without flattening the data as one-dimensional arrays. For example, consider this matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

By default, with row-major layout, the code generator flattens the matrix as a one-dimensional array:

{1, 2, 3, 4, 5, 6}

Now, by selecting the configuration parameter **Preserve Stateflow local data array dimensions**, you can implement the matrix as a two-dimensional array:

{{1, 2, 3}, {4, 5, 6}}

To preserve Stateflow local data array dimensions, you must first select row-major layout. For more information, see Select Array Layout for Matrices in Generated Code.

Pass-by-reference semantics in functions

In graphical functions, in truth table functions, and in MATLAB functions in Stateflow charts, you can use the same variable name for both input and output. For example, this MATLAB function uses the variables y1 and y2 as both inputs and outputs:

```
function [y1, y2, y3] = f(y1, u, y2)
    y1 = y1 + 1;
    y2 = y2 + 1;
    y3 = u + y2;
end
```

If you export this function to C code, y1 and y2 are passed by reference (as pointers) and u is passed by value. Passing inputs by reference reduces the number of times that the generated code copies intermediate data, resulting in more optimal code.

Pass-by-reference semantics are supported in Simulink function blocks but not in Simulink functions in Stateflow charts.

Functionality being removed or changed

Stateflow charts that integrate custom code may need to turn off option Import Custom Code in the Configuration Parameters
Behavior change

When you import custom code to a Stateflow chart, you may need to turn off the **Import Custom Code** option in the Simulation pane of the Configuration Parameters. Prior to R2018b, this option was called **Parse Custom Code Symbols**. For more information see Import custom code (Simulink).

R2018a

Version: 9.1

New Features

Bug Fixes

Compatibility Considerations

Truth Table Editor: Design combinatorial logic within the Simulink and Stateflow editing environment by using edit-time checking, animation, and step-by-step debugging

Truth tables are fully integrated into the Stateflow editing environment. To add and manage data to your truth table, use the Symbols pane. To modify the properties of your truth table, use the Property Inspector. You can drag, cut, copy, and paste multiple rows or columns with multi-selection capabilities. As you modify your truth table, the generated content is automatically updated.

Just-In-Time Debugger: Set breakpoints and debug Stateflow charts while using Just-In-Time simulation

You can run Stateflow charts with debugging support while in Just-In-Time (JIT) mode. JIT mode improves the model update performance of your Stateflow charts. See Speed Up Simulation.

Implicit entry,during action type for unspecified state actions

If you do not specify the state action type explicitly for a statement, the chart treats that statement as an `entry,during` action.

Compatibility Considerations

Chart behavior for unspecified state actions has changed.

Before R2018a	R2018a
Charts using MATLAB as the action language did not allow unspecified state actions. The Stateflow editor added the state action type `entry` to any unspecified state actions.	**Charts using MATLAB as the action language** allow unspecified state actions, treating them as `entry,during` type actions.
Charts using C as the action language allowed unspecified state actions, treating them as `entry` type actions.	**Charts using C as the action language** allow unspecified state actions, treating them as `entry,during` type actions.

To allow for backward compatibility in R2018a:

- To preserve the original chart behavior, opening C action language charts that you saved in a previous version adds the `entry` label to all unspecified state actions. The change does not affect MATLAB action language charts that you saved in a previous version.

- Exporting a chart to a previous version adds the `entry,during` label to all unspecified state actions.

Input events for atomic subcharts

To use an atomic subchart that has been saved in a library, but not use the entire saved set of input events, you can disable input events. Under the **Mappings** tab in the **Properties** dialog box, you can disable input events that you do not need. To disable an input event, under **Input Event Mapping**, select `<disabled>` from the drop-down list.

R2017b

Version: 9.0

New Features

Bug Fixes

Simulink Subsystem as a Stateflow State: Design states by using continuous and periodic Simulink algorithms to model hybrid systems

In Stateflow, you can model hybrid dynamic systems, which include continuous or periodic Simulink algorithms embedded in Stateflow states. Use Simulink state reader and state writer blocks or the Stateflow action language to initialize the state of Simulink blocks when you transition from one Simulink based state to another.

You can also use Simulink based states linked to a subsystem in a library model. When you update the library subsystem, the changes are reflected in all Stateflow charts containing the block.

Sequence Viewer: Visualize state changes, event activity, and function calls over time

The Message Viewer block has been renamed the Sequence Viewer block. With this block, you can visualize state changes, event activities, and function calls in your Stateflow chart.

State and Data Visualization: Stream state activity and data directly from Stateflow to the Simulation Data Inspector

You can directly log and view self, child, and leaf state activity for your Stateflow chart by using the Simulation Data Inspector.

Transition Syntax Cues: Create transition labels using syntax cues

While you are editing your chart, use action language cues to create an event, condition, or an action on a transition. After creating a transition, Stateflow shows cues to create a condition or an action. Click a cue and begin typing to add a condition or action.
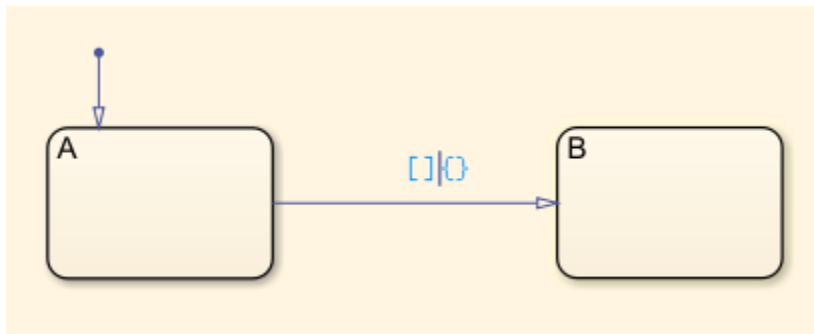
If your chart includes an event or a message, an additional cue for the event or message appears to the left of the condition cue.
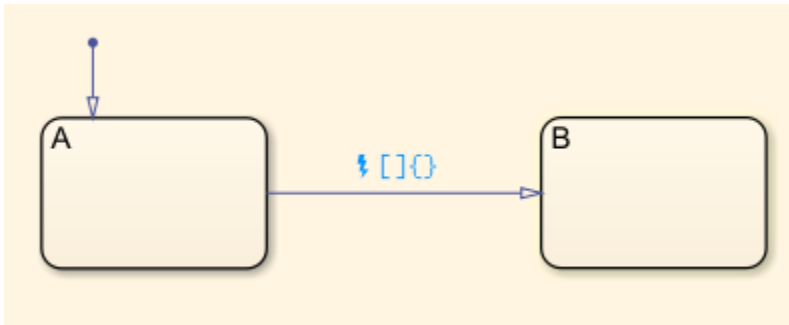
Symbols pane preferences

To control when objects and symbols in your chart are highlighted, use the preferences button ⚙ ▾ in the Symbols pane.

Conversion of Switch-Case statements with parameters

When generating Switch-Case statements from an If-Elseif-Else flow chart, parameters that are used on the right hand side of a condition are preserved.

Local data initialization

For local data, you can set the initial value to be resolved to a Simulink parameter. Before you initialize local data to a Simulink parameter, click **Allow initial value to resolve to a parameter** in the Property Inspector. See `Simulink.Parameter` (Simulink).

Scoped Simulink functions

In Stateflow, you can call external scoped Simulink functions by using qualified dot notation. See Scoping Simulink Functions in Subsystems (Simulink).

R2017a

Version: 8.9

New Features

Bug Fixes

Stateflow Layout: Automatically improve chart readability

With Arrange Automatically, Stateflow arranges your charts to:

- Expand states and transitions to fit their label strings.
- Resize similar states to be the same size.
- Align states if they were slightly misaligned.
- Straighten transitions.
- Reposition horizontal transition labels to the midpoint.

To format your chart, select **Chart > Arrange > Arrange Automatically**.

Temporal Logic Operators: Express state machine logic more concisely by using the duration and the elapsed operators

In 2017a, you can use the operators `duration` and `elapsed` to implement new temporal logic capabilities for Stateflow. These expressions evaluate simulation time of your Stateflow chart.

Operator	Syntax	Description	Example
elapsed	elapsed(sec)	Returns the simulation time in seconds (`sec`) that has elapsed since the activation of the associated state.	Assign to `y` the length of time since the state has been active in seconds: `en, du: y = elapsed(sec);`
duration	duration(condExp)	Returns the seconds after the conditional expression, `condExp`, becomes `true`, within the statement time step.	Return `true` if the time in seconds since `Phi > 1` is greater than 50: `duration(Phi > 1) > 50`

Message Operations: Manage messages and analyze message queues with the keywords discard, length, isvalid, and receive

You can use the keywords `discard`, `isvalid`, `length`, and `receive` for Stateflow messages.

- `receive`(M): equivalent to message on *M* in state actions or message guard *M* on transitions. This function returns true if a valid message M exists, or a new message can be removed from the associated queue.
- `length`(M): returns the number of messages in the queue associated with *M*. To determine how many spaces are left in the queue, subtract the length from the queue capacity.
- `isvalid`(M): returns true if message *M* is valid. A message is valid when it is removed from the queue but has not been forwarded or discarded.

  `discard`(M): discards a valid message. After you discard a message, you can remove a new message from the queue within the same time step.

Editing cues for creating junctions and states

While you are editing your chart, use cues to quickly create a junction or state. When you draw a transition from a junction or state, a cue appears at the end of the junction. The cue shows the outline of a junction and a state. Initially, the junction is highlighted.
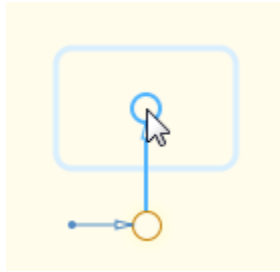
To create the junction, click the highlighted cue. To select a state, move the cursor until the state cue is highlighted.
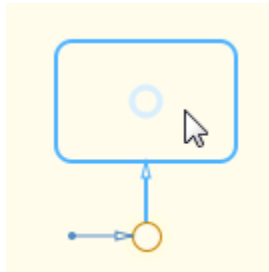
Then, click the state cue.

Automatic port generation

Ports can now be automatically generated for your Stateflow chart. Create a signal and drag it to a supported block that already has all its inports and output ports connected. Automatic port generation is supported in the following blocks:

- Charts
- Truth Tables
- State Transition Tables

Automatic correction of variable type assignment errors

Within your Stateflow chart that uses MATLAB as the action language, variable type assignment errors between MATLAB and Stateflow are automatically corrected for constants and constant expressions. For example, this Stateflow chart shows the variable `var1` being initialized to the integer value 3. Prior to 2017a, when assigning a variable you had to cast the data type in the assignment.
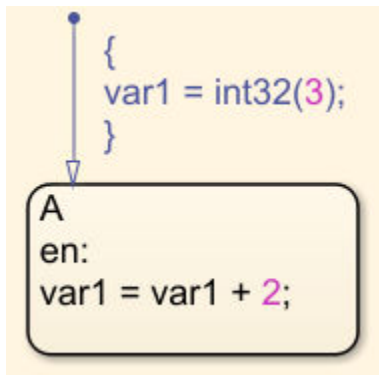
In 2017a, when assigning a variable to a constant or constant expression you do not have to specify the data type. The type of the right side of your equation is propagated to the left side for constants and constant expressions.
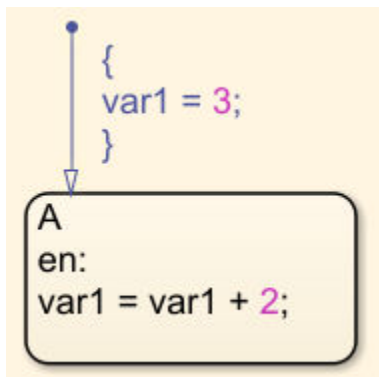
Reduce use of coder.extrinsic

To declare functions that are not supported for code generation, use `coder.extrinsic` only once within a chart. Atomic subcharts in a chart that has already declared a function with `coder.extrinsic` must reuse `coder.extrinsic`.

Zoom in State Transition Tables

To zoom in and out of your State Transition Table, select **View > Zoom**. At any zoom level, you can drag and drop objects, print, and resize your rows and columns. New items are added in to your State Transition Table at the zoom level that is consistent with your chart. When you save your model, the zoom level is also saved.

Absolute-time temporal logic code generation

For some absolute-time constructs using fixed-point parameters, Stateflow generates more efficient code that does not contain floating point operations.

For example, consider `after(DELAY, sec)` in a chart with a sample time of the chart < 1 second where DELAY is a fixed-point parameter. Previously the code generator created the following code:

```
counter >= (uint32_T)ceil((real_T)DELAY * 0.05 / 0.1 - 1e-9)
```

Now, it generates:

```
(counter >> 1) >= DELAY
```

This code contains fewer operations and does not include floating-point operations.

State behavior specification for Truth Table blocks with function-call input events

If you define a function-call input event for a Truth Table block, you can now specify the state behavior when this event reenables the block. To specify this behavior, use the **States When Enabling** block parameter located in the **Properties > Advanced** section of the Property Inspector.

When you set **States When Enabling** to `Held`, the simulation maintains the most recent values of the states when the function-call event reenables the Truth Table block. If you set **States When Enabling** to `Reset`, the function-call event reverts states to their initial conditions.

R2016b

Version: 8.8

New Features

Bug Fixes

Compatibility Considerations

Edit-Time Checking: Detect and fix potential issues in charts at design time

Identify modeling issues while you edit charts with edit-time checking. Edit-time checking provides visual cues for Stateflow errors and warnings. Objects are highlighted in red or orange on the chart to alert you to issues with your model. To display information about the issue, hover your cursor over a highlighted object and click the error or warning icon. The Stateflow editor highlights these issues:

- Syntax errors on states or transitions
- Transition action precedes a condition action along this path
- Dangling transitions
- Invalid default transition
- Default transition is missing
- Transition shadowing
- State not reachable on the execution path
- Unexpected backtracking
- Transition loops outside natural parent
- Invalid transitions crossing into or out of a graphical function

After you click the error or warning icon, a window appears showing suggestions for you to fix the issue. When possible, click **Fix** for Stateflow to apply a fix. To turn off the edit-time checking, select **Display** > **Error & Warnings**. See Modeling Rules That Stateflow Detects During Edit Time.

Symbol Manager: Create and manage data, events, and messages directly in the Stateflow Editor

In the Symbol manager, you can view and manage data, events, and messages while working in the Stateflow editor. To open the Symbols pane, select **View** > **Symbols**. From the Symbols pane you can:

- Add and delete data, events, and messages.
- Set the object type and scope.
- Change the port number.
- Edit the name of an object and update all instances of the object name in the chart.
- Undo and redo changes in type, name, and port number.
- Detect unused objects.
- Detect and fix unresolved objects.
- Trace between objects in the window and where the objects are used in the chart.
- View and edit object properties in the Property Inspector.

Use the **Edit** menu to undo and redo many symbol property changes. When you select an object in the Symbols pane, Stateflow highlights uses of the object in the chart. See Trace Data, Events, and Messages with the Symbols Window. When you rename an object, select **Ctrl+Enter** to rename all instances of the object. To open the Property Inspector, in the Symbols pane, right-click the object and select `Inspect`.

Property Inspector: Edit properties of graphical and nongraphical objects directly in the Stateflow Editor

Property inspector view is available for Stateflow blocks. While working in the Stateflow editor, you can modify the properties of your state machine objects. To open the Property Inspector window, select **View** > **Property Inspector**, or right-click an object in the Symbols pane and select `Inspect`. See Set Data Properties and Specify Chart Properties.

State Transition Table Debugging: Design and debug tabular state machines faster by using animation, syntax highlighting, and breakpoints

When you design and debug a state transition table in R2016b, the state transition table editor offers these improvements:

- Syntax highlighting — Keywords are highlighted in blue, comments are highlighted in green.
- Animation — When you run a simulation, the active state or transition is highlighted.
- Breakpoints — You can set breakpoints directly in the state transition table.
- Debug tooltips — When you set breakpoints, you can hover your cursor over table cells to see what data is used and the data value.

Syntax Highlighting: Identify events and function names easily in charts with MATLAB as the action language

In charts that have MATLAB as the action language, function and events names are highlighted.

- Function names (blue)
- Event names (orange)

This syntax highlighting improves readability of the charts and matches the syntax highlighting of C charts.

Scoped Simulink Function Access: Call exported chart functions with restricted scope from Simulink function blocks

You can now encapsulate model components with scoped functions exported from Stateflow charts. Place the charts within a subsystem hierarchy, and call the functions by using qualified dot notation, *chartName.functionName* from a Simulink Caller block. Before R2016b, you placed your Stateflow chart with exported functions at the top level of the model, and set the functions as global throughout the Simulink model.

To call scoped Stateflow functions with qualified notation from a Simulink Caller block, select the chart property **Export Chart Level Functions**, and clear **Treat Exported Functions as Globally Visible**. To make all Stateflow functions available to the entire model, including other Stateflow charts, select **Treat Exported Functions as Globally Visible**. See Simulink Functions in Stateflow.

Compatibility Considerations

Stateflow chart properties for exporting functions have changed.

Before R2016b	In R2016b
Export Chart Level Functions (Make Global) allowed Stateflow blocks throughout the model to call the Stateflow functions.	**Export Chart Level Functions** allows Simulink Caller blocks to call Stateflow functions in the local hierarchy by using qualified dot notation, *chartName.functionName*.
Allow exported functions to be called by Simulink allowed Simulink Caller blocks to call Stateflow functions.	**Treat Exported Functions as Globally Visible** allows Stateflow and Simulink Caller blocks throughout the model to call the Stateflow functions. Do not use qualified notation to call these functions.

When you open a model created in a previous version, to maintain similar behavior, Stateflow selects **Export Chart Level Functions** and **Treat Exported Functions as Globally Visible** for charts with property **Export Chart Level Functions (Make Global)** selected.

Additional changes to behavior are listed in this table.

Before R2016b	In R2016b
Functions exported from a chart with **Export Chart Level Functions (Make Global)** selected, but not **Allow exported functions to be called by Simulink** were allowed to have the same name as a Simulink Function.	Functions exported from a chart with **Export Chart Level Functions** and **Treat Exported Functions as Globally Visible** are not allowed to have the same name as a Simulink Function in the model.
Supported scalar expansion of exported function input or output data.	Does not support scalar expansion of exported functions input or output data.
Support calling exported functions from charts with different sample rates.	Does not support calling exported functions from charts with different sample rates. Continuous time charts cannot call export functions.
Supports specifying the input or output data type using the `type` operator.	Does not support specifying the input or output data type using the `type` operator.

Diagnostic configuration parameters

Stateflow detects new diagnostics with these configuration parameters on the Diagnostics pane.

Configuration Parameter	Default Setting	Diagnostic
Absolute time temporal value shorter than the sampling period	`Warning`	Detects when a state or transition absolute time operator uses a time value that is shorter than the sample time for the Stateflow block
Self transition on leaf state	`Warning`	Detects when a self-transition on a leaf state can be removed. If there are no actions in the leaf state or on the self-transition, then the self-transition has no affect on chart execution.

Configuration Parameter	Default Setting	Diagnostic
Execute-at-Initialization disabled in presence of input events	`Warning`	Detects when triggered or enabled charts are not running at initialization.
Use of machine-parented data instead of Data Store Memory	`Warning`	Detects machine-parented data that should be replaced with chart-parented data of scope Data Store Memory.
Unreachable Execution Path	`Warning`	Detects when a path or object on a chart is unreachable and will not be executed.

Set the diagnostic action to `none`, `warning`, or `error` for each parameter.

Compatibility Considerations

The issues detected by the Diagnostics pane configuration parameter **Transition shadowing** are now detected by **Unreachable Execution Path**. **Transition shadowing** is no longer visible and is controlled by the setting for **Unreachable Execution Path**. When you load a model created in a version previous to R2016b, Stateflow changes the setting for **Unreachable Execution Path** to match the previous setting for the **Transition shadowing** parameter.

Diagnostic level option for message queue overflows

For each message queue, choose the level of diagnostic for queue overflows:

- `Error` (default)
- `Warning`
- `None`

Before R2016b, all message overflows caused an error.

Message Viewer updates to inspect values of structured data and sequencing of function calls

The Message Viewer block has been updated to display the values of structured data during model execution and sequencing of function calls for these function call types:

- Calls to Simulink Function blocks — Fully supported.
- Calls to Stateflow graphical or Stateflow MATLAB functions — With this support:

  - Scoped — Select the **Export chart level functions** chart option. Use the *chartName.functionName* dot notation.
  - Global — Select the **Treat exported functions as globally visible** chart option. Do not need the dot notation.

For more information, see Work with Message Viewer.

Bus support for Simulink Caller blocks calling Stateflow functions

Simulink Caller blocks can call exported Stateflow functions with buses as input and output data. Stateflow can also call Simulink functions with buses as input and output data.

Conditional breakpoints in MATLAB Functions for run-time debugging

To help you debug code, you can enter a MATLAB expression as a condition on a breakpoint inside a MATLAB function. Simulation then pauses on that breakpoint only when the condition is true. To set a conditional breakpoint, in the MATLAB function editor, right-click beside the line of code and select **Set Conditional Breakpoint**. Type the condition in the pop-up window. You can use any valid MATLAB expression as a condition. This condition expression can include numerical values and any data that is in scope at the breakpoint.

When you right-click a breakpoint, you can choose:

- `Set/Modify the condition`
- `Disable breakpoint`
- `Clear breakpoint`

You can also perform these actions from the **Breakpoints** menu in the MATLAB function editor.

Compiler optimization parameter support for faster simulation

In R2016b, Stateflow applies the setting for the configuration parameter **Compiler optimization level** to Stateflow blocks. To speed up the simulation time for your model, set the configuration parameter to **Optimizations on (faster runs)**. The application of compiler optimizations consumes extra time during the build process. The default setting, **Optimizations off (faster build)** disables compiler optimizations and provides the fastest build times.

Text Autocompletion for State Transition Tables

When typing in a state transition table, a list of suggested completions appears. Choose between the suggestions with the up and down arrows. To select a suggestion, use **Enter**.

R2016a

Version: 8.7

New Features

Bug Fixes

Compatibility Considerations

Smart Editing Cues: Accelerate common editing tasks with just-in-time contextual prompts

When you select a Stateflow block, a cue appears where you select common actions. When you move your cursor over the cue, an action bar appears. Click the action you want to perform. For Stateflow blocks, you can comment or uncomment the block or hide or display the block name.

Inside the Stateflow editor, use the cue and action bar to perform tasks on objects or groups of objects, such as:

- Toggle breakpoints
- Create superstates
- Comment and uncomment objects
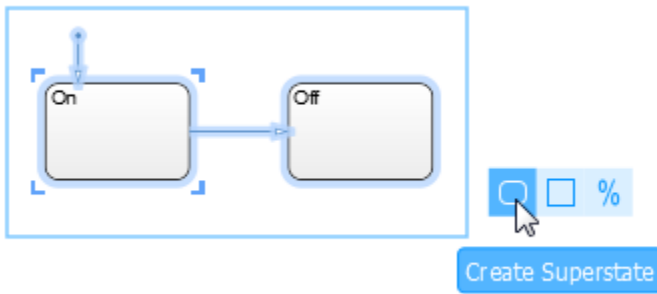
Intelligent Chart Completion: Build charts faster with automatic addition of default transitions and creation of complementary state names

The Stateflow editor provides you with these enhancements.

- *Automatic addition of default transition*: When you create a state in a new chart or new level of hierarchy, Stateflow automatically adds a default transition.
- *Creation of complementary state names*: When you copy a state with a common name, the copied state is given a complementary name.
- *Transition creation guide*: When you align states or junctions, blue transition guides appear. To create a transition, hover over one end of a guide to choose the direction and click to create.

Simulink Units: Specify, visualize, and check consistency of units on chart interfaces

Stateflow supports the specification of a unit property for data inputs and outputs of Stateflow blocks. Specify units by using the **Unit (e.g., m, m/s^2, N*m)** parameter for input or output data on charts, state transition tables, or truth tables. This parameter uses autocompletion to help you specify units.

Stateflow checks for inconsistencies in units between the data objects in Stateflow and their corresponding Simulink signals during model update. See Units in Stateflow.

Output Logging: Log output signals for charts

You can log output data of different types and sizes in Stateflow blocks. In previous releases, you could log only local data. See Configure States and Data for Logging.

JIT for Messages: Reduce model update time for messages with JIT compilation technology

Charts that contain messages qualify for Just-In-Time (JIT) compilation mode to improve model update performance. Stateflow applies JIT mode if possible.

API changes for commented objects

Previously, to determine if Stateflow objects were explicitly or implicitly commented out you used these properties:

- `Comment.Explicit` (Read/Write)
- `Comment.Implicit`(Read only)
- `IsCommented`(Read only)

In R2016a, the names of the properties have changed to:

- `IsExplicitlyCommented`(Read/Write)
- `IsImplicitlyCommented`(Read only)

Use the method `IsCommented()` to query if an object is explicitly or implicitly commented out. This method returns a Boolean.

Compatibility Considerations

Update scripts that use these API object properties.

Before R2016a	R2016a
`Comment.Explicit`	`IsExplicitlyCommented`
`Comment.Implicit`	`IsImplicitlyCommented`
`IsCommented` is a property	`isCommented` is a method

Stateflow model templates for common design patterns

From the Simulink Start page, by using a template, you can create a model with a Stateflow block. Use templates to start building your model from these common design patterns.

Template	Design Pattern
Blank Chart	Blank Stateflow chart
Simple Stateflow Chart	Chart containing basic building blocks for a state diagram
Hierarchical Chart	Chart that models a hierarchical state diagram

Template	Design Pattern
Simple State Transition Table	State transition table containing a tabular state machine
Moore Chart	Chart that uses Moore machine semantics

UserData parameter available for storing values

Use `set_param()` and `get_param()` to store and retrieve values in the `UserData` parameter for Stateflow blocks. Previously, Stateflow used the `UserData` parameter for storage of internal data.

R2015aSP1

Version: 8.5.1

Bug Fixes

R2015b

Version: 8.6

New Features

Bug Fixes

Compatibility Considerations

Multilingual Labels: Use any language to create comments and descriptions in states and transitions

You can use international characters to create comments and descriptions inside Stateflow charts.

Messages: Objects that carry data and can be queued

In Stateflow, you can send, receive, and forward messages that can hold data. You can also guard conditions or state actions with messages. You access message data by reading it or writing to it. Use messages for:

- Scheduling operations within charts
- Communicating between charts that run asynchronously
- Failure and diagnostic modeling

Messages interact in Simulink through message input and output ports.

Messages do not trigger a chart to wake up when a message is received. With events, if the receiver cannot immediately respond to the event, then the event is lost. Messages are queued at the message input port until the chart wakes up. When the chart wakes up, it responds to the messages from the input queue. Messages can be removed from the queue for processing during the chart execution period. At the end of chart execution, message processing is completed and the message is destroyed. For more information, see How Messages Work in Stateflow Charts.

You can create local messages. A local message has its own queue with the same queue properties as the message input port.

To visualize the progress of messages during simulation, use the Message Viewer block. For more information, see Work with Message Viewer.

Overflow and data range detection settings unified with Simulink

Previously, you controlled overflow detection in Stateflow blocks with the configuration parameter **Detect wrap on overflow**. This parameter was on the **Simulation Target** pane in the Model Configuration Parameters dialog box.

You now control the overflow detection for Stateflow blocks with the configuration parameter **Wrap on overflow**. This parameter is on the **Diagnostics: Data Validity** pane in the Model Configuration Parameters dialog box. Choose one of these three settings: `none`, `warning`, and `error`.

Previously, you controlled data range error checking in Stateflow blocks in the Stateflow Editor, with **Simulation > Debug > MATLAB & Stateflow Error Checking Options > Data Range**.

You now control data range checking with the configuration parameter **Simulation range checking**. This parameter is on the **Diagnostics: Data Validity** pane in the Model Configuration Parameters dialog box. Choose one of these settings: `none`, `warning`, and `error`.

See Diagnostics Pane: Data Validity.

Compatibility Considerations

When you open a Stateflow model saved in a previous release, a change in behavior is possible. These Stateflow options are no longer valid:

- **Detect wrap on overflow** on the **Simulation Target** pane in the Model Configuration Parameters dialog box
- **Simulation** > **Debug** > **MATLAB & Stateflow Error Checking Options** > **Data Range** in the Stateflow editor

In R2015b, Stateflow determines overflow detection and data range by the settings of these Simulink options on the **Diagnostics: Data Validity** pane in the Model Configuration Parameters dialog box.

- **Wrap on overflow**
- **Simulation range checking**

Set each configuration parameter to `none`, `warning`, or `error`. If the previous Stateflow options saved with the model were set differently than the current Simulink options, you see a warning.

The command-line parameter `SFSimOverflowDetection` is no longer valid. Use `IntegerOverflowMsg` instead. The API parameter `Debug.RunTimeCheck.DataRangeChecks` for a `Stateflow.Machine` is no longer valid. Use the command-line parameter `SignalRangeChecking` instead.

When you save a current Stateflow model in a previous version, the current Simulink parameters are saved. Both the Stateflow overflow and data range parameters are saved as selected.

New State Transition Table Editor: Dock state transition tables within the Stateflow editor window

The new state transition table editor is docked inside of the Stateflow editor window. When you open a state transition table, it no longer opens a window outside of the Stateflow editor.

Monitor State Activity in Code: Bind active state child variable to Simulink.Signal for controlling its properties in generated code

You can specify active state data as a local variable. Create the active state output port in the chart Properties window. In the Model Explorer, change the scope of the data from `output` to `local`. You can specify information for code generation by binding the local state activity data to a `Simulink.Signal` object. Modify the properties in `CoderInfo`.

Initial values supported for data in charts that use MATLAB as the action language

You can assign initial values for data with a scope of output or local in charts using MATLAB as the action language.

Continuous-time update method not allowed in Moore charts

Compatibility Considerations

In Moore charts, you cannot set the update method to `Continuous`. For modeling systems with continuous-time in Stateflow, use Classic or Mealy charts.

R2015a

Version: 8.5

New Features

Bug Fixes

Compatibility Considerations

JIT compilation technology to reduce model update time

Stateflow uses just-in-time (JIT) compilation technology to improve model update performance of many charts. For these charts, Stateflow does not generate C code or a MEX-file to simulate the chart. Stateflow applies JIT mode to charts that qualify. You do not have to enable it.

When a chart uses JIT mode, debugging is disabled. For charts in JIT mode, debugging is no longer tied to animation. Even when debugging is disabled, you see chart animation. To debug, set a breakpoint in the chart. Stateflow enables debugging, and does not use JIT mode.

Compatibility Considerations

By default, the software uses JIT mode on charts to speed up compilation time. When a chart uses JIT mode, debugging is disabled. During simulation, you cannot set a breakpoint. If you set a breakpoint in a chart before simulation begins, Stateflow enables debugging on the chart. You no longer directly enable or disable debugging with **Enable debugging/animation** on the **Simulation Target** pane of the Configuration Parameters dialog box or menu option.

In previous releases, if you set the command-line parameter `SFSimEnableDebug`, the software enabled debugging and animation. Now, setting this parameter prevents the chart from using JIT mode. To gain the performance of JIT mode, do not set this command line parameter.

Some charts do not qualify for JIT mode, such as charts that integrate custom C code or use signal logging. In these cases, the software defaults to MEX-file generation with debugging enabled. For optimal simulation performance for these charts, turn off debugging by using this command.

```
sfc('coder_options', 'forceDebugOff', 1);
```

After you run this command, these charts do not have debugging, animation, or run-time error checking.

In previous releases, you single stepped through a chart, and then stepped into an atomic subchart, or stepped out to another chart. Now, single stepping in a chart does not step into or out of other charts.

Mapping of atomic subchart variables with main chart variables of different scope

You can map variables in atomic subcharts to variables at the main chart level of different scopes. For example, you can now map an atomic subchart input to a main chart variable that is an input, output, local, or parameter. These mappings are supported.

Atomic Subchart Variable Scope	Main Chart Variable Scope
Input	Input, Output, Local, Parameter
Output	Output, Local
Data Store Memory	Data Store Memory, Local
Parameter	Parameter

You can also map a variable in an atomic subchart to an element of a bus in the main chart.

Do not export graphical functions from an atomic subchart that maps variables to variables at the main chart level with a different scope. For example, if an atomic subchart maps an input variable to

an output variable in the main chart, the scope of the variable is different between the atomic subchart and the main chart. For this atomic subchart, do not export graphical functions.

Also, you cannot log signals from atomic subcharts that map variables with different scopes.

Moore chart improvements for functions, local data, and code readability

Moore charts are improved to support additional programming constructs, and enforce Moore semantics. Beginning in R2015a, in Moore charts, you can:

- Include local data.
- Include graphical functions.
- Include MATLAB functions.
- Include truth tables.
- Program actions in all levels of state hierarchy, not just leaf states.
- Easily read the generated code.
- Design an algebraic loop

For more information, see Design Considerations for Moore Charts.

Compatibility Considerations

Moore charts now have tighter semantic restrictions to ensure that outputs rely only on current state, and do not rely on input or previous output values. To relax this restriction, in the **Diagnostics > Stateflow** pane of the Model Configuration Parameters dialog box, change the setting for **Read-before-write to output in Moore chart**. Change the setting from `error` to `warning` or `none`.

Beginning in R2015a, Moore charts have these limitations:

- You cannot export functions.
- You cannot enable super step semantics.
- You cannot use more than one input event.

The limitation to not allow data store memory (DSM) is enforced in Mealy and Moore charts.

Nonprefixed enumerations in charts using MATLAB as action language

You can now refer to enumerated values in charts that use MATLAB as the action language without fully specifying the class name. To do so, explicitly specify the enumeration type as the data type for any data used in the Stateflow chart. Then you can use a simplified nonprefixed identifier for that enumeration type.

For example, you have the following enumeration defined on the MATLAB path:

```
classdef(Enumeration) TrafficColors < Simulink.IntEnumType
  enumeration
    Red(0)
    Yellow(1)
    Green(2)
```

```
    end
end
```

If you have data in your chart where the type is `Enum: TrafficColors`, you can use the nonprefixed notation `Red`, instead of `TrafficColors.Red`.

For more information, see Notation for Enumerated Values in Charts and Define Enumerated Data in a Chart.

Removal of transition error checking

In R2015a, the error checking option, **Simulation > Debug > MATLAB & Stateflow Error Checking Options > Transition Conflict**, has been removed. In previous releases, this option was available for C charts that used implicit ordering of transitions. Use explicit ordering of transitions to ensure that there are no transition conflicts.

Removal of set breakpoints options in property dialog boxes

In R2015a, the options to set breakpoints in the property dialog boxes of these objects have been removed:

- Charts
- States
- Transitions
- Graphical functions
- Truth tables
- Atomic subcharts
- State transition tables

You can set breakpoints for these objects directly in the Stateflow editor. For more information, see Set Breakpoints to Debug Charts.

R2014b

Version: 8.4

New Features

Bug Fixes

Compatibility Considerations

Comment-out capability to disable objects in the state diagram

You can now comment out Stateflow objects to exclude the objects from simulation. Use commenting to:

- Debug a chart by making minor changes between simulation runs.
- Test and verify the effects of objects on simulation results.
- Create incremental changes for rapid iterative design.

For more information, see Commenting Stateflow Objects in a Chart.

Window to manage conditional breakpoints and watch chart data

The new Stateflow Breakpoints and Watch Data window provides easier management of all Stateflow breakpoints and watch data. In the **Breakpoints** tab, you can:

- Set conditions on breakpoints.
- Disable and clear individual breakpoints.
- View the hit count for breakpoints.

In the **Watch Data** tab, you can:

- Watch data at any point in a simulation.
- View data values in different library instances at the same time.
- View highlights on data that changed from the last time simulation paused.

This new window replaces the Stateflow Debugger. You now control Stateflow error checking options (Transition Conflict, Data Range, and Detect Cycles) from **Simulation > Debug > Stateflow Error Checking Options**. For more information, see Manage Stateflow Breakpoints and Watch Data.

Compatibility Considerations

Breakpoints and watch data are no longer stored with the model. The breakpoints and watch data are associated with the MATLAB session. You can save the current list to a file and reload it to another session to restore the list of breakpoints and watch data.

Simulink blocks that create and call functions across Simulink and Stateflow

You can call the new Simulink Function block from inside Stateflow charts. The new Simulink Caller blocks can also call exported Stateflow functions.

User-controlled enumeration size for active state output

You can now set the storage type and size for enumerations created with active state output. Choose the storage type in **Base storage type for automatically created enumerations:** under the **Optimization > Stateflow** pane of the Configuration Parameters dialog box. If you need a smaller memory footprint, use this option.

Faster chart simulation and animation

Lightning Fast animation provides the fastest simulation speed by buffering the highlights. During **Lightning Fast** animation, the more recently highlighted objects are in a bolder, lighter blue. These highlights fade away as simulation time progresses. For more information, see Animate Stateflow Charts.

Improved state transition matrix

The state transition matrix has traceability of cells to the state transition table. You can interact with the state transition matrix to search and filter states. You can easily view the conditions and events of the state machine. For more information, see View State Reactions with State Transition Matrix.

Active state output not allowed with Initialize Outputs Every Time Chart Wakes Up

Compatibility Considerations

Do not set the chart property **Initialize Outputs Every Time Chart Wakes Up** on charts that use active state output. The behavior is unpredictable.

19

R2014a

Version: 8.3

New Features

Bug Fixes

Compatibility Considerations

Intelligent tab completion in charts

Stateflow provides context-sensitive editing assistance with tab completion. You can quickly select syntax-appropriate options for keywords, data, event, and function names.

Single chart block in Stateflow library with MATLAB as the default action language

In R2014a, there is one Stateflow Chart block that defaults to using MATLAB as the action language. You can modify the action language of a chart to use C syntax. For more information, see Modify the Action Language for a Chart.

Bus signal logging in charts

You can now log bus signals in Stateflow charts.

Output of leaf-state activity to Simulink

In previous releases, you could output self and child activity to Simulink. In R2014a, you can also output leaf-state activity using automatically managed enumerations.

UTF-16 character support for Stateflow blocks

You can use international characters when naming these Stateflow blocks:

- Charts
- State Transition Tables
- Truth Tables

Syntax auto-correction inserts explicit cast for literals

The auto-correction for MATLAB syntax now inserts explicit casts for literals.

Improved algebraic loop handling in Simulink can affect Stateflow blocks

Compatibility Considerations

A model with a Bus Selector block between two Stateflow exported graphical functions that share the same memory can produce different results.

Typedef generation management for imported buses and enumerations

For MATLAB functions and charts, you can manage generation of `typedef` definitions for imported bus and enumeration types in the Configuration Parameters dialog box, on the **Simulation Target**

pane. You can choose to provide the `typedef` definition in a custom header file, or have Simulink generate the definitions.

Updated Search & Replace tool

The Stateflow **Search & Replace** tool has been updated to simplify the controls and improve the color selection of the interface.

Support of complex data types with data store memory

Stateflow now supports complex data types for use with Data Store Memory.

Streamlined MEX compiler setup and improved troubleshooting

You no longer have to use `mex -setup` to choose a compiler. `mex` automatically locates and uses a supported installed compiler. You can use `mex -setup` to change the default compiler. See Changing Default Compiler.

Moore chart outputs cannot depend on inputs

Compatibility Considerations

In previous versions of Stateflow, in Moore charts that used MATLAB as the action language, you could assign an output that was dependent on an input. This construct violates Moore semantics, and Stateflow now generates a compiler error. For more information, see Design Considerations for Moore Charts.

Transition conflict error checking only on C charts with implicit execution order

Transition Conflict in **Error checking options** of the Stateflow debugger is only valid for C charts that use implicit transition execution ordering. The debugger does not check transition ordering for charts that use MATLAB as the action language, or C charts with explicit transition ordering.

R2013b

Version: 8.2

New Features

Bug Fixes

LCC compiler included on Windows 64-bit platform for running simulations

The Windows® 64-bit platform now includes LCC-win64 as the default compiler for running simulations. You no longer have to install a separate compiler for simulation in Stateflow and Simulink. You can run simulations in Accelerator and Rapid Accelerator modes using this compiler.

Note The LCC-win64 compiler is not available as a general compiler for use with the command-line MEX in MATLAB. It is a C compiler only. You cannot use it for SIL/PIL modes.

LCC-win64 is used only when another compiler is not configured in MATLAB. To build MEX files, you must install a compiler. See https://www.mathworks.com/support/compilers/current_release/.

Tab completion for keywords and data in charts

Press the **Tab** key for automatic word completion of keywords, data, and function names in charts.

Pattern Wizard for inserting logic patterns into existing flow charts

You can now use the Pattern Wizard to add loop or decision logic to a previously created pattern in a flow chart.

Absolute time temporal logic keywords, msec and usec, for specifying short time intervals

You can now specify milliseconds and microseconds for absolute time temporal logic in charts.

Continuous time support in charts with MATLAB as the action language

Charts that use MATLAB as the action language now support continuous time mode with zero crossing detection.

Content preview for Stateflow charts

When a chart is closed, you can preview the content of Stateflow charts in Simulink. You can see an outline of the contents of a chart. During simulation you can see chart animation. When a chart is open, you can preview the content of subcharts, Simulink functions, and graphical functions. For details, see Content Preview for Stateflow Objects.

For example, the Temporal Logic chart uses content preview, and the Without Temporal Logic chart does not.

Temporal Logic

Without Temporal Logic
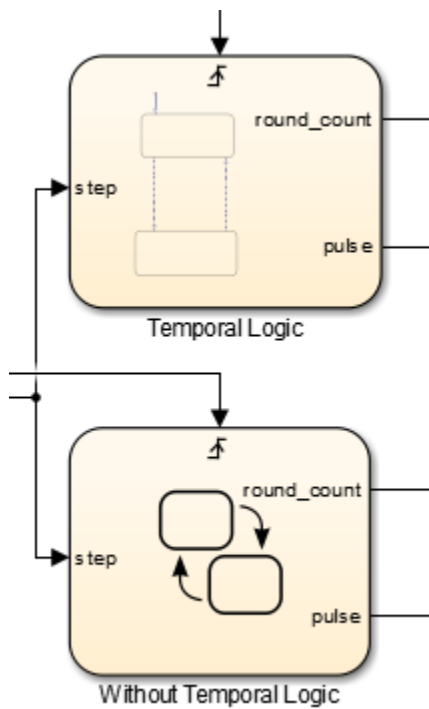
Code generation improvement for absolute-time temporal logic in charts with discrete sample times

For charts with discrete sample time that are not inside a triggered or enabled subsystem, absolute-time temporal operators generate improved code. The generated code now uses integer counters to track time instead of the Simulink time counter. This allows more efficient code, as well as enabling this code for use in software-in-the-loop(SIL) and processor-in-the-loop(PIL) simulation modes.

R2013a

Version: 8.1

New Features

Bug Fixes

Compatibility Considerations

Output of child-state activity to Simulink using automatically managed enumerations

In previous releases, you could output whether or not a state is active by selecting **Output state activity** in the State properties dialog box. In R2013a, you can now also output to Simulink the child activity of a chart, state, atomic subchart, or State Transition Table as an enumeration.

Masking of Stateflow block to customize appearance, parameters, and documentation

In R2013a, you can mask a chart, Truth Table block, or State Transition Table block directly. In previous releases, you had to place the Stateflow block in a subsystem, and then mask that subsystem.

Compatibility Considerations

In R2013a, MATLAB scripts or functions that rely on the `MaskType` property of Stateflow blocks need to be updated. For example, `get_param(handle_to_SF_block, 'MaskType')` now returns an empty value instead of `'Stateflow'`. For backward compatibility, using `find_system('MaskType','Stateflow')` returns all the Stateflow blocks. However, use the Stateflow API instead, as a better practice. See Overview of the Stateflow API. Do not create masks with Mask Type "Stateflow", because the behavior is unpredictable.

Option to parse Stateflow chart to detect syntax errors and unresolved symbols without updating diagram

You can detect unresolved symbols in a chart without updating the diagram or starting simulation. The Stateflow parser can access all required information for detecting unresolved symbols, such as enumerated data types and exported graphical functions from other charts.

Propagation of parameter names to generated code for improved code readability

In previous releases, Stateflow parameters in the generated code were not derived from the parameter names. In R2013a, parameter names appear unchanged in the code, which provides better traceability between the chart and the generated code.

Complex inputs and outputs for exported graphical functions

Exported graphical functions now support inputs and outputs of complex type.

Use of type(data_name) for specifying output data type disallowed for buses

If your chart specifies output data type using the expression `type(data_name)`, you get an error if `data_name` is of bus type.

Compatibility Considerations

A model created in a previous release might cause an error in R2013a if the chart contains an output with the data type specification `type(data_name)` if `data_name` is of bus type.

New and enhanced examples

The following demo has been added in R2013a:

Example...	Shows how you can...
`sf_trafffic_light`	Model a traffic light system with active state output.

The following demos have been enhanced in R2013a:

Example...	Shows how you can...
`sf_aircraft`	Model now uses active state output.

R2012b

Version: 8.0

New Features

Bug Fixes

Compatibility Considerations

New editor for Stateflow charts and Simulink models with tabbed windows and model browser tree

The new editor unifies Stateflow and Simulink functionality. The Stateflow Editor shares most of the same menu items with the Simulink Editor, and provides the following enhancements:

- **Unified canvas**, for editing Stateflow charts and Simulink models in the same window.
- **Tabbed windows**, for accessing Stateflow charts in the same context as Simulink models.
- **Model Browser tree**, for browsing the complete model hierarchy, including Stateflow charts.
- **Cross-platform consistency**, for accessing the same functionality on Windows, UNIX®, and Mac platforms.

Stateflow Editor menu bar changes

Menu bar changes in the new Stateflow Editor are the same as for the new Simulink Editor.

Stateflow Editor context menu changes

All changes for the following three Stateflow Editor context menus are the same as for the new Simulink Editor:

- From the canvas
- From a block
- From a signal

The following sections describe changes to context menus that are specific to Stateflow:

- "From a chart" on page 22-2
- "From a function" on page 22-3
- "From a transition" on page 22-3
- "From a state" on page 22-3

From a chart

R2012a Stateflow Editor Context Menu	New Stateflow Editor Equivalent
Patterns	**Add Patterns**.
Add Note	Not available from context menu. From the palette, select the Annotation icon (🅰).
Cut **Copy**	Not available from context menu. Use the context menu for an item in the chart that you want to cut or copy.
Back **Forward** **Go To Parent**	Not available from context menu. From the menu bar, use **View > Navigate**.

R2012a Stateflow Editor Context Menu	New Stateflow Editor Equivalent
Execution Order	Not available from context menu.
	From the menu bar, use **Chart > Parameters**.
Font Size	Not available from context menu.
Arrowhead Size	From the menu bar, use **Chart > Format**.
Format > Align Items	Not available from context menu.
Format > Distribute Items	From the menu bar, use **Chart > Arrange**.
Format > Resize Items	
Fit To View	Not available from context menu.
	From the menu bar, use **View > Zoom**.
Breakpoints	**Set Breakpoints on Chart Entry** and **Clear Breakpoints**.
Debug	Not available from context menu.
	From the menu bar, use **Simulation > Debug**.
Find	Not available from context menu.
	From the menu bar, use **Edit > Find**.
Edit Library	**Library Link**.

From a function

In the R2012a Stateflow Editor, right-clicking a Stateflow function displays the chart context menu. In the new Stateflow Editor, each kind of Stateflow function (for example, graphical function or Truth Table) has its own context menu.

From a transition

In the R2012a Stateflow Editor, right-clicking a Stateflow transition displays the chart context menu. In the new Stateflow Editor, a transition has its own context menu.

There is no longer a **Smart** menu item. In the R2012a Stateflow Editor, the **Smart** menu item enabled smart transitions. Smart transitions have ends that slide around the surfaces of states and junctions. When the source and/or destination objects are moved and resized in the chart, these transitions use sliding and other behaviors to enable you to produce an aesthetically pleasing chart. The new Stateflow Editor uses smart transitions all the time.

From a state

In the R2012a Stateflow Editor, right-clicking a Stateflow state displays the chart context menu. In the new Stateflow Editor, a state has its own context menu.

Stateflow keyboard and mouse shortcut changes

The new Simulink Editor and Stateflow Editor use the same navigation shortcuts.

Task	R2012a Stateflow Editor Shortcut	New Stateflow Editor Equivalent
Display the parent of the currently displayed chart or subchart.	**..** (two periods)	To navigate to the parent, use the up arrow on the toolbar or use the **Escape** key.
Zoom in by an incremental amount.	**+** or **r** or **R**	Use mouse scroll wheel or **Ctrl+ +** (the plus sign)
Zoom out by an incremental amount.	**-** or **v** or **V**	Use mouse scroll wheel or **Ctrl+-** (the minus sign).
Fit chart to screen.	**0** or **Space Bar**	Use **Space Bar** or from the menu bar, use **View > Zoom > Fit To View**.
Zoom to normal view.	**1**	**Alt+1**
Move the current view down within the full chart.	**2**	Use the Stateflow Editor scroll bars.
Move the current view down and right within the full chart.	**3**	Use the Stateflow Editor scroll bars.
Move the current view left within the full chart.	**4**	Use the Stateflow Editor scroll bars.
Fit the currently selected object to full view. If no object is selected, the chart is fit to full view.	**5**	Not yet supported
Move the current view right within the full chart.	**6**	Use the Stateflow Editor scroll bars.
Move the current view up and left within the full chart.	**7**	Use the Stateflow Editor scroll bars.
Move the current view up within the full chart.	**8**	Use the Stateflow Editor scroll bars.
Move the current view up and right within the full chart.	**9**	Use the Stateflow Editor scroll bars.

Editing assistance through smart guides, drag margins, transition indicator lines, and just-in-time error notifications

The new Stateflow Editor makes it easier to create and modify charts by providing these enhancements:

- **Smart guides**, for aligning objects interactively as you place them in the chart.
- **Drag margins**, which allows all objects within a container to move together. The mouse cursor changes to a double-arrow when you are within the drag margins of an object.
- **Transition indicator lines**, for identifying the label associated with a selected transition.
- **Just-in-time error notification**, for flagging illegal object placement during editing (for example, when two states overlap).

State transition tables that provide tabular interface to model state machines

A state transition table is an alternative way of expressing modal logic. Instead of drawing states and transitions graphically in a Stateflow chart, you express the modal logic in tabular format. Stateflow automatically generates a graphical state chart from the tabular format, so you can use animation and in-chart debugging.

Benefits of using state transition tables include:

- Ease of modeling train-like state machines, where the modal logic involves transitions from one state to its neighbor
- Concise, compact format for a state machine
- Reduced maintenance of graphical objects

  When you add or remove states from a chart, you have to rearrange states, transitions, and junctions. When you add or remove states from a state transition table, you do not have to rearrange any graphical objects.

The new block is available in `sflib`. You can add the block to a new model by entering `sfnew('-STT')` at the MATLAB command line.

For more information, see Tabular Expression of Modal Logic and Model Bang-Bang Controller with a State Transition Table.

MATLAB language for state and transition labels with chart syntax auto-correction

In R2012b, you can use MATLAB as the action language to program Stateflow charts. Benefits of using MATLAB as the action language include:

- MATLAB syntax support in state labels and transition labels

  You can use the same MATLAB code that you write in a script or enter at the command line.
- Automatic identification of unresolved symbols in the new Symbol Wizard

  When you update the diagram or start simulation, the Symbol Wizard provides a list of unresolved data in your chart and infers the scope.
- Automatic inference of size, type, and complexity for data in the chart, based on usage (unless explicitly defined in the Model Explorer)
- Support for control flow logic in state labels

  For example, you can write `if-else` statements directly inside state actions:

```
StateA
du:
if (x > 0)
  x = x + 1;
else
  x = x + 2;
end
```

  You do not need to create a separate graphical function to define the flow logic.

- Automatic correction of common syntax errors.

  For example, if you type x++ on a transition segment, the expression is automatically converted to the correct MATLAB syntax, {x=x+1}.

For more information, see MATLAB Syntax for States and Transitions and Model Event-Driven System Using MATLAB Expressions.

In-chart debugging with visual breakpoints and datatips

In R2012b, the Stateflow debugger includes the following enhancements:

- Display of data values when hovering over a state or transition

  When you hover over a state or transition, all data values in scope for that object appear in a popup list.
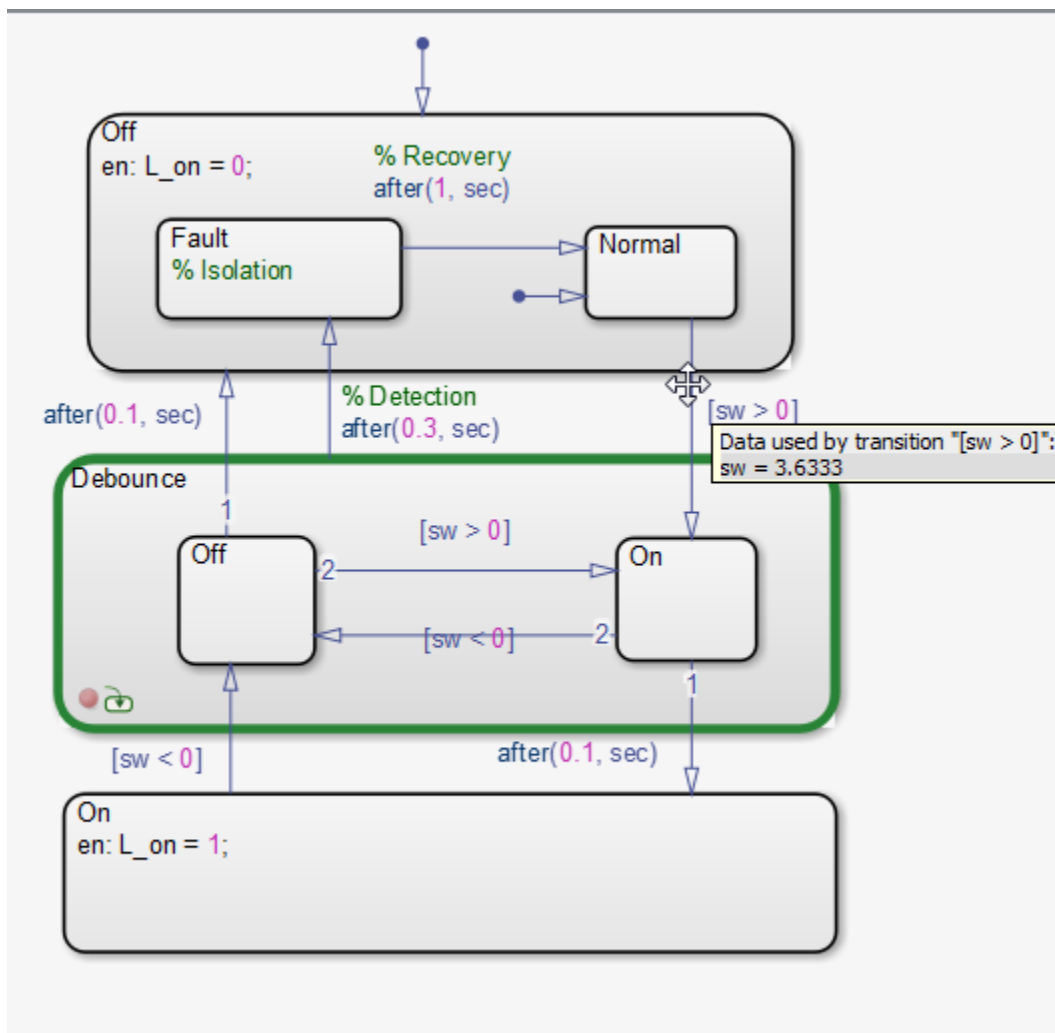
- **Step Over** and **Step Out** options on the debugger
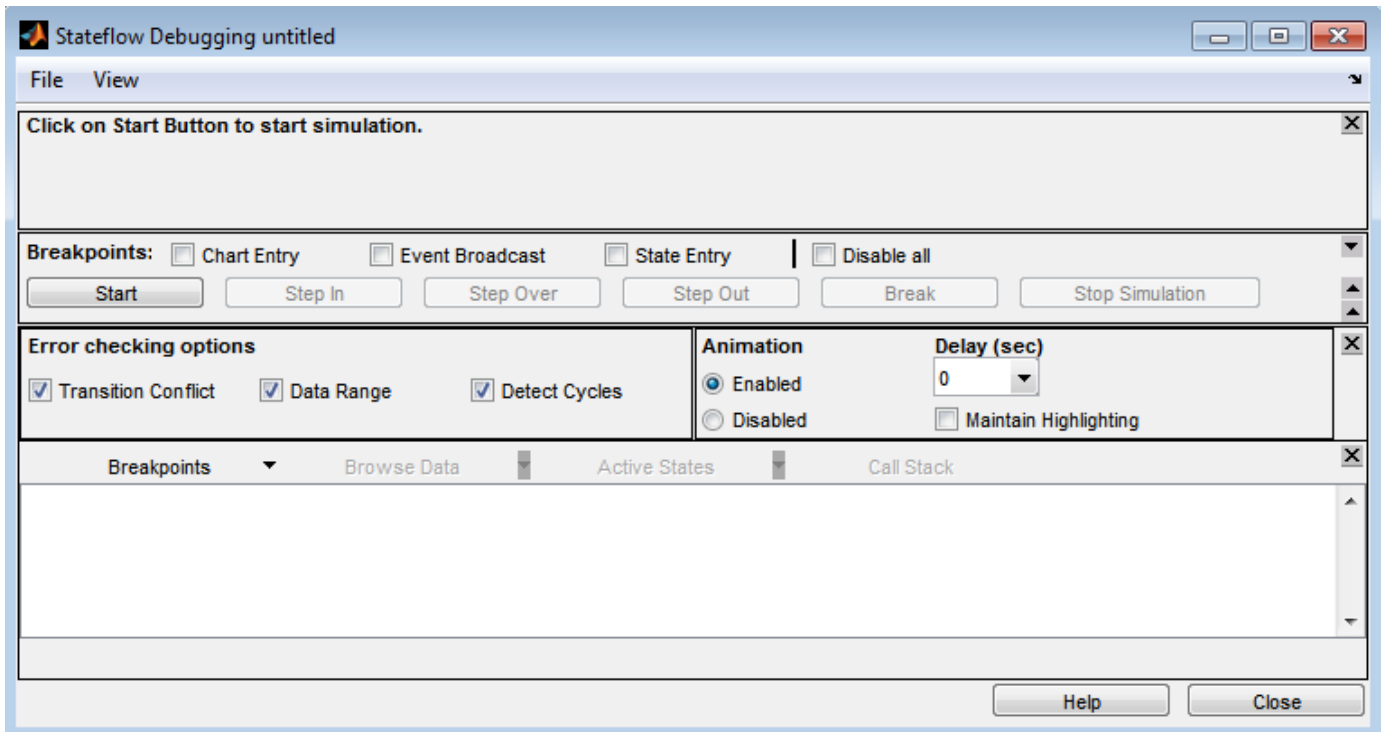
- When you click **Step Over**, you can skip the entire execution of a function call when the chart is in debug mode.
- When you click **Step Out**, you can skip the rest of the execution for a function call when the chart is in debug mode.
- Badges on graphical chart objects to indicate breakpoint settings
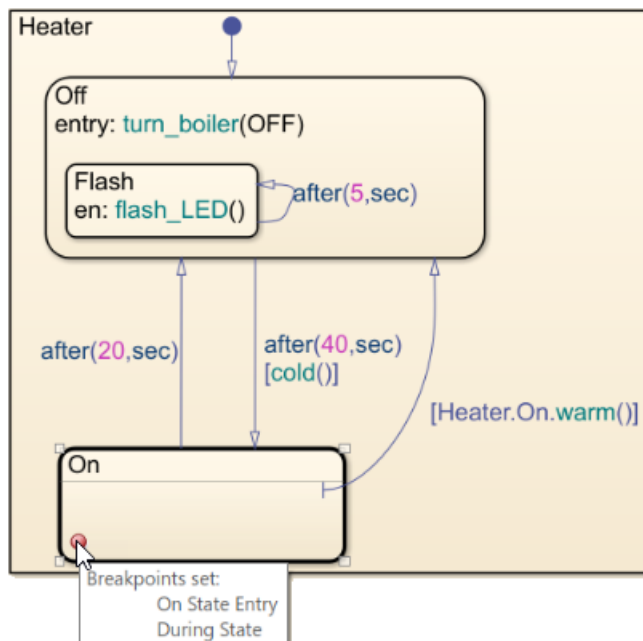
- When you hover over the badge on an object, you see a list of breakpoints in a popup list.
- To modify breakpoint settings, you can click the badge on an object instead of opening the properties dialog box.

See Set Breakpoints to Debug Charts.

Compatibility Considerations

In R2012b, you no longer need to launch the Stateflow debugger to stop chart execution at active breakpoints. Lifting this restriction means that during simulation, chart execution always stops at active breakpoints during simulation, even if the debugger is not running. To prevent unintended interruption of chart execution, Stateflow software automatically disables — but does not delete — existing breakpoints for all objects in charts created in earlier releases. Disabled breakpoints appear as gray badges; you can enable them as needed. See Relationship Between Breakpoints and the Debugger and Set Local Breakpoints.

In addition, in models created in earlier versions, Stateflow software removes `When Transition is Tested` breakpoints from transitions that do not have conditions. Starting in R2012b, you can set only `When Transition is Valid` breakpoints on transitions with no conditions.

Reuse of graphical functions with atomic boxes

In R2012b, you can use atomic boxes to reuse graphical functions across multiple charts and models. With atomic boxes, you can reuse models with graphical functions multiple times as referenced blocks in a top model. Because there are no *exported* graphical functions, you can use more than one instance of that referenced block in the top model.

For more information, see Reusing Functions with an Atomic Box.

Fewer restrictions for converting states to atomic subcharts

In R2012b, you can convert a state to an atomic subchart when the state accesses chart local data that has any of the following properties:

- `[M N]` size, where M and N are parameters that represent the data dimensions
- One of the following non-built-in data types:

  - Bus type
  - Alias type
  - Fixed-point type of nonzero fraction length, such as `fixdt(1,16,3)`

In previous releases, conversion of a state to an atomic subchart required that the chart local data have a static, deterministic size and a built-in data type.

Diagnostic for undirected local event broadcasts

In R2012b, you can detect undirected local event broadcasts using a new diagnostic in the **Diagnostics** > **Stateflow** pane of the Model Configuration Parameters dialog box. You can set the diagnostic level of **Undirected event broadcasts** to `none`, `warning`, or `error`.

Undirected local event broadcasts can cause unwanted recursive behavior in a chart and inefficient code generation. You can avoid this behavior by using the `send` operator to create *directed* local event broadcasts. For more information, see Guidelines for Avoiding Unwanted Recursion in a Chart and Broadcasting Events to Synchronize States.

Compatibility Considerations

For new models created in R2012b and existing models created in previous releases, the default diagnostic setting is `warning` to discourage the use of undirected local event broadcasts. Models that did not warn in previous releases might now issue a warning because the chart contains an undirected local event broadcast.

Diagnostic for transition action specified before condition action

In R2012b, you can detect specified transition actions before specified condition actions in transition paths, using a new diagnostic in the **Diagnostics > Stateflow** pane of the Model Configuration Parameters dialog box. You can set the diagnostic level of **Transition action specified before condition action** to `none`, `warning`, or `error`.

In a transition path with multiple transition segments, a specified transition action for a transition segment does not execute until the final destination for the entire transition path becomes valid. A specified condition action for a transition segment executes as soon as the condition becomes true. When a transition with a specified transition action precedes a transition with a specified condition action in the same transition path, the condition action for the succeeding transition might execute before the transition action for the preceding transition. When this diagnostic warns for transition paths containing transition actions specified before condition actions, you can identify out-of-order execution.

For more information, see Transitions.

Compatibility Considerations

In previous releases, the specification of transition actions before condition actions causes an error during simulation. To suppress this error for all models in future MATLAB sessions, use the following command:

```
sfpref('ignoreUnsafeTransitionActions',1);
```

In R2012b, the `ignoreUnsafeTransitionActions` preference does not exist and the default value of the **Transition action specified before condition action** diagnostic is `warning`. The warning occurs for all instances of transition actions specified before condition actions, even if you changed the `ignoreUnsafeTransitionActions` preference in a previous release.

Parentheses to identify function-call output events on chart and truth table block icons

In R2012b, function-call output events appear on Chart and Truth Table block icons with parentheses after the event name. This appearance is consistent with the rendering of input triggers on Function-Call Subsystem block icons.

Resolution of qualified state and data names

The algorithm for resolving a qualified state or data name performs a localized search for states and data that match the given path by looking in each level of the Stateflow hierarchy between the chart level and the parent of the state or data. The algorithm does not perform an exhaustive search of all states and data in the entire chart.

In previous releases, a warning would appear when the search resulted in no matches or multiple matches. In R2012b, this warning has changed to an error. For more information, see Checking State Activity and Using Dot Notation to Identify Data in a Chart.

Compatibility Considerations

Stateflow charts created in earlier releases now generate an error instead of a warning when the search for a qualified state or data name results in no matches or multiple matches.

Support for simulating charts in a folder that has the # symbol on 32-bit Windows platforms

In R2012b, on 32-bit Windows platforms, you can use the `lcc` compiler to simulate charts in a folder with the # symbol in its name.

Mac screen menubar enabled when Stateflow is installed

In previous releases, the Mac screen menubar was disabled when Stateflow was installed. This behavior was necessary to enable the Stateflow Editor menu options to work normally on a Mac.

In R2012b, the Mac screen menubar is enabled when Stateflow is installed.

Option to print charts to figure windows no longer available

In R2012b, printing the current view of a chart to a figure window is no longer available.

Compatibility Considerations

To print the current view of a chart, you can send the output directly to a printer or to a file. Available file formats include PS, EPS, JPG, PNG, and `TIFF`.

End of Broadcast breakpoint no longer available for input events

In R2012b, the option to set a breakpoint at **End of Broadcast** is no longer available for input events.

Compatibility Considerations

In previous releases, you could set both **Start of Broadcast** and **End of Broadcast** breakpoints for input events. Starting in R2012b, Stateflow ignores **End of Broadcast** breakpoints on input events for existing models.

Boxes can no longer be converted to states

You can no longer convert a Stateflow box object to a state object and vice versa.

Compatibility Considerations

In previous releases, you were able to convert a box object to a state object and vice versa. You must now delete the box or state and replace it with a new box or state with the same name.

R2012a

Version: 7.9

New Features

Bug Fixes

Compatibility Considerations

API Method for Highlighting Chart Objects

You can use the new `highlight` method to highlight one of the following objects in a chart:

- Box
- State
- Atomic subchart
- Transition
- Junction
- Graphical function
- MATLAB function
- Simulink function
- Truth table function

For more information, see highlight.

API Method for Finding Transitions That Terminate on States, Boxes, or Junctions

You can use the new `sinkedTransitions` method to find all inner and outer transitions whose destination is a state, box, or junction.

For more information, see sinkedTransitions.

API Property That Specifies the Destination Endpoint of a Transition

Transition objects now have a `DestinationEndPoint` property that describes the location of the transition endpoint at the destination object.

For more information, see Transition Properties.

Structures and Enumerated Data Types Supported for Inputs and Outputs of Exported Graphical Functions

In R2012a, inputs and outputs of exported graphical functions can use enumerated data types or structures. For more information, see Rules for Exporting Chart-Level Graphical Functions.

Mappings Tab in Atomic Subchart Properties Dialog Lists All Valid Scopes

In R2012a, the **Mappings** tab in the atomic subchart properties dialog lists all valid scopes for data and event mapping. All valid scopes appear, regardless of whether a data or event of that scope exists in the chart.

In previous releases, the **Mappings** tab listed only the scopes of data and events that existed in the chart and omitted any scope that did not exist. For more information, see Mapping Variables for Atomic Subcharts.

Full Decision Coverage When Suppressing Default Cases in the Generated Code

In R2011b, selecting **Suppress generation of default cases for Stateflow switch statements if unreachable** in the Configuration Parameters dialog box would result in decision coverage of less than 100%. In R2012a, you get full decision coverage when suppressing default cases in the generated code for your Stateflow chart.

For more information about decision coverage, see Model Coverage for Stateflow Charts in the Simulink Verification and Validation™ documentation.

Specification of Custom Header Files in the Configuration Parameters Dialog Box Required for Enumerated Types

If data in your chart uses an enumerated type with a custom header file, include the header information in the **Simulation Target** > **Custom Code** pane of the Configuration Parameters dialog box. In the **Header file** section, add the following statement:

```
#include "<custom_header_file_for_enum>.h"
```

For more information, see Rules for Using Enumerated Data in a Stateflow Chart in the Stateflow User's Guide.

Compatibility Considerations

In earlier releases, custom header files for enumerated types did not need to appear in the Configuration Parameters dialog box.

Removal of 'Use Strong Data Typing with Simulink I/O' in a Future Release

In a future release, the **Use Strong Data Typing with Simulink I/O** check box will be removed from the Chart properties dialog box because strong data typing will always be enabled.

When this check box is cleared, the chart accepts and outputs only signals of type `double`. This setting ensures that charts created prior to R11 can interface with Simulink input and output signals without type mismatch errors. For charts created in R11 and newer releases, disabling strong data typing is unnecessary. Also, many Stateflow features do not work when a chart disables strong data typing.

Compatibility Considerations

In R2012a, updating the diagram causes a parse warning to appear when a chart disables strong data typing. To prevent the parse warning, select the **Use Strong Data Typing with Simulink I/O** check box in the Chart properties dialog box.

R2011b

Version: 7.8

New Features

Bug Fixes

Compatibility Considerations

Chart Property to Control Saturation for Integer Overflow

A new chart property, **Saturate on integer overflow**, enables you to control the behavior of data with signed integer types when overflow occurs. The check box appears in the Chart properties dialog box.

Check Box	When to Use This Setting	Overflow Handling	Example of a Result
Selected	Overflow is possible for data in your chart and you want explicit saturation protection in the generated code.	Overflows saturate to either the minimum or maximum value that the data type can represent.	An overflow associated with a signed 8-bit integer saturates to –128 or +127.
Cleared	You want to optimize efficiency of the generated code.	The behavior depends on the C compiler you use for generating code.	The number 130 does not fit in a signed 8-bit integer and wraps to –126.

Arithmetic operations for which you can enable saturation protection are:

- Unary minus: –a
- Binary operations: a + b, a – b, a * b, a / b, a ^ b
- Assignment operations: a += b, a -= b, a *= b, a /= b

For new charts, this check box is selected by default. When you open charts saved in previous releases, the check box is cleared to maintain backward compatibility.

For more information, see Handling Integer Overflow for Chart Data in the Stateflow User's Guide.

Enhanced User Interface for Logging Data and States

A new **Logging** tab on the Data and State properties dialog boxes enables you to log signals in the same way that you do in Simulink. For more information, see Logging Data Values and State Activity in the Stateflow User's Guide.

Control of Default Case Generation for Switch-Case Statements in Generated Code

You can specify whether or not to always generate default cases for switch-case statements. This optimization works on a per-model basis and applies to the code generated for a state that has multiple substates. Use the following check box on the **Code Generation > Code Style** pane of the Configuration Parameters dialog box:

Check Box	When to Use This Setting	Format of Switch-Case Statements
Selected	Provide better code coverage by ensuring that every branch in the generated code is falsifiable.	Exclude the default case when it is unreachable.
Cleared	Ensure MISRA C® compliance and provide a fallback in case of RAM corruption.	Always include a default case.

This readability optimization is available for embedded real-time (ERT) targets and requires a license for Embedded Coder software. For new models, this check box is cleared by default. When you open models saved in previous releases, the check box is also cleared to maintain backward compatibility.

For more information, see Code Generation Pane: Code Style.

Detection of State Inconsistency Errors at Compile Time Instead of Run Time

In R2011b, you can detect inconsistency errors earlier in the model development process. If you select **Edit > Update Diagram** in the Simulink Editor, you get an error when Stateflow statically detects that there are no active children during execution of a chart or a state.

Compatibility Considerations

In previous releases, static detection of these inconsistency errors did not occur until run time.

Ability to Model Persistent Output Data for Mealy and Moore Charts

In previous releases, Mealy and Moore charts automatically applied the initial value of outputs every time the chart woke up. Both chart types ensured that outputs did not depend on previous values of outputs by enforcing the chart property **Initialize Outputs Every Time Chart Wakes Up**.

In R2011b, this restriction has been lifted. You can now choose whether or not to initialize outputs every time a Mealy or Moore chart wakes up. If you disable this chart property, you enable latching of outputs (carrying over output values computed in previous time steps). This enhancement enables you to model persistent output data for Mealy and Moore charts.

For more information, see Building Mealy and Moore Charts in the Stateflow User's Guide.

Control of Diagnostic for Multiple Unconditional Transitions from One Source

You can control the behavior of the Stateflow diagnostic that detects multiple unconditional transitions from the same state or the same junction. Set **Transition shadowing** to `none`, `warning`, or `error` on the **Diagnostics** > **Stateflow** pane of the Configuration Parameters dialog box.

For more information, see Diagnostics Pane: Stateflow in the Simulink Graphical User Interface documentation.

MEX Compilation with Microsoft Windows SDK 7.1 Now Supported

You can use Microsoft® Windows Software Development Kit (SDK) 7.1 as a MEX compiler for simulation on 32- and 64-bit Windows machines. For a list of supported compilers, see Choosing a Compiler in the Stateflow User's Guide.

Simulation Supported When the Current Folder Is a UNC Path

In R2011b, you can simulate models with Stateflow blocks when the current folder is a UNC path. In previous releases, simulation of those models required that the current folder not be a UNC path.

Removal of the Coverage Tab from the Stateflow Debugger

In R2011b, the **Coverage** tab of the Stateflow debugger has been removed. In previous releases, clicking the **Coverage** tab would show the following message:

```
Coverage feature obsoleted. Please use Simulink Verification and Validation
in order to get complete coverage of Simulink/Stateflow objects.
```

Test Point Selection Moved to the Logging Tab in Properties Dialog Boxes

The **Test point** check box has moved from the **General** tab to the **Logging** tab on State and Data properties dialog boxes.

R2011a

Version: 7.7

New Features

Bug Fixes

Compatibility Considerations

Migration of Stateflow Coder Features to New Product

In R2011a, all functionality previously available for the Stateflow Coder product is now part of the new Simulink Coder product.

Embedded MATLAB Functions Renamed as MATLAB Functions in Stateflow Charts

In R2011a, Embedded MATLAB functions have been renamed as MATLAB functions in Stateflow charts. This name change has the following effects:

- The function box now shows `MATLAB Function` instead of `eM` in the upper-left corner.
- Traceability comments in the generated code for embedded real-time targets now use `MATLAB Function` instead of `Embedded MATLAB Function`.
- For truth table functions in your chart, the **Settings** > **Language** menu now provides `Stateflow Classic` and `MATLAB` as the choices.

Scripts that use the `Stateflow.EMFunction` constructor method continue to work. All properties and methods for this object remain the same.

Use of MATLAB Expressions to Specify Data Size

In R2011a, you can enter MATLAB expressions in the **Size** field of the Data properties dialog box:

This enhancement enables you to use additional constructs, such as:

- Variables in the MATLAB base workspace
- Enumerated values on the MATLAB search path
- Expressions that use `fi` objects

For more information, see Sizing Stateflow Data in the Stateflow User's Guide.

Compatibility Considerations

For the **Size** field, name conflict resolution works differently from previous releases. In R2011a, when multiple variables with identical names exist, the variable with the highest priority is used:

1 Mask parameters

2 Model workspace

3 MATLAB base workspace

4 Stateflow data

In previous releases, Stateflow data took precedence over all other variables with identical names.

Ability to Change Data Values While Debugging

Previously, you could not change the values of Stateflow data while debugging a chart. Now you can change data values while the chart is in debug mode and see how simulation results change. For more information, see Changing Data Values During Simulation in the Stateflow User's Guide.

Ability to Debug a Single Chart When Multiple Charts Exist in a Model

When **Enable debugging/animation** is enabled on the **Simulation Target** pane of the Configuration Parameters dialog box, this setting applies to all charts in your model. In R2011a, you can enable or disable debugging on a chart-by-chart basis, using the **Debug** menu in the Stateflow Editor:
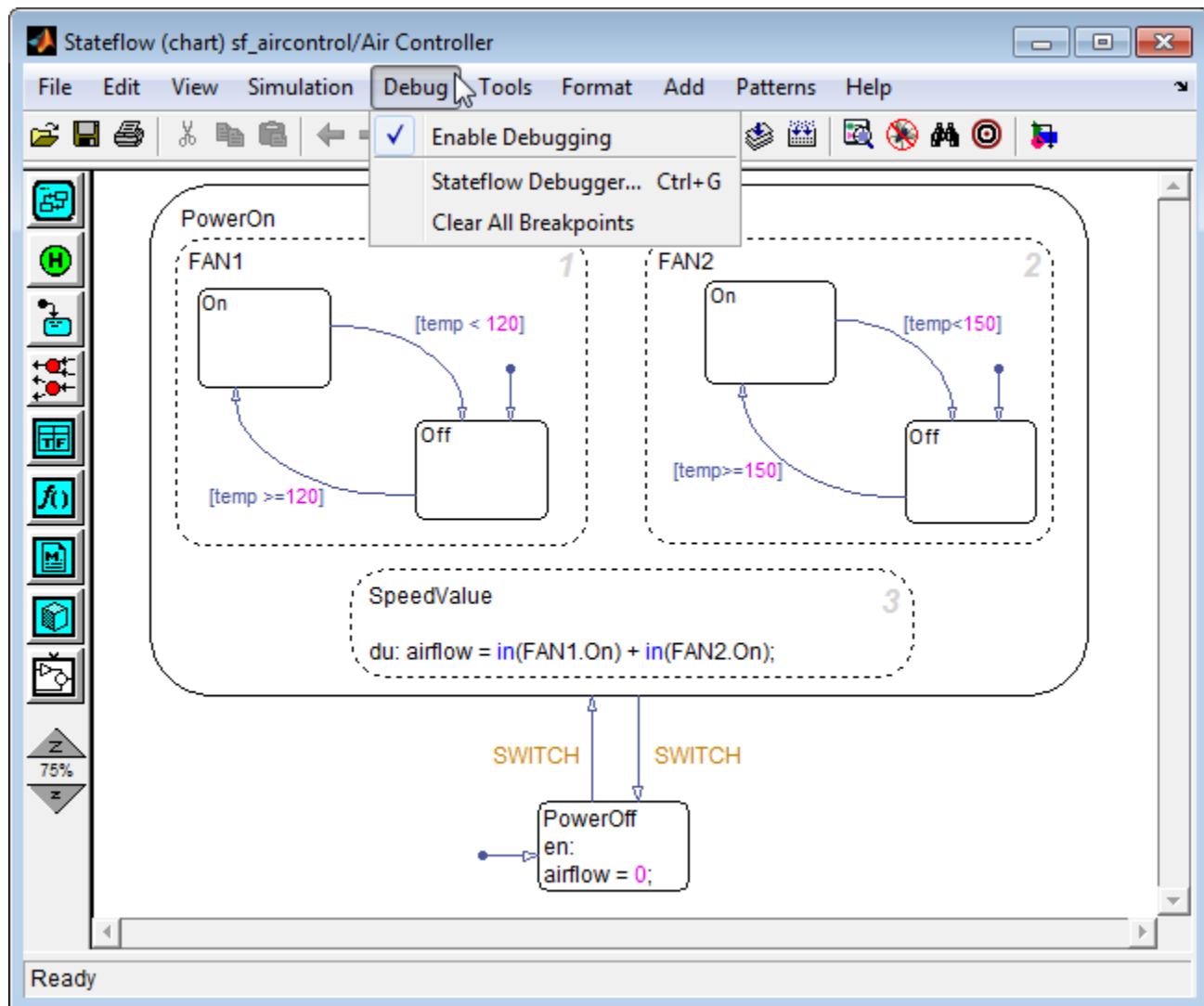
This enhancement enables you to focus on debugging a single chart, instead of having to debug all charts in the model. For details, see How to Enable Debugging for Charts in the Stateflow User's Guide.

You can also clear all breakpoints for a specific chart by selecting **Debug > Clear All Breakpoints** in the Stateflow Editor. For more information, see Clearing All Breakpoints in the Stateflow User's Guide.

In previous releases, you could open the debugger by selecting **Tools > Debug** in the Stateflow Editor. In R2011a, this menu option has moved to **Debug > Stateflow Debugger**.

Support for Input Events in Atomic Subcharts

In R2011a, you can use input events in atomic subcharts. For more information, see Making States Reusable with Atomic Subcharts in the Stateflow User's Guide.

Control of Generated Function Names for Atomic Subcharts

In R2011a, the generated function names for atomic subcharts follow the identifier naming rules for subsystem methods on the **Code Generation > Symbols** pane of the Configuration Parameters dialog box:
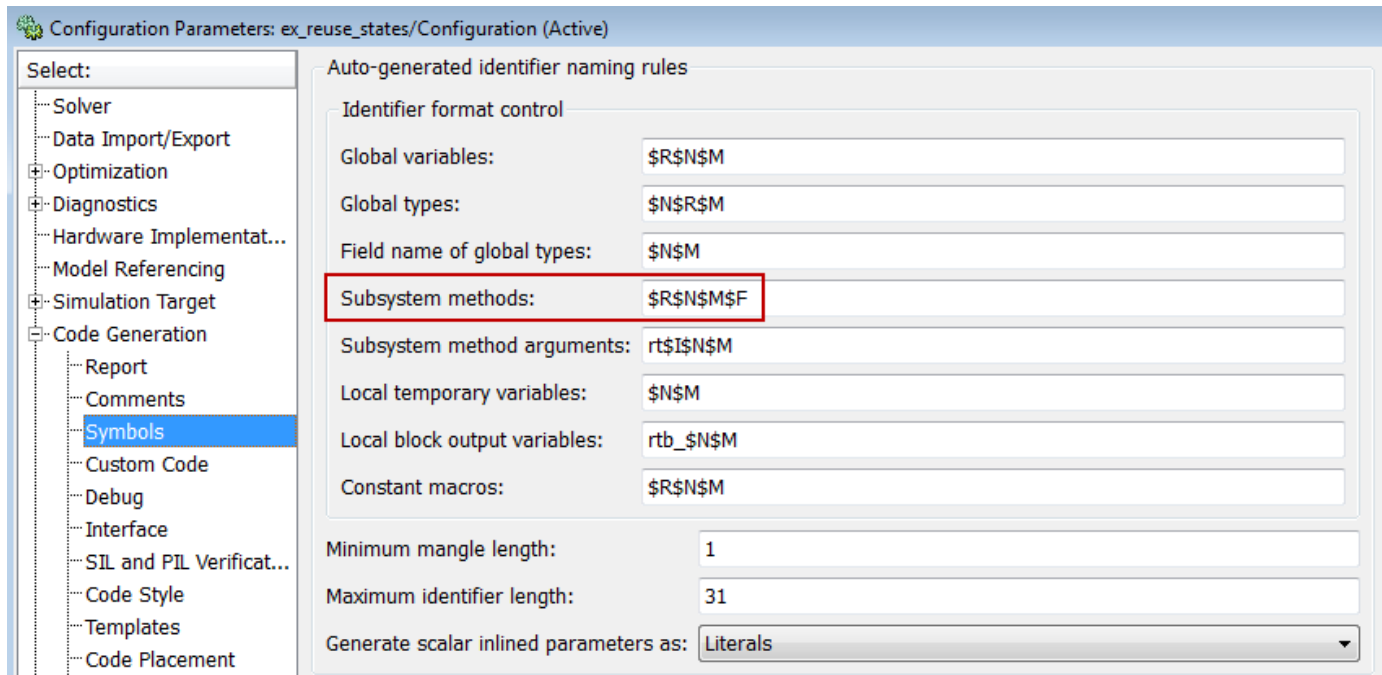
This enhancement enables you to control the format of generated function names for atomic subcharts when building an embedded real-time (ERT) target. For more information, see Generating Reusable Code for Unit Testing in the Stateflow User's Guide.

Enhanced Data Sorting in the Stateflow Debugger

In previous releases, the Stateflow debugger sorted data by scope first, before alphabetically listing data. In R2011a, the debugger sorts data alphabetically in the **Browse Data** section, without regard to scope. This enhancement helps you find specific data quickly when your chart contains many variables, for example, over a hundred.

Data sorting depends solely on the variable name and not on hierarchy. For example, if you have chart-parented data named `arrayOut` and state-parented data named `arrayData`, the list that appears in the **Browse Data** section is:

```
S.arrayData
arrayOut
```

The state name has no effect on data sorting.

For more information, see Watching Data Values During Simulation in the Stateflow User's Guide.

Option to Maintain Highlighting of Active States After Simulation

In R2011a, you can highlight the states that are active at the end of a simulation by selecting **Maintain Highlighting** in the Stateflow debugger.
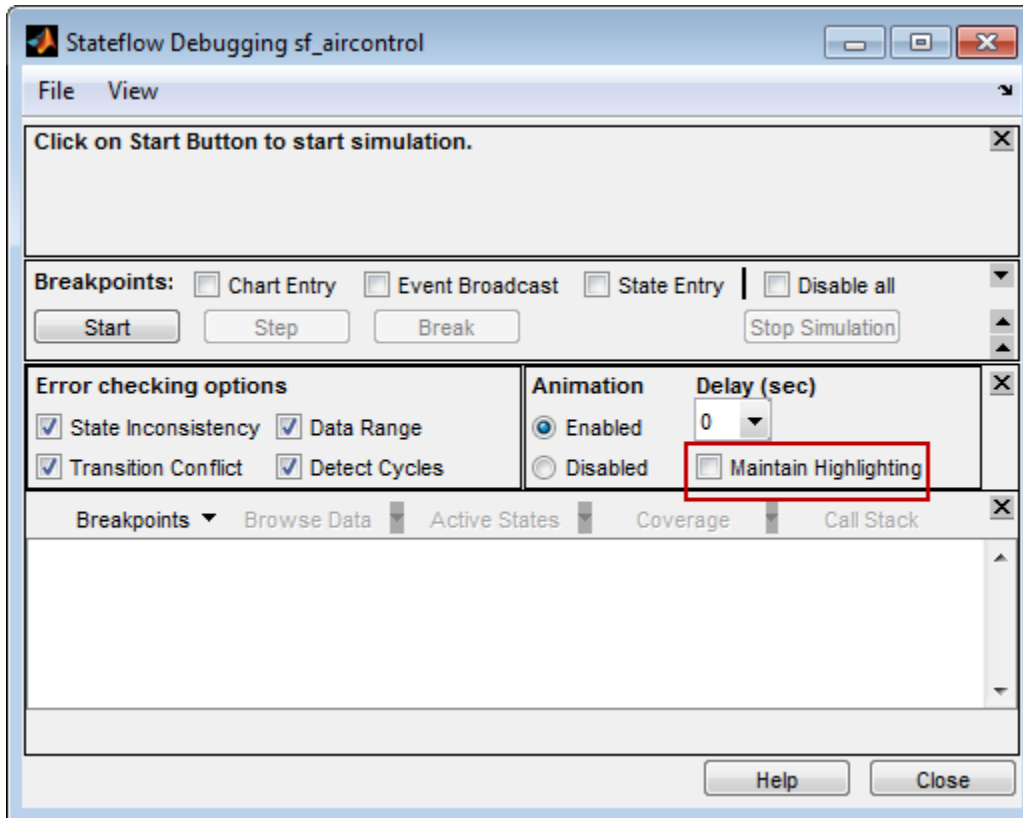
This enhancement enables you to inspect the active states of a chart after simulation ends, without having to use the SimState method `highlightActiveStates`. For more information, see Animating Stateflow Charts in the Stateflow User's Guide.

Right-Click Options for Setting Local Breakpoints

For graphical chart objects, you can now right-click the object to set local breakpoints. This enhancement enables you to set breakpoints more quickly, without having to open the properties dialog box for:

- Charts
- States
- Transitions
- Graphical functions
- Truth table functions

For more information, see Setting Local Breakpoints in the Stateflow User's Guide.

New Signal Logging Format That Simplifies Access to States and Local Data

You can now select a format for signal logging data. Use the **Signal logging format** parameter on the **Data Import/Export** pane of the Configuration Parameters dialog box to specify the format:

- `ModelDataLogs` — `Simulink.ModelDataLogs` format (the default; before R2011a, it was the only supported format)
- `Dataset` — `Simulink.SimulationData.Dataset` format (new in R2011a)

The `Dataset` format:

- Supports logging multiple data values for a given time step, which enhances signal logging of Stateflow data
- Uses MATLAB `timeseries` objects to store logged data (rather than `Simulink.Timeseries` and `Simulink.TsArray` objects), which enables you to work with logged data in MATLAB without a Simulink license
- Avoids the limitations of the `ModelDataLogs` format, which Bug Report 495436 describes

For more information, see Logging Data Values and State Activity.

Compatibility Considerations

In previous releases, selecting **Enable debugging/animation** on the **Simulation Target** pane of the Configuration Parameters dialog box would implicitly set all data and states in a Stateflow chart to be test points. In R2011a, you must select the **Test point** check box explicitly for data and states to appear in the Signal Selector dialog box of a Scope or Floating Scope block.

If you load models from previous releases that rely on the implicit behavior, mark the appropriate data or states as test points to ensure that they appear in the Signal Selector dialog box. For more information, see Monitoring Test Points in Stateflow Charts in the Stateflow User's Guide.

Support for Buses in Data Store Memory

You can now use buses, but not arrays of buses, as shared data in Stateflow data store memory.

Enhanced Readability of State Functions

In R2011a, state functions are more readable due to improved inlining heuristics.

Support for Arrays of Buses as Inputs and Outputs of Charts and Functions

In R2011a, you can pass arrays of buses as inputs and outputs of the following Stateflow objects:

- Charts
- MATLAB functions
- Simulink functions

Default Setting of 'States When Enabling' Chart Property Now Held

For new charts, the default setting of the **States When Enabling** chart property is `Held`. In previous releases, the default setting was `Inherit`. For more information, see Controlling States When Function-Call Inputs Reenable Charts in the Stateflow User's Guide.

Initial Value Vectors with Fixed-Point or Enumerated Values Now Evaluate Correctly

In previous releases, if you set an initial value vector using fixed-point or enumerated values, all elements of that vector would have the same value as the first element. For example:

For this initial value vector...	The chart used these values...
`[fi(1,1,16,0) fi(2,1,16,0)]`	`[1 1]`, instead of `[1 2]`
`[Colors.Red Colors.Yellow Colors.Green]`	`[Colors.Red Colors.Red Colors.Red]`, instead of `[Colors.Red Colors.Yellow Colors.Green]`

In R2011a, this bug has been fixed.

Compatibility Considerations

If you have any models that rely on the behavior of initial value vectors from previous releases, these models will behave differently in R2011a.

Mac Screen Menubar Disabled When Stateflow Is Installed

In R2011a, the Mac screen menubar is disabled when Stateflow is installed. This behavior enables Stateflow Editor menu options to work normally on a Mac.

To enable the Mac screen menubar, modify the `java.opts` file by adding the following line:

```
-Dapple.laf.useScreenMenuBar=true
```

To prevent a slowdown in the MATLAB Editor, check that the `java.opts` file contains the following line:

```
-Dapple.awt.graphics.UseQuartz=true
```

A `java.opts` file can reside in the folder from which you launch MATLAB or in the `bin/maci64` subfolder within the MATLAB root folder. A `java.opts` file in the latter location applies to all users, but individual users might not have permissions to modify a `java.opts` file there. If there is a `java.opts` file in both locations with settings that conflict, the setting in the `java.opts` file in the folder from which you launch MATLAB takes precedence. You might want to check both locations to see whether you have existing `java.opts` files and then decide which one to modify.

- To create a new `java.opts` file or modify an existing copy in the folder from which you launch MATLAB:

   1 Quit MATLAB.

   2 Relaunch MATLAB and immediately enter the following line in the Command Window:

```
edit java.opts
```

- To create or modify a `java.opts` file that applies to all users, you can enter the following line in the MATLAB Command Window at any time:

```
edit(fullfile(matlabroot,'bin','maci64','java.opts'))
```

R2010bSP2

Version: 7.6.2

Bug Fixes

R2010bSP1

Version: 7.6.1

Bug Fixes

R2010b

Version: 7.6

New Features

Bug Fixes

Compatibility Considerations

New Atomic Subcharts to Create Reusable States for Large-Scale Modeling

In R2010b, you can use atomic subcharts to:

- Break up a chart into standalone parts to facilitate team development
- Reuse states across multiple charts and models
- Animate and debug multiple charts side-by-side during simulation
- Use simulation to test changes, one-by-one, without recompiling the entire chart
- Generate reusable code for specific states or subcharts to enhance unit testing

For more information, see Making States Reusable with Atomic Subcharts in the Stateflow User's Guide.

Stateflow Library Charts Now Support Instances with Different Data Sizes, Types, and Complexities

In R2010b, you can use library link charts that specify different data sizes, types, and complexities. Previously, all library charts had to use the same settings for data size, type, and complexity. For more information, see Creating Specialized Chart Libraries for Large-Scale Modeling in the Stateflow User's Guide.

Support for Controlling Stateflow Diagnostics in the Configuration Parameters Dialog Box

In R2010b, you can control the behavior of the following Stateflow diagnostics in the **Diagnostics > Stateflow** pane of the Configuration Parameters dialog box:

- Unused data and events
- Unexpected backtracking
- Invalid input data access in chart initialization
- No unconditional default transitions
- Transition outside natural parent

For more information, see Diagnostics Pane: Stateflow in the Simulink Graphical User Interface.

Enhanced Custom-Code Parsing to Improve Reporting of Unresolved Symbols

In R2010b, you can resolve symbols in your chart to symbols defined in custom code while parsing the chart. This enhancement enables more accurate and earlier reporting of unresolved symbols. Previously, the parser assumed that any unresolved chart symbols were defined in custom code. You could not resolve chart symbols to symbols in your custom code until make time. If the chart symbols were undefined in the custom code, a make error would appear.

Also, the Symbol Autocreation Wizard was previously available only for 32-bit Windows platforms that use `lcc` for the mex compiler. In R2010b, the Symbol Autocreation Wizard is available to help you fix unresolved symbols, regardless of the compiler or platform.

To enable or disable custom-code parsing, you can use the **Parse custom code symbols** check box on the **Simulation Target > Custom Code** pane of the Configuration Parameters dialog box.

For more information, see:

- Parse custom code symbols in the Simulink Graphical User Interface
- Resolving Undefined Symbols in Your Chart in the Stateflow User's Guide

Temporal Logic Conditions Can Now Guard Transitions Originating from Junctions

Previously, you could not use temporal logic conditions on transitions that originated from junctions. Now you can use temporal logic conditions on transitions from junctions as long as the full transition path connects two states. For more information, see Rules for Using Temporal Logic Operators and Example of Detecting Elapsed Time in the Stateflow User's Guide.

Data Dialog Box Enhancements

In R2010b, the following changes to the Data properties dialog box apply:

Parameters	Location in R2010a	Location in R2010b	Benefit of Location Change
• **Initial value** • **Minimum** • **Maximum**	**Value Attributes** tab	**General** tab	Consistent with blocks in the Simulink library that specify these parameters on the same tab as the data type.
• **Test point** • **Watch in debugger**	**Value Attributes** tab	**General** tab	Increases visibility of commonly used parameters.
• **Save final value to base workspace** • **First index** • **Units**	**Value Attributes** tab	**Description** tab	Consolidates parameters related to the data description.

Branching of Function-Call Output Events No Longer Requires Binding of Event to a State

Previously, using a Function-Call Split block to branch a function-call output event from a chart to separate subsystems required binding of the event to a state. In R2010b, binding is no longer required.

Passing Real Values to Function Inputs of Complex Type Disallowed

In R2010b, you cannot pass real values to function inputs of complex type. This restriction applies to the following types of chart functions:

- Graphical functions

- Truth table functions
- Embedded MATLAB® functions
- Simulink functions

Compatibility Considerations

If you have existing models that pass real values to function inputs of complex type, an error now appears when you try to simulate your model.

Using Chart Block That Accesses Global Data in For Each Subsystem Disallowed

In R2010b, the following model configuration produces an error during Real-Time Workshop® code generation:

- A Chart block resides in a For Each Subsystem.
- The Chart block tries to access global data from Data Store Memory blocks.

New and Enhanced Demos

The following demos have been added in R2010b:

Demo...	Shows how you can...
sf_atomic_sensor_pair	Model a redundant sensor pair using atomic subcharts
sf_electrohydraulic	Model a servo mechanism for use in electrohydraulic systems

The following demo has been enhanced in R2010b:

Demo...	Now...
sf_elevator	Models an elevator system with two identical lifts using atomic subcharts

R2010a

Version: 7.5

New Features

Bug Fixes

Compatibility Considerations

Support for Combining Actions in State Labels

You can now combine `entry`, `during`, and `exit` actions in a single line on state labels. This concise syntax provides enhanced readability for your chart and helps eliminate redundant code. For more information, see Combining State Actions to Eliminate Redundant Code in the Stateflow User's Guide.

New Diagnostic Detects Unused Data and Events

A new diagnostic now detects unused Stateflow data and events during simulation. A warning message appears, alerting you to data and events that you can remove. This enhancement helps you reduce the size of your model by removing objects that have no effect on simulation.

This diagnostic checks for usage of Stateflow data, except for the following types:

- Machine-parented data
- Inputs and outputs of Embedded MATLAB functions

This diagnostic checks for usage of Stateflow events, except for the following type:

- Input events

For more information, see Diagnostic for Detecting Unused Data and Diagnostic for Detecting Unused Events in the Stateflow User's Guide.

Enhanced Support for Variable-Size Chart Inputs and Outputs

You can explicitly pass variable-size chart inputs and outputs as inputs and outputs of the following functions:

- Embedded MATLAB functions
- Simulink functions
- Truth table functions that use Embedded MATLAB action language

For more information, see Using Variable-Size Data in Stateflow Charts in the Stateflow User's Guide.

Support for Chart-Level Data with Fixed-Point Word Lengths Up to 128 Bits

Chart-level data now support up to 128 bits of fixed-point precision for the following scopes:

- Input
- Output
- Parameter
- Data Store Memory

This increase in maximum precision from 32 to 128 bits provides these enhancements:

- Supports generating efficient code for targets with non-standard word sizes
- Allows charts to work with large fixed-point signals

You can explicitly pass chart-level data with these fixed-point word lengths as inputs and outputs of the following functions:

- Embedded MATLAB functions
- Simulink functions
- Truth table functions that use Embedded MATLAB action language

For more information, see Using Fixed-Point Data in Stateflow Charts in the Stateflow User's Guide.

New 'States When Enabling' Property for Charts with Function-Call Input Events

The new chart property **States When Enabling** helps you specify how states behave when a function-call input event reenables a chart. You can select one of the following settings in the Chart properties dialog box:

- `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.

This enhancement helps you more accurately control the behavior of a chart with a function-call input event. For more information, see Controlling States When Function-Call Inputs Reenable Charts and Setting Properties for a Single Chart in the Stateflow User's Guide.

Support for Tunable Structures of Parameter Scope in Charts

You can now define structures of parameter scope that are tunable. For more information, see Defining Structures of Parameter Scope in the Stateflow User's Guide.

Enhanced Real-Time Workshop Code Generation for Noninlined State Functions

If you prevent inlining for a state, Real-Time Workshop generated code contains a new static function `inner_default_statename` when:

- Your chart contains a flow graph where an inner transition and default transition reach the same junction inside a state.
- This flow graph is complex enough to exceed the inlining threshold.

For more information, see What Happens When You Prevent Inlining in the Stateflow User's Guide.

Enhanced Real-Time Workshop Code Generation for sizeof Function

When you use the `sizeof` function in generated code to determine vector or matrix dimensions, `sizeof` always takes an input argument that evaluates to a data type.

Behavior in Prior Releases	Behavior in R2010a	Benefits of Change in Code
Input argument references the address of the variable, for example: `sizeof(&a[0])`	Input argument evaluates to the data type of the variable, for example: `sizeof(uint8_T [256])`	Ensures consistent results between simulation and code generation.

Enhanced Real-Time Workshop Code Generation for Custom-Code Function Calls

When you use custom-code function calls in generated code, vector and matrix input arguments always use pass-by-reference instead of pass-by-value behavior.

Behavior in Prior Releases	Behavior in R2010a	Benefits of Change in Code

Custom-code function calls might use either pass-by-reference or pass-by-value.
 	
For pass-by-value, a `memcpy` operation creates and stores a temporary variable in the generated code, for example:
 	
`int t[10];`
`for (i=0; i<10; i++) {`
`t[i] = y[i];`
`}`
`fcn(t);`
	Custom-code function calls use pass-by-reference, for example:
	
	`fcn(&y[0]);`
		• Ensures consistent results between simulation and code generation.
		• Less memory usage because a temporary variable is not necessary.
		• Faster execution of generated code because a `memcpy` operation is not necessary.

Data Change Implicit Event No Longer Supports Machine-Parented Data

The implicit event `change(`*`data_name`*`)` no longer works for machine-parented data. In R2010a, this implicit event works only with data at the chart level or lower in the hierarchy.

Compatibility Considerations

For machine-parented data, consider using change detection operators to determine when data values change. For more information, see Detecting Changes in Data Values in the Stateflow User's Guide.

Support for Machine-Parented Events Completely Removed

Support for machine-parented events has been completely removed. In R2010a, an error message appears when you try to simulate models that contain events at the machine level.

Compatibility Considerations

To prevent undesired behavior for simulation and code generation, do not use machine-parented events. For simulation, broadcasting an event to all charts in your model causes the following to occur:

- Charts wake up without regard to data dependencies.
- Charts that are disabled might wake up.
- Charts that use function-call or edge-triggered events wake up.
- Charts unrelated to the event wake up.
- Infinite recursive cycles can occur because the chart that broadcasts the event wakes up.

For code generation, machine-parented events prevent code reuse for the entire model.

MEX Compilation with Microsoft Visual Studio .NET 2003 No Longer Supported

Support for Microsoft Visual Studio® .NET 2003 as a MEX compiler for simulation has been removed because MATLAB and Simulink no longer support this compiler. For information about alternative compilers, see Choosing a Compiler in the Stateflow User's Guide.

Code Generation Status Messages No Longer Shown in Command Window

For Windows platforms, messages about Stateflow or Embedded MATLAB code generation and compilation status now appear only on the status bar of the Simulink Model Editor when you update diagram. Previously, these messages also appeared in the MATLAB Command Window. This enhancement minimizes distracting messages at the command prompt.

Change in Behavior for Appearance of Optimization Parameters

Previously, the Configuration Parameters dialog box showed the Stateflow section of the **Optimization** pane only when both of the following conditions were true:

- Real-Time Workshop and Stateflow licenses were available.
- Your model included Stateflow charts or Embedded MATLAB Function blocks.

In R2010a, the Configuration Parameters dialog box shows the Stateflow section of the **Optimization** pane when both licenses are available. Your model need not include any Stateflow charts or Embedded MATLAB Function blocks.

For a list of optimization parameters, see Optimization Pane: General in the Simulink Graphical User Interface.

Enhanced Inlining of Generated Code That Calls Subfunctions

In R2010a, Real-Time Workshop Embedded Coder software inlines generated code for Stateflow charts, even if the generated code calls a subfunction that accesses global Simulink data. This optimization uses less RAM and ROM.

Check Box for 'Treat as atomic unit' Now Always Selected

In existing models, simulation and code generation of Stateflow charts and Truth Table blocks always behave as if the **Treat as atomic unit** check box in the Subsystem Parameters dialog box is selected. Starting in R2010a, this check box is always selected for consistency with existing behavior.

R2009bSP1

Version: 7.4.1

Bug Fixes

R2009b

Version: 7.4

New Features

Bug Fixes

Compatibility Considerations

Ability to Copy Simulink Function-Call Subsystems and Paste in Stateflow Editor as Simulink Functions, and Vice Versa

You can copy a function-call subsystem from a model and paste directly in the Stateflow Editor. This enhancement eliminates the steps of manually creating a Simulink function in your chart and pasting the contents of the subsystem into the new function. You can also copy a Simulink function from a chart and paste directly in a model as a function-call subsystem.

For more information, see Using Simulink Functions in Stateflow Charts in the Stateflow User's Guide.

Ability to Generate Switch-Case Statements for Flow Graphs and Embedded MATLAB Functions Using Real-Time Workshop Embedded Coder Software

If a flow graph or Embedded MATLAB function in your chart uses `if-elseif-else` decision logic, you can choose to generate `switch-case` statements during Real-Time Workshop Embedded Coder code generation. `Switch-case` statements provide more readable and efficient code than `if-elseif-else` statements when multiple decision branches are possible.

When you load models created in R2009a and earlier, this optimization is off to maintain backward compatibility. In previous versions, `if-elseif-else` logic appeared unchanged in generated code.

For more information, see:

- Enhancing Readability of Generated Code for Flow Graphs
- Enhancing Readability of Generated Code for MATLAB Functions
- Code Generation Pane: Code Style

Support for Creating Switch-Case Flow Graphs Using the Pattern Wizard

In the Pattern Wizard, you can now choose to create a flow graph with `switch-case` decision logic. For more information, see Modeling Logic Patterns and Iterative Loops Using Flow Graphs in the Stateflow User's Guide.

Support for Using More Than 254 Events in a Chart

You can now use more than 254 events in a chart. The previous limit of 254 events no longer applies. This enhancement supports large-scale models with charts that send and receive hundreds of events during simulation. Although Stateflow software does not limit the number of events, the underlying C compiler enforces a theoretical limit of (2^31)-1 events for the generated code.

For more information, see Defining Events in the Stateflow User's Guide.

Improved Panning and Selection of States and Transitions When Using Stateflow Debugger

During single-step mode, the Stateflow Debugger no longer zooms automatically to the chart object that is executing. Instead, the debugger opens the subviewer that contains that object. This enhancement minimizes visual disruptions as you step through your analysis of a simulation.

For more information, see Options to Control Execution Rate in the Debugger in the Stateflow User's Guide.

Stateflow Compilation Status Added to Progress Indicator on Simulink Status Bar

For Windows platforms, messages about Stateflow compilation status now appear on the status bar of the Simulink Model Editor when you update diagram.

Support for Chart Inputs and Outputs That Vary in Dimension During Simulation

Charts now support input and output data that vary in dimension during simulation. In this release, only Embedded MATLAB functions nested in the charts can manipulate these input and output data.

For more information, see Using Variable-Size Data in Stateflow Charts and Working with Variable-Size Data in MATLAB Functions in the Stateflow User's Guide.

New Compilation Report for Embedded MATLAB Functions in Stateflow Charts

The new compilation report provides compile-time type information for the variables and expressions in your Embedded MATLAB functions. This information helps you find the sources of error messages and understand type propagation issues, particularly for fixed-point data types. For more information, see Working with MATLAB Function Reports in the Simulink User's Guide.

Compatibility Considerations

The new compilation report is not supported by the MATLAB internal browser on Sun™ Solaris™ 64-bit platforms. To view the compilation report on Sun Solaris 64-bit platforms, you must have your MATLAB Web preferences configured to use an external browser, for example, Mozilla® Firefox®. To learn how to configure your MATLAB Web preferences, see Web Preferences.

Enhanced Support for Replacing Math Functions with Target-Specific Implementations

You can now replace the following math functions with target-specific implementations:

Function	Data Type Support
atan2	Floating-point

Function	Data Type Support
`fmod`	Floating-point
`ldexp`	Floating-point
`max`	Floating-point and integer
`min`	Floating-point and integer

Replacement of `abs` now works for both floating-point and integer arguments. Previously, replacement of `abs` with a target function worked only for floating-point arguments.

For more information about Target Function Libraries, see:

- Replacement of C Math Library Functions with Target-Specific Implementations
- Replacing Operators with Target-Specific Implementations

Enhanced Context Menu Support for Adding Flow Graph Patterns to Charts

In a chart, you can now right-click at any level of the hierarchy (for example, states and subcharts) to insert flow graphs using the **Patterns** context menu. Previously, options in this context menu were available only if you right-clicked at the chart level.

Option to Log Chart Signals Available in the Stateflow Editor

To log chart signals, you can select **Tools > Log Chart Signals** in the Stateflow Editor. Previously, you had to right-click the Stateflow block in the Model Editor to open the Signal Logging dialog box.

For more information, see What You Can Log During Chart Simulation in the Stateflow User's Guide.

Default Precision Set to Double for Calls to C Math Functions

When you call C math functions, such as `sin`, `exp`, or `pow`, double precision applies unless the first input argument is explicitly single precision. For example, if you call the `sin` function with an integer argument, a cast of the input argument to a floating-point number of type `double` replaces the original argument. This behavior ensures consistent results between Simulink blocks and Stateflow charts for calls to C math functions.

To force a call to a single-precision version of a C math function, you must explicitly cast the function argument using the `single` cast operator. This method works only when a single-precision version of the function exists in the selected Target Function Library as it would in the 'C99 (ISO)' Target Function Library. For more information, see Calling C Functions in Actions and Type Cast Operations in the Stateflow User's Guide.

Change in Text and Visibility of Parameter Prompt for Easier Use with Fixed-Point Advisor and Fixed-Point Tool

In the Data properties dialog box, the **Lock output scaling against changes by the autoscaling tool** check box is now **Lock data type setting against changes by the fixed-point tools**. Previously, this check box was visible only if you entered an expression or a fixed-point data type, such as `fixdt(1,16,0)`. This check box is now visible for any data type specification. This

enhancement enables you to lock the current data type settings on the dialog box against changes that the Fixed-Point Advisor or Fixed-Point Tool chooses.

For more information, see Fixed-Point Data Properties and Automatic Scaling of Stateflow Fixed-Point Data in the Stateflow User's Guide.

Charts Closed By Default When Opening Models Saved in Formats of Earlier Versions

If you save a model with Stateflow charts in the format of an earlier version, the charts appear closed when you open the new MDL-file.

R2009a

Version: 7.3

New Features

Bug Fixes

Compatibility Considerations

Support for Saving the Complete Simulation State at a Specific Time

You can save the complete simulation state at a specific time and then load that state for further simulation. This enhancement provides these benefits:

- Enables running isolated segments of a simulation without starting from time t = 0, which saves time
- Enables testing of the same chart configuration with different settings
- Enables testing of hard-to-reach chart configurations by loading a specific simulation state

For more information, see Saving and Restoring Simulations with SimState in the Stateflow User's Guide.

Enhanced Support for Enumerated Data Types

In R2009a, you can use enumerated data in Embedded MATLAB functions, truth table functions that use Embedded MATLAB action language, and Truth Table blocks. See Using Enumerated Data in Stateflow Charts in the Stateflow User's Guide.

New Boolean Keywords in Stateflow Action Language

You can now use `true` and `false` as Boolean keywords in Stateflow action language. For more information, see Supported Symbols in Actions in the Stateflow User's Guide.

Enhanced Control of Inlining State Functions in Generated Code

In R2009a, a new **Function Inline Option** parameter is available in the State properties dialog box. This parameter enables better control of inlining state functions in generated code, which provides these benefits:

- Prevents small changes to a model from causing major changes to the structure of generated code
- Enables easier manual inspection of generated code, because of a one-to-one mapping between the code and the model

For more information, see Controlling Inlining of State Functions in Generated Code in the Stateflow User's Guide.

New Diagnostic to Detect Unintended Backtracking Behavior in Flow Graphs

A new diagnostic detects unintended backtracking behavior in flow graphs during simulation. A warning message appears, with suggestions on how to fix the flow graph to prevent unintended backtracking. For more information, see Best Practices for Creating Flow Graphs in the Stateflow User's Guide.

Use of Basic Linear Algebra Subprograms (BLAS) Libraries for Speed

Embedded MATLAB functions in Stateflow charts can now use BLAS libraries to speed up low-level matrix operations during simulation. For more information, see Simulation Target Pane: General.

Enhanced Support for Replacing C Math Functions with Target-Specific Implementations

You can now replace the `pow` function with a target-specific implementation. For more information about Target Function Libraries, see:

- Replacement of C Math Library Functions with Target-Specific Implementations
- Replacing Operators with Target-Specific Implementations

Smart Transitions Now Prefer Straight Lines

In R2009a, the graphical behavior of smart transitions has been enhanced as follows:

- Smart transitions maintain straight lines between states and junctions whenever possible. Previously, smart transitions would preserve curved lines.
- When you drag a smart transition radially around a junction, the end on the junction follows the tip to maintain a straight line by default. Previously, the end on the junction would maintain its original location and not follow the tip of the transition.

For more information, see What Smart Transitions Do in the Stateflow User's Guide.

Clicking Up-Arrow Button in the Stateflow Editor Closes Top-Level Chart

When a top-level chart appears in the Stateflow Editor, clicking the up-arrow button in the toolbar causes the chart to close and the Simulink model that contains the chart to appear. This behavior is consistent with clicking the up-arrow button in the toolbar of a Simulink subsystem window.

Previously, clicking the up-arrow button for a top-level chart would cause the Simulink model to appear, but the chart would not close. For more information, see Navigating Subcharts in the Stateflow User's Guide.

Enhanced Type Resolution for Symbols

In R2009a, type resolution for Stateflow data has been enhanced to support any MATLAB expression that evaluates to a type.

Enhanced Code Generation for Stateflow Events

In R2009a, the generated code for managing Stateflow events uses a deterministic numbering method. This enhancement minimizes unnecessary differences in the generated code for charts between R2009a and any future release.

Enhanced Real-Time Workshop Generated Code for Charts with Simulink Functions

In R2009a, Real-Time Workshop generated code for charts with Simulink functions no longer uses unneeded global variables for the function inputs and outputs. The interface can be represented by

local temporary variables or completely eliminated by optimizations, such as expression folding. This enhancement provides reduced RAM consumption and faster execution time.

Use of en, du, ex, entry, during, and exit for Data and Event Names Being Disallowed in a Future Version

In a future version of Stateflow software, use of `en`, `du`, `ex`, `entry`, `during`, or `exit` for naming data or events will be disallowed. In R2009a, a warning message appears when you run a model that contains any of these keywords as the names of data or events.

Compatibility Considerations

To avoid warning messages, rename any data or event that uses `en`, `du`, `ex`, `entry`, `during`, or `exit` as an identifier.

Support for Machine-Parented Events Being Removed in a Future Version

In a future version of Stateflow software, support for machine-parented events will be removed. In R2009a, a warning message appears when you simulate models that contain events at the machine level.

Compatibility Considerations

To prevent undesired behavior for simulation and code generation, do not use machine-parented events. For simulation, broadcasting an event to all charts in your model causes the following to occur:

- Charts wake up without regard to data dependencies.
- Charts that are disabled might wake up.
- Charts that use function-call or edge-triggered events wake up.
- Charts unrelated to the event wake up.
- Infinite recursive cycles can occur because the chart that broadcasts the event wakes up.

For code generation, machine-parented events prevent code reuse for the entire model.

R2008b

Version: 7.2

New Features

Bug Fixes

Compatibility Considerations

Support for Embedding Simulink Function-Call Subsystems in a Stateflow Chart

You can use a Simulink function to embed a function-call subsystem in a Stateflow chart. You fill this function with Simulink blocks and call it in state actions and on transitions. Like graphical functions, truth table functions, and Embedded MATLAB functions, you can use multiple return values with Simulink functions.

For more information, see Using Simulink Functions in Stateflow Charts in the Stateflow User's Guide.

Support for Using Enumerated Data Types in a Stateflow Chart

You can use data of an enumerated type in a Stateflow chart.

For more information, see Using Enumerated Data in Stateflow Charts in the Stateflow User's Guide and Enumerations and Modeling in the Simulink User's Guide.

New Alignment, Distribution, and Resizing Commands for Stateflow Charts

You can use alignment, distribution, and resizing commands on graphical chart objects, such as states, functions, and boxes.

For more information, see Formatting Chart Objects in the Stateflow User's Guide.

Unified Simulation and Embeddable Code Generation Options for Stateflow Charts and Truth Table Blocks
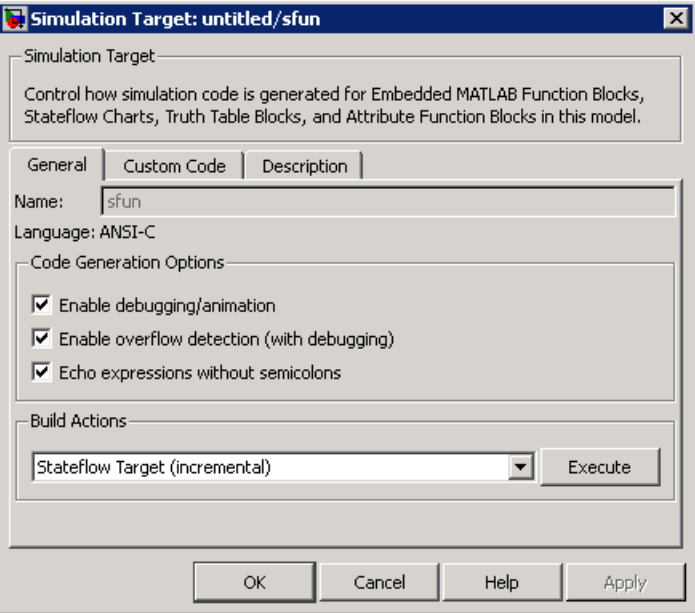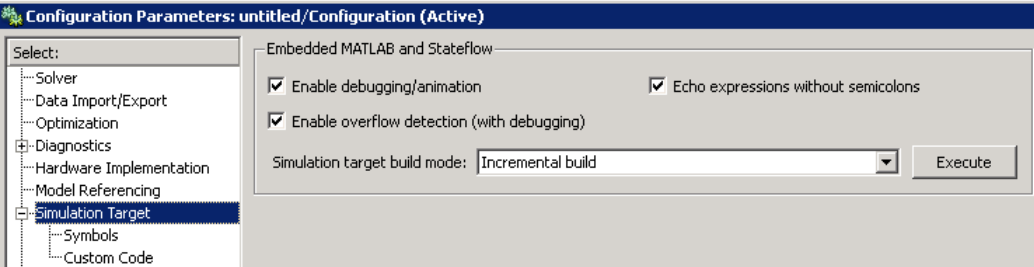
You can use a single dialog box to specify simulation and embeddable code generation options that apply to Stateflow charts and Truth Table blocks. These changes apply:

Type of Model	Simulation Options	Embeddable Code Generation Options
Nonlibrary	Migrated from the Simulation Target dialog box to the Configuration Parameters dialog box	Enhanced with new options in the **Real-Time Workshop** pane of the Configuration Parameters dialog box
Library	Migrated from the Simulation Target dialog box to the Configuration Parameters dialog box	Migrated from the RTW Target dialog box to the Configuration Parameters dialog box
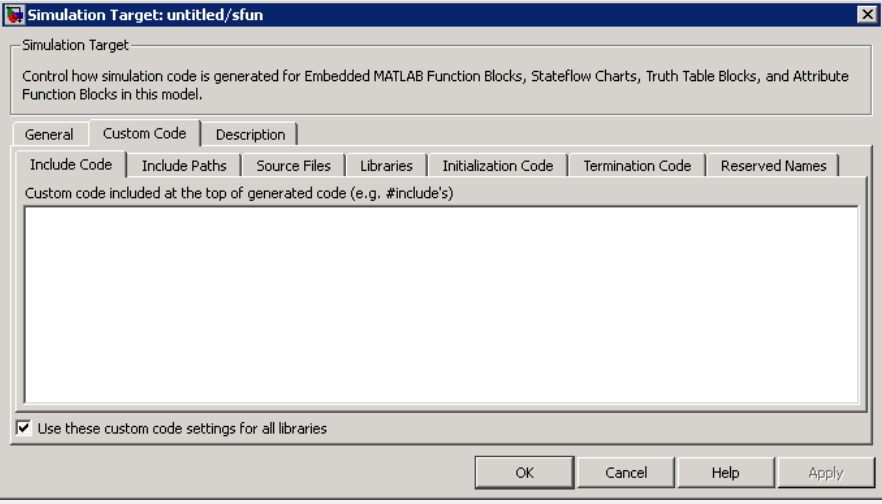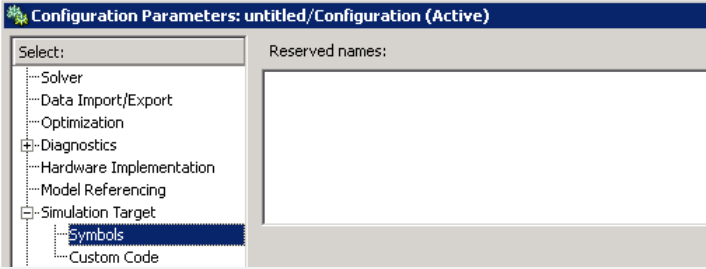
GUI Changes in Simulation Options for Nonlibrary Models
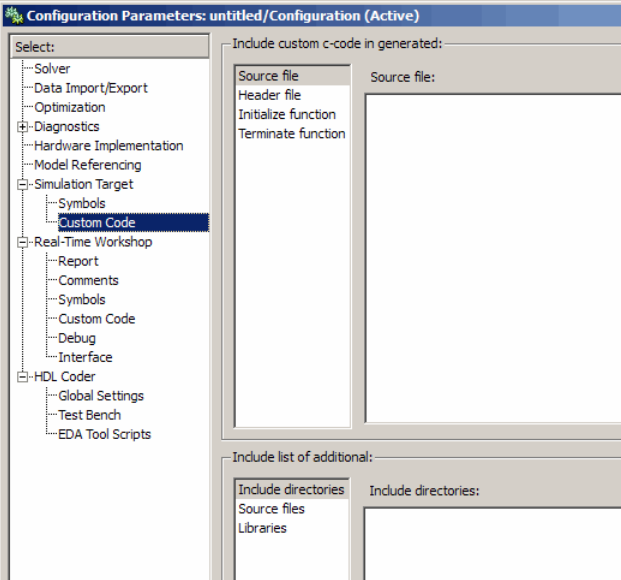
The following sections describe changes in the panes of the Simulation Target dialog box for nonlibrary models.

Changes for the General Pane of the Simulation Target Dialog Box

Release	Appearance
Previous	**General** pane of the Simulation Target dialog box
New	**Simulation Target** pane of the Configuration Parameters dialog box

Changes for the Custom Code Pane of the Simulation Target Dialog Box

Release	Appearance
Previous	**Custom Code** pane of the Simulation Target dialog box
New	**Simulation Target > Symbols** pane of the Configuration Parameters dialog box

Release	Appearance
New	**Simulation Target > Custom Code** pane of the Configuration Parameters dialog box

Changes for the Description Pane of the Simulation Target Dialog Box

In previous releases, the **Description** pane of the Simulation Target dialog box appeared as follows.

In R2008b, these options are no longer available. For older models where the **Description** pane contained information, the text is now accessible only in the Model Explorer. When you select **Simulink Root > Configuration Preferences** in the **Model Hierarchy** pane, the text appears in the **Description** field for that model.

Nonlibrary Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box

For nonlibrary models, the following table maps each GUI option in the Simulation Target dialog box to the equivalent in the Configuration Parameters dialog box. The options are listed in order of appearance in the Simulation Target dialog box.

Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
General > Enable debugging / animation	**Simulation Target > Enable debugging / animation**	on
General > Enable overflow detection (with debugging)	**Simulation Target > Enable overflow detection (with debugging)**	on
General > Echo expressions without semicolons	**Simulation Target > Echo expressions without semicolons**	on
General > Build Actions	**Simulation Target > Simulation target build mode**	`Incremental build`
None	**Simulation Target > Custom Code > Source file**	`''`
Custom Code > Include Code	**Simulation Target > Custom Code > Header file**	`''`
Custom Code > Include Paths	**Simulation Target > Custom Code > Include directories**	`''`
Custom Code > Source Files	**Simulation Target > Custom Code > Source files**	`''`
Custom Code > Libraries	**Simulation Target > Custom Code > Libraries**	`''`
Custom Code > Initialization Code	**Simulation Target > Custom Code > Initialize function**	`''`
Custom Code > Termination Code	**Simulation Target > Custom Code > Terminate function**	`''`
Custom Code > Reserved Names	**Simulation Target > Symbols > Reserved names**	`{}`
Custom Code > Use these custom code settings for all libraries	None	Not applicable

Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
Description > Description	None **Note** If you load an older model that contained user-specified text in the **Description** field, that text now appears in the Model Explorer. When you select **Simulink Root > Configuration Preferences** in the **Model Hierarchy** pane, the text appears in the **Description** field for that model.	Not applicable
Description > Document Link	None	Not applicable

Note For nonlibrary models, **Simulation Target** options in the Configuration Parameters dialog box are also available in the Model Explorer. When you select **Simulink Root > Configuration Preferences** in the **Model Hierarchy** pane, you can select **Simulation Target** in the **Contents** pane to access the options.

GUI Changes in Simulation Options for Library Models

The following sections describe changes in the panes of the Simulation Target dialog box for library models.
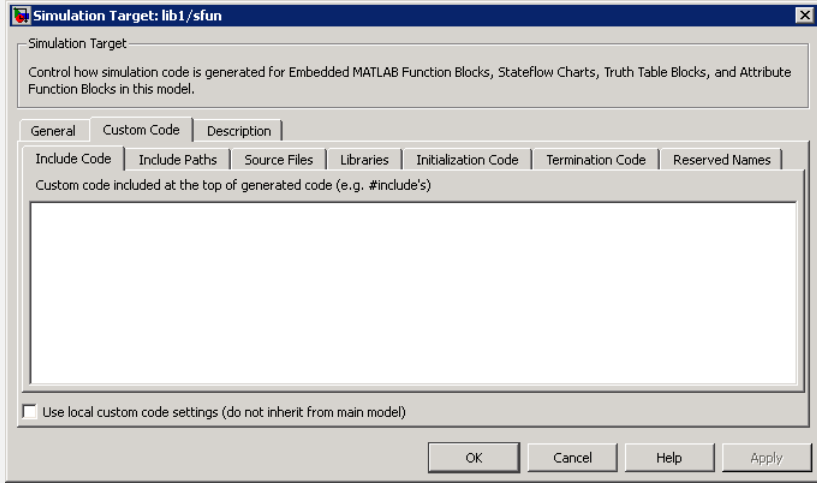
Changes for the General Pane of the Simulation Target Dialog Box
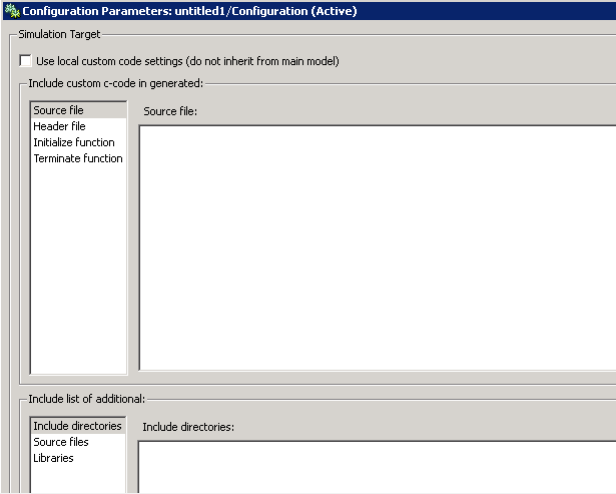
In previous releases, the **General** pane of the Simulation Target dialog box for library models appeared as follows.
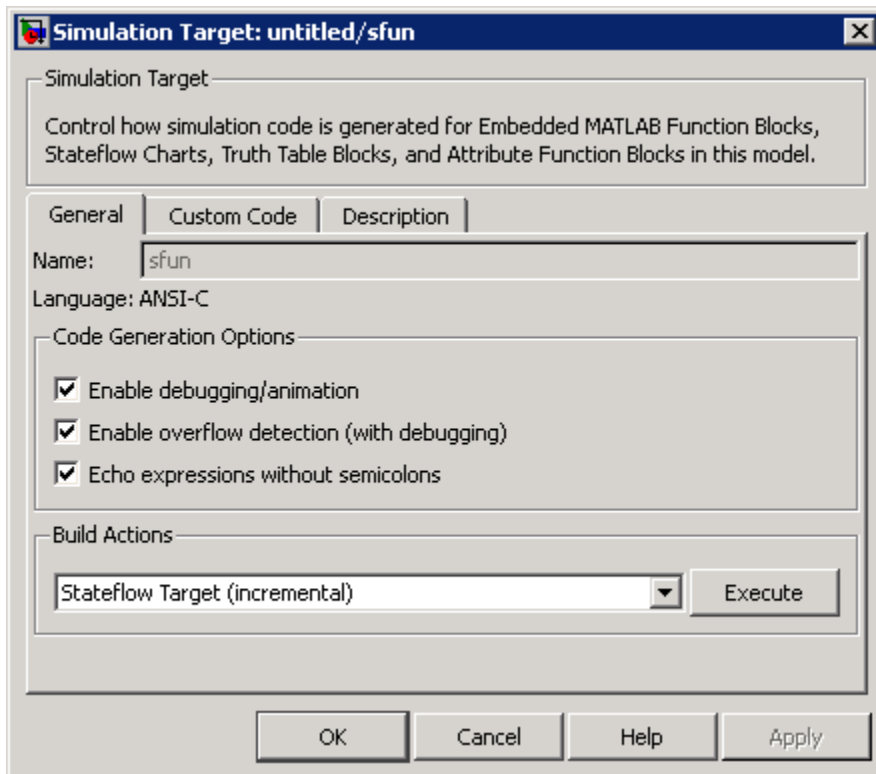
In R2008b, these options are no longer available. All library models inherit these option settings from the main model to which the libraries are linked.

Changes for the Custom Code Pane of the Simulation Target Dialog Box

Release	Appearance
Previous	**Custom Code** pane of the Simulation Target dialog box

Release	Appearance
New	**Simulation Target** pane of the Configuration Parameters dialog box **Configuration Parameters: untitled1/Configuration (Active)** — Simulation Target — ☐ Use local custom code settings (do not inherit from main model) — Include custom c-code in generated: — Source file / Header file / Initialize function / Terminate function Source file: — Include list of additional: — Include directories / Source files / Libraries Include directories:

Changes for the Description Pane of the Simulation Target Dialog Box

In previous releases, the **Description** pane of the Simulation Target dialog box appeared as follows.

Simulation Target: untitled/sfun

Simulation Target

Control how simulation code is generated for Embedded MATLAB Function Blocks, Stateflow Charts, Truth Table Blocks, and Attribute Function Blocks in this model.

| General | Custom Code | Description |

Description

Default Simulink S-Function Target.

Document Link:

OK   Cancel   Help   Apply

In R2008b, these options are no longer available. For older models where the **Description** pane contained information, the text is discarded.

Library Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box

For library models, the following table maps each GUI option in the Simulation Target dialog box to the equivalent in the Configuration Parameters dialog box. The options are listed in order of appearance in the Simulation Target dialog box.

Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
General > Enable debugging / animation	None	Not applicable
General > Enable overflow detection (with debugging)	None	Not applicable
General > Echo expressions without semicolons	None	Not applicable
General > Build Actions	None	Not applicable
None	**Simulation Target > Source file**	' '
Custom Code > Include Code	**Simulation Target > Header file**	' '
Custom Code > Include Paths	**Simulation Target > Include directories**	' '
Custom Code > Source Files	**Simulation Target > Source files**	' '
Custom Code > Libraries	**Simulation Target > Libraries**	' '
Custom Code > Initialization Code	**Simulation Target > Initialize function**	' '
Custom Code > Termination Code	**Simulation Target > Terminate function**	' '
Custom Code > Reserved Names	None	Not applicable
Custom Code > Use local custom code settings (do not inherit from main model)	**Simulation Target > Use local custom code settings (do not inherit from main model)**	off
Description > Description	None	Not applicable
Description > Document Link	None	Not applicable

Note For library models, **Simulation Target** options in the Configuration Parameters dialog box are not available in the Model Explorer.

GUI Enhancements in Real-Time Workshop Code Generation Options for Nonlibrary Models

The following sections describe enhancements to the **Real-Time Workshop** pane of the Configuration Parameters dialog box for nonlibrary models.

Enhancement for the Real-Time Workshop: Symbols Pane of the Configuration Parameters Dialog Box

In previous releases, the **Real-Time Workshop > Symbols** pane of the Configuration Parameters dialog box appeared as follows.
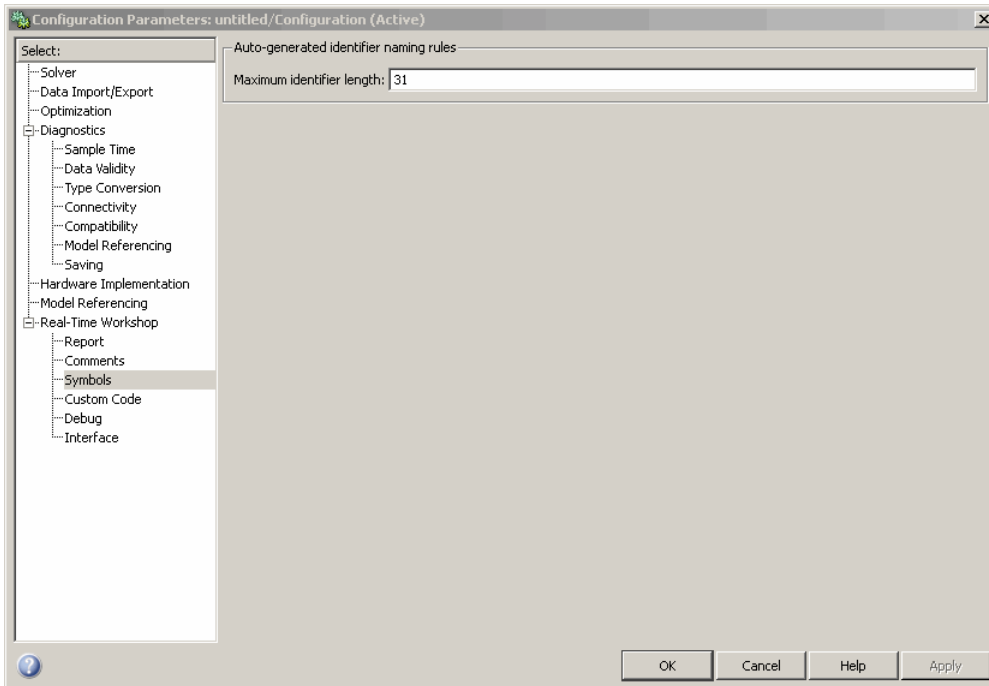
In R2008b, a new option is available in this pane: **Reserved names**. You can use this option to specify a set of keywords that the Real-Time Workshop build process should not use. This action prevents naming conflicts between functions and variables from external environments and identifiers in the generated code.
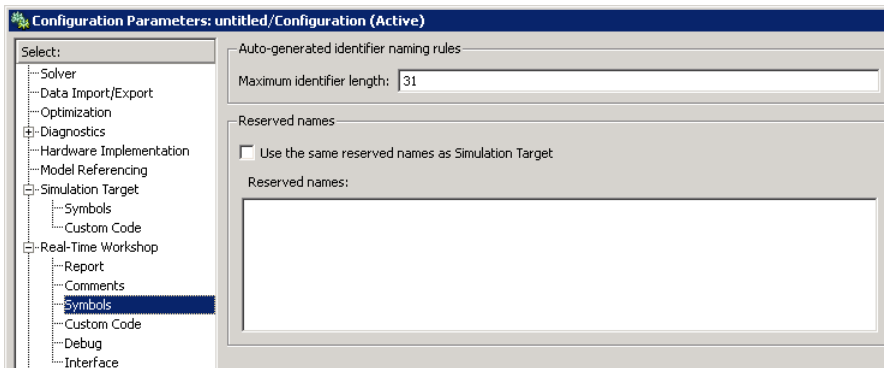
You can also choose to use the reserved names specified in the **Simulation Target > Symbols** pane to avoid entering the same information twice for the nonlibrary model. Select the **Use the same reserved names as Simulation Target** check box.

Enhancement for the Real-Time Workshop: Custom Code Pane of the Configuration Parameters Dialog Box

In previous releases, the **Real-Time Workshop > Custom Code** pane of the Configuration Parameters dialog box appeared as follows.
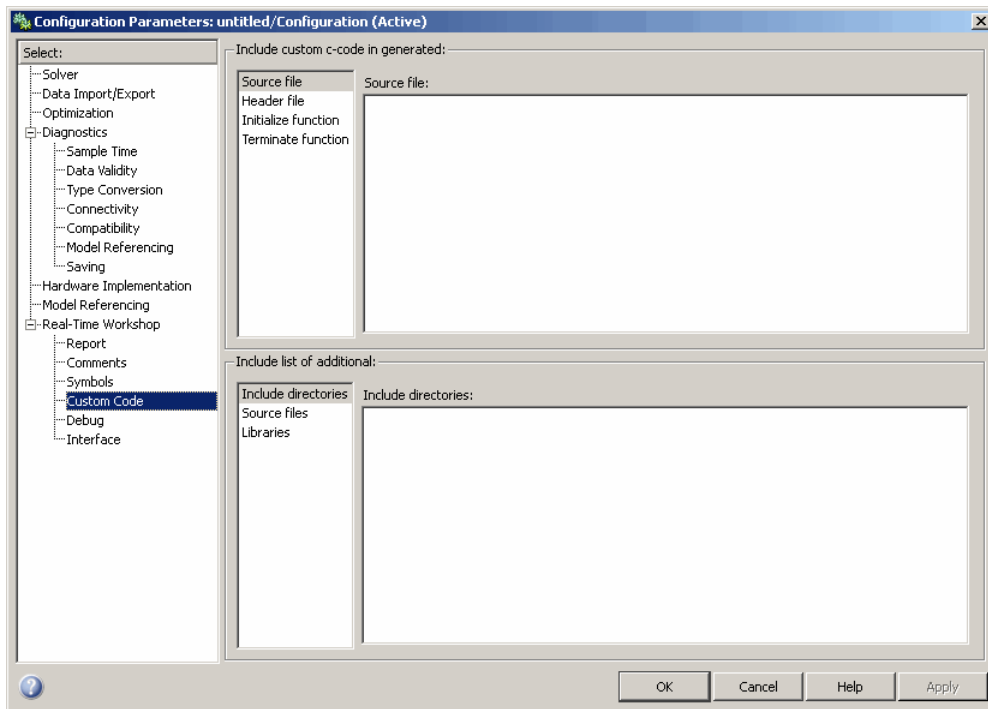
In R2008b, a new option is available in this pane: **Use the same custom code settings as Simulation Target**. You can use this option to copy the custom code settings from the **Simulation Target > Custom Code** pane to avoid entering the same information twice for the nonlibrary model.
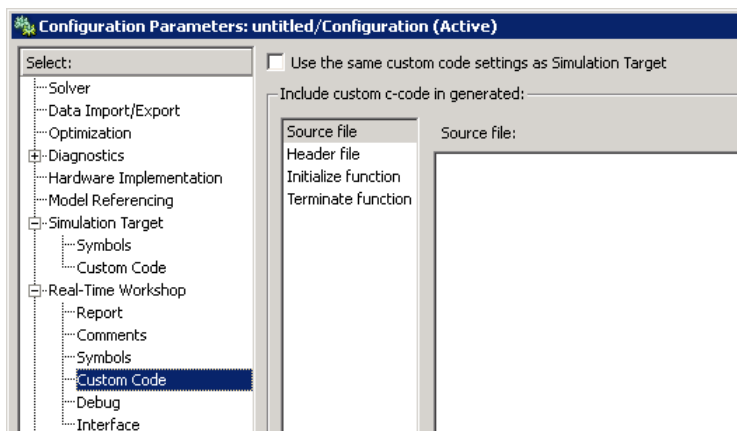
GUI Changes in Real-Time Workshop Code Generation Options for Library Models

The following sections describe changes in the panes of the RTW Target dialog box for library models.
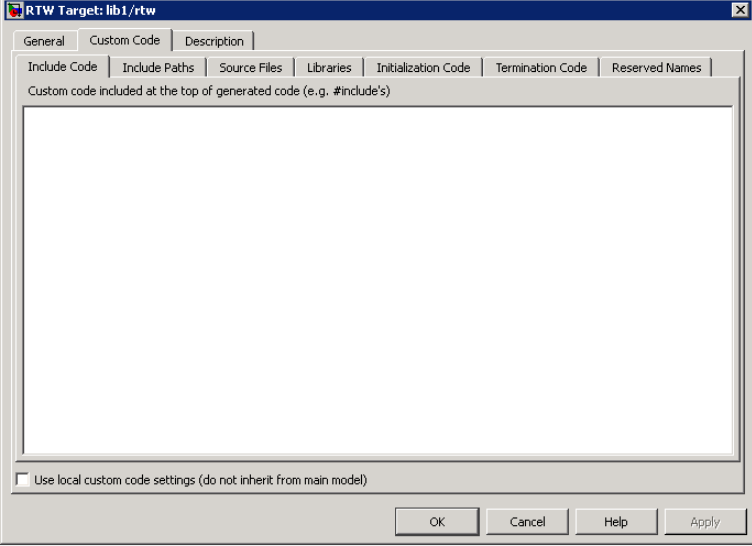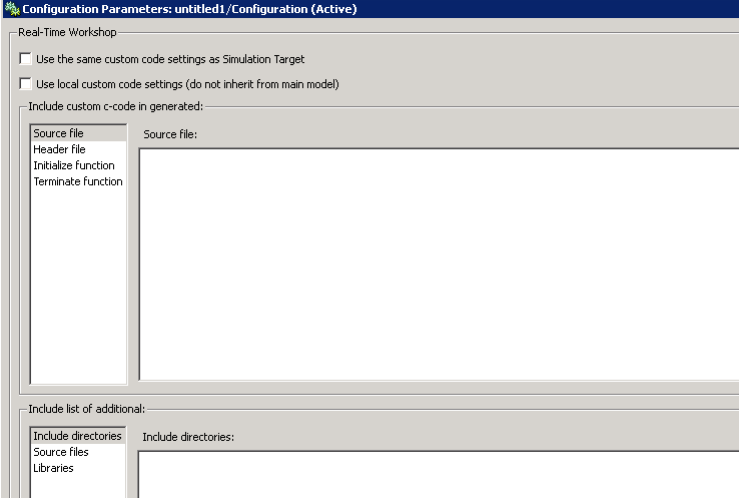
Changes for the General Pane of the RTW Target Dialog Box

In previous releases, the **General** pane of the RTW Target dialog box for library models appeared as follows.

RTW Target: lib1/rtw

General | Custom Code | Description

Name: rtw

Language: ANSI-C

Code Generation Options

☑ Comments in generated code

☑ Use bitsets for storing state configuration

☑ Use bitsets for storing boolean data

☑ Compact nested if-else using logical AND/OR operators

☑ Recognize if-elseif-else in nested if-else statements

☑ Replace constant expressions by a single constant

☐ Minimize array reads using temporary variables

☐ Preserve symbol names

☐ Append symbol names with parent names

☐ Use chart names with no mangling

Build Actions

Stateflow Target ▾ | Execute

OK | Cancel | Help | Apply
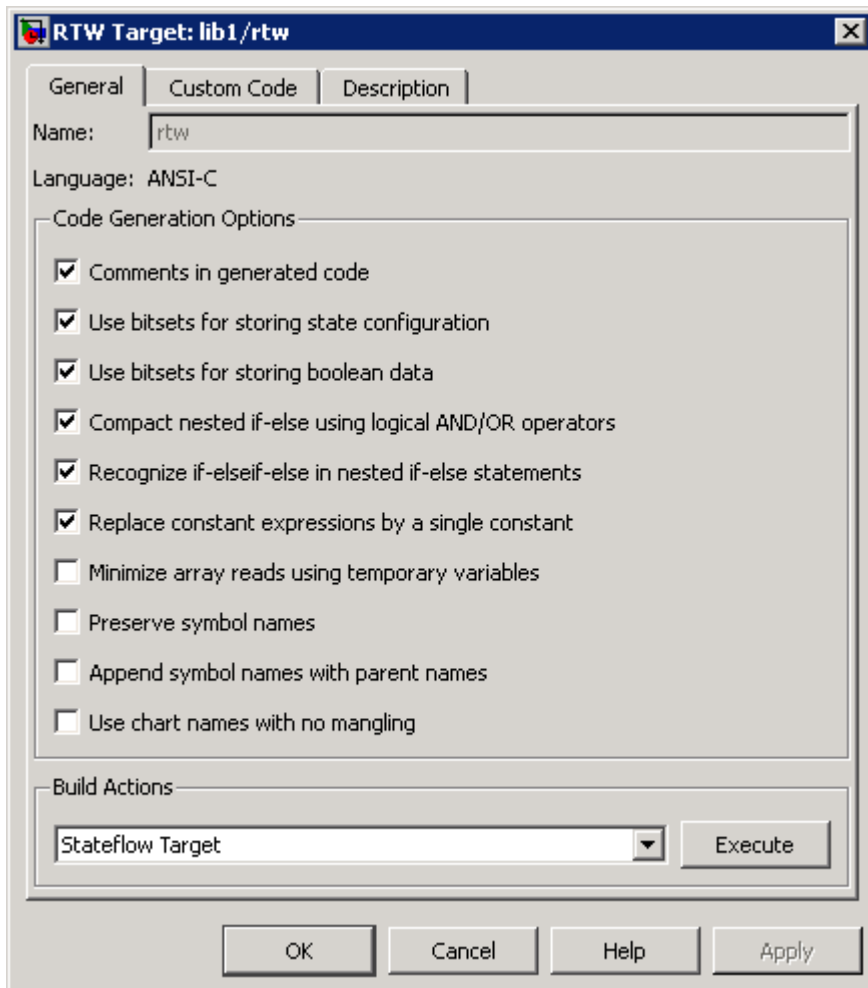
In R2008b, these options are no longer available. During Real-Time Workshop code generation, options specified for the main model are used.

Changes for the Custom Code Pane of the RTW Target Dialog Box

Release	Appearance
Previous	**Custom Code** pane of the RTW Target dialog box
New	**Real-Time Workshop** pane of the Configuration Parameters dialog box

Changes for the Description Pane of the RTW Target Dialog Box

In previous releases, the **Description** pane of the RTW Target dialog box appeared as follows.
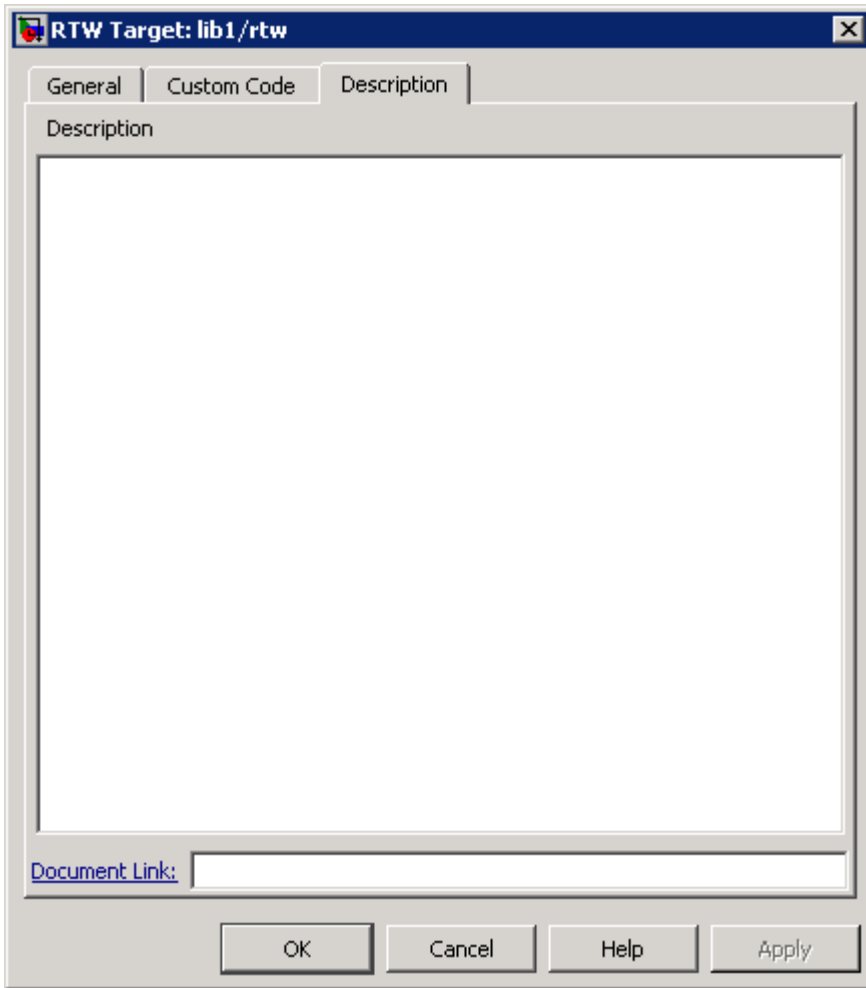
In R2008b, these options are no longer available. For older models where the **Description** pane contained information, the text is discarded.

Library Models: Mapping of GUI Options from the RTW Target Dialog Box to the Configuration Parameters Dialog Box

For library models, the following table maps each GUI option in the RTW Target dialog box to the equivalent in the Configuration Parameters dialog box. The options are listed in order of appearance in the RTW Target dialog box.

Old Option in the RTW Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
General > Comments in generated code	None	Not applicable
General > Use bitsets for storing state configuration	None	Not applicable
General > Use bitsets for storing boolean data	None	Not applicable

| Old Option in the RTW Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
General > Compact nested if-else using logical AND/OR operators	None	Not applicable
General > Recognize if-elseif-else in nested if-else statements	None	Not applicable
General > Replace constant expressions by a single constant	None	Not applicable
General > Minimize array reads using temporary variables	None	Not applicable
General > Preserve symbol names	None	Not applicable
General > Append symbol names with parent names	None	Not applicable
General > Use chart names with no mangling	None	Not applicable
General > Build Actions	None	Not applicable
None	**Real-Time Workshop > Source file**	' '
Custom Code > Include Code	**Real-Time Workshop > Header file**	' '
Custom Code > Include Paths	**Real-Time Workshop > Include directories**	' '
Custom Code > Source Files	**Real-Time Workshop > Source files**	' '
Custom Code > Libraries	**Real-Time Workshop > Libraries**	' '
Custom Code > Initialization Code	**Real-Time Workshop > Initialize function**	' '
Custom Code > Termination Code	**Real-Time Workshop > Terminate function**	' '
Custom Code > Reserved Names	None	Not applicable
Custom Code > Use local custom code settings (do not inherit from main model)	**Real-Time Workshop > Use local custom code settings (do not inherit from main model)**	off
None	**Real-Time Workshop > Use the same custom code settings as Simulation Target**	off
Description > Description	None	Not applicable
Description > Document Link	None	Not applicable

Note For library models, **Real-Time Workshop** options in the Configuration Parameters dialog box are not available in the Model Explorer.

Mapping of Target Object Properties to Parameters in the Configuration Parameters Dialog Box

Previously, you could programmatically set options for simulation and embeddable code generation by accessing the API properties of Target objects `sfun` and `rtw`, respectively. In R2008b, the API properties of Target objects `sfun` and `rtw` are replaced by parameters that you configure using the commands `get_param` and `set_param`.

Mapping of Object Properties to Simulation Parameters for Nonlibrary Models

The following table maps API properties of the Target object `sfun` for nonlibrary models to the equivalent parameters in R2008b. Object properties are listed in alphabetical order; those not listed in the table do not have equivalent parameters in R2008b.

Old `sfun` Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
`CodeFlagsInfo ('debug')`	**General > Enable debugging / animation**	`SFSimEnableDebug` string - `off`, **`on`**	**Simulation Target > Enable debugging / animation**
`CodeFlagsInfo ('overflow')`	**General > Enable overflow detection (with debugging)**	`SFSimOverflowDetection` string - `off`, **`on`**	**Simulation Target > Enable overflow detection (with debugging)**
`CodeFlagsInfo ('echo')`	**General > Echo expressions without semicolons**	`SFSimEcho` string - `off`, **`on`**	**Simulation Target > Echo expressions without semicolons**
`CustomCode`	**Custom Code > Include Code**	`SimCustomHeaderCode` *string* - `''`	**Simulation Target > Custom Code > Header file**
`CustomInitializer`	**Custom Code > Initialization Code**	`SimCustomInitializer` *string* - `''`	**Simulation Target > Custom Code > Initialize function**
`CustomTerminator`	**Custom Code > Termination Code**	`SimCustomTerminator` *string* - `''`	**Simulation Target > Custom Code > Terminate function**
`ReservedNames`	**Custom Code > Reserved Names**	`SimReservedNameArray` *string array* - **`{}`**	**Simulation Target > Symbols > Reserved names**
`UserIncludeDirs`	**Custom Code > Include Paths**	`SimUserIncludeDirs` *string* - `''`	**Simulation Target > Custom Code > Include directories**
`UserLibraries`	**Custom Code > Libraries**	`SimUserLibraries` *string* - `''`	**Simulation Target > Custom Code > Libraries**

Old sfun Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
UserSources	**Custom Code > Source Files**	SimUserSources *string* - ''	**Simulation Target > Custom Code > Source files**

Mapping of Object Properties to Simulation Parameters for Library Models

The following table maps API properties of the Target object `sfun` for library models to the equivalent parameters in R2008b. Object properties are listed in alphabetical order; those not listed in the table do not have equivalent parameters in R2008b.

Old sfun Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CustomCode	**Custom Code > Include Code**	SimCustomHeaderCode *string* - ''	**Simulation Target > Header file**
CustomInitializer	**Custom Code > Initialization Code**	SimCustomInitializer *string* - ''	**Simulation Target > Initialize function**
CustomTerminator	**Custom Code > Termination Code**	SimCustomTerminator *string* - ''	**Simulation Target > Terminate function**
UseLocalCustomCodeSettings	**Custom Code > Use local custom code settings (do not inherit from main model)**	SimUseLocalCustomCode string - **off**, on	**Simulation Target > Use local custom code settings (do not inherit from main model)**
UserIncludeDirs	**Custom Code > Include Paths**	SimUserIncludeDirs *string* - ''	**Simulation Target > Include directories**
UserLibraries	**Custom Code > Libraries**	SimUserLibraries *string* - ''	**Simulation Target > Libraries**
UserSources	**Custom Code > Source Files**	SimUserSources *string* - ''	**Simulation Target > Source files**

Mapping of Object Properties to Code Generation Parameters for Library Models

The following table maps API properties of the Target object `rtw` for library models to the equivalent parameters in R2008b. Object properties are listed in alphabetical order; those not listed in the table do not have equivalent parameters in R2008b.

Old `rtw` Object Property	Old Option in the RTW Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CustomCode	**Custom Code > Include Code**	CustomHeaderCode *string* - ''	**Real-Time Workshop > Header file**
CustomInitializer	**Custom Code > Initialization Code**	CustomInitializer *string* - ''	**Real-Time Workshop > Initialize function**
CustomTerminator	**Custom Code > Termination Code**	CustomTerminator *string* - ''	**Real-Time Workshop > Terminate function**
UseLocalCustomCodeSettings	**Custom Code > Use local custom code settings (do not inherit from main model)**	RTWUseLocalCustomCode string - **off**, on	**Real-Time Workshop > Use local custom code settings (do not inherit from main model)**
UserIncludeDirs	**Custom Code > Include Paths**	CustomInclude *string* - ''	**Real-Time Workshop > Include directories**
UserLibraries	**Custom Code > Libraries**	CustomLibrary *string* - ''	**Real-Time Workshop > Libraries**
UserSources	**Custom Code > Source Files**	CustomSource *string* - ''	**Real-Time Workshop > Source files**

New Parameters in the Configuration Parameters Dialog Box for Simulation and Embeddable Code Generation

In R2008b, new parameters are added to the Configuration Parameters dialog box for simulation and embeddable code generation.

New Simulation Parameters for Nonlibrary Models

The following table lists the new simulation parameters that apply to nonlibrary models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
SimBuildMode string – **sf_incremental_build**, sf_nonincremental_build, sf_make, sf_make_clean, sf_make_clean_objects	**Simulation Target > Simulation target build mode**	Specifies how you build the simulation target for a model.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
SimCustomSourceCode *string* - ''	**Simulation Target > Custom Code > Source file**	Enter code lines to appear near the top of a generated source code file.

New Simulation Parameter for Library Models

The following table lists the new simulation parameter that applies to library models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
SimCustomSourceCode *string* - ''	**Simulation Target > Source file**	Enter code lines to appear near the top of a generated source code file.

New Code Generation Parameters for Nonlibrary Models

The following table lists the new code generation parameters that apply to nonlibrary models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
ReservedNameArray *string array* - **{}**	**Real-Time Workshop > Symbols > Reserved names**	Enter the names of variables or functions in the generated code that match the names of variables or functions specified in custom code.
RTWUseSimCustomCode string – **off**, on	**Real-Time Workshop > Custom Code > Use the same custom code settings as Simulation Target**	Specify whether to use the same custom code settings as those specified for simulation.
UseSimReservedNames string – **off**, on	**Real-Time Workshop > Symbols > Use the same reserved names as Simulation Target**	Specify whether to use the same reserved names as those specified for simulation.

New Code Generation Parameters for Library Models

The following table lists the new code generation parameters that apply to library models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
CustomSourceCode *string* – ''	**Real-Time Workshop > Source file**	Enter code lines to appear near the top of a generated source code file.
RTWUseSimCustomCode string – **off**, on	**Real-Time Workshop > Use the same custom code settings as Simulation Target**	Specify whether to use the same custom code settings as those specified for simulation.

Compatibility Considerations

Updating Scripts That Set Options Programmatically for Simulation and Embeddable Code Generation

In previous releases, you could use the Stateflow API to set options for simulation and embeddable code generation by accessing the Target object (`sfun` or `rtw`) in a Stateflow machine. For example, you could set simulation options programmatically by running these commands in a MATLAB script:

```
r = slroot;
machine = r.find('-isa','Stateflow.Machine','Name','main_mdl');
t_sim = machine.find('-isa','Stateflow.Target','Name','sfun');
t_sim.setCodeFlag('debug',1);
t_sim.setCodeFlag('overflow',1);
t_sim.setCodeFlag('echo',1);
t_sim.getCodeFlag('debug');
t_sim.getCodeFlag('overflow');
t_sim.getCodeFlag('echo');
```

In R2008b, you must update your scripts to use the `set_param` and `get_param` commands to configure simulation and embeddable code generation. For example, you can update the previous script as follows:

```
cs = getActiveConfigSet(gcs);
set_param(cs,'SFSimEnableDebug','on');
set_param(cs,'SFSimOverflowDetection','on');
set_param(cs,'SFSimEcho','on');
get_param(cs,'SFSimEnableDebug');
get_param(cs,'SFSimOverflowDetection');
get_param(cs,'SFSimEcho');
```

For information about...	See...
Object properties and their equivalent parameters in R2008b	Properties of Target objects `sfun` and `rtw` that are no longer supported in R2008b cannot be updated using the command-line API.
Using the `set_param` and `get_param` commands	Using Command-Line API to Set Simulation and Code Generation Parameters.

Accessing Target Options for Library Models

In previous releases, you could access target options for library models via the **Tools** menu in the Stateflow Editor or the **Contents** pane of the Model Explorer. In R2008b, you must use the **Tools** menu to access target options for library models. For example, to specify parameters for the simulation target, select **Tools > Open Simulation Target** in the Stateflow Editor.

What Happens When You Load an Older Model in R2008b

When you use R2008b to load a model created in an earlier version, dialog box options and the equivalent object properties for simulation and embeddable code generation targets migrate automatically to the Configuration Parameters dialog box, except in the cases that follow.

For the simulation target of a nonlibrary model, these options and properties do not migrate to the Configuration Parameters dialog box. The information is discarded when you load the model, unless otherwise noted.

Option in the Simulation Target Dialog Box of a Nonlibrary Model	Equivalent Object Property
Custom Code > Use these custom code settings for all libraries	`ApplyToAllLibs`
Description > Description	`Description` **Note** If you load an older model that contains user-specified text in the **Description** field, that text now appears in the Model Explorer. When you select **Simulink Root > Configuration Preferences** in the **Model Hierarchy** pane, the text appears in the **Description** field for that model.
Description > Document Link	`Document`

For the simulation target of a library model, these options and properties do not migrate to the Configuration Parameters dialog box. The information is discarded when you load the model.

Option in the Simulation Target Dialog Box of a Library Model	Equivalent Object Property
General > Enable debugging / animation	`CodeFlagsInfo('debug')`
General > Enable overflow detection (with debugging)	`CodeFlagsInfo('overflow')`
General > Echo expressions without semicolons	`CodeFlagsInfo('echo')`
General > Build Actions	None
Custom Code > Reserved Names	`ReservedNames`
Description > Description	`Description`
Description > Document Link	`Document`

For the embeddable code generation target of a library model, these options and properties do not migrate to the Configuration Parameters dialog box. The information is discarded when you load the model.

Option in the RTW Target Dialog Box of a Library Model	Equivalent Object Property
General > Comments in generated code	`CodeFlagsInfo('comments')`
General > Use bitsets for storing state configuration	`CodeFlagsInfo('statebitsets')`
General > Use bitsets for storing boolean data	`CodeFlagsInfo('databitsets')`
General > Compact nested if-else using logical AND/OR operators	`CodeFlagsInfo('emitlogicalops')`
General > Recognize if-elseif-else in nested if-else statements	`CodeFlagsInfo('elseifdetection')`
General > Replace constant expressions by a single constant	`CodeFlagsInfo('constantfolding')`

Option in the RTW Target Dialog Box of a Library Model	Equivalent Object Property
General > Minimize array reads using temporary variables	`CodeFlagsInfo('redundantloadelimination')`
General > Preserve symbol names	`CodeFlagsInfo('preservenames')`
General > Append symbol names with parent names	`CodeFlagsInfo('preservenameswithparent')`
General > Use chart names with no mangling	`CodeFlagsInfo('exportcharts')`
General > Build Actions	None
Custom Code > Reserved Names	ReservedNames
Description > Description	`Description`
Description > Document Link	`Document`

What Happens When You Save an Older Model in R2008b

When you use R2008b to save a model created in an earlier version, parameters for simulation and embeddable code generation from the Configuration Parameters dialog box are saved. However, properties of API Target objects `sfun` and `rtw` are not saved if those properties do not have an equivalent parameter in the Configuration Parameters dialog box. Properties that do not migrate to the Configuration Parameters dialog box are discarded when you load the model. Therefore, old Target object properties are not saved even if you choose to save the model as an older version (for example, R2007a).

Workaround for Library Models If They No Longer Use Local Custom Code Settings

Behavior in R2008a and Earlier Releases

In R2008a and earlier releases, the main model simulation target had a custom code option **Use these custom code settings for all libraries**, or the target property `ApplyToAllLibs`. The library model simulation target had a similar custom code option **Use local custom code settings (do not inherit from main model)**, or the target property `UseLocalCustomCodeSettings`.

The following criteria determined which custom code settings would apply to the library model:

If `ApplyToAllLibs` for the main model is...	And `UseLocalCustomCodeSettings` for the library model is...	Then the library model uses...
True	False	Main model custom code
True	True	Local custom code
False	True	Local custom code
False	False	Local custom code (by default, but ambiguous)

The last case was ambiguous, because the main model did not propagate custom code settings and the library model did not specify use of local custom code settings either. In this case, the default behavior was to use local custom code settings for the library model.

Behavior in R2008b

In R2008b, the **Use these custom code settings for all libraries** option for the main model is removed. The library model either picks up its local custom code settings if specified to do so, or uses

the main model custom code settings when the **Use local custom code settings** option is not selected. This change introduces backward incompatibility for older models that use the "False (main model), False (library model)" setup for specifying custom code settings.

Workaround to Prevent Backward Incompatibility

To resolve the ambiguity in older models, you must explicitly select **Use local custom code settings** for the library model when you want the local custom code settings to apply:

1   Open the Stateflow simulation target for the library model.

    a   Load the library model and unlock it.

    b   Open one of the library charts in the Stateflow Editor.

    c   Select **Tools > Open Simulation Target**.

2   In the dialog box that appears, select **Use local custom code settings (do not inherit from main model)**.

New Pattern Wizard for Consistent Creation of Logic Patterns and Iterative Loops

You can use the Stateflow Pattern Wizard to create commonly used flow graphs such as `for`-loops in a quick and consistent manner.

For more information, see Modeling Logic Patterns and Iterative Loops Using Flow Graphs.

Support for Initializing Vectors and Matrices in the Data Properties Dialog Box

In the Data properties dialog box, you can initialize vectors and matrices in the **Initial value** field of the **Value Attributes** pane.

For more information, see How to Define Vectors and Matrices.

Change in Default Mode for Ordering Parallel States and Outgoing Transitions

The default mode for ordering parallel states and outgoing transitions is now explicit. When you create a new chart, you define ordering explicitly in the Stateflow Editor. However, if you load a chart that uses implicit ordering, that mode is retained until you switch to explicit ordering.

For more information, see Execution Order for Parallel States and Evaluation Order for Outgoing Transitions.

Optimized Inlining of Code Generated for Stateflow Charts

In R2008b, Real-Time Workshop code generation is enhanced to enable optimized inlining of code generated for Stateflow charts.

More Efficient Parsing for Nonlibrary Models

When you parse a nonlibrary model, library charts that are not linked to this model are ignored. This enhancement enables more efficient parsing for nonlibrary models.

Change in Casting Behavior When Calling MATLAB Functions in a Chart

When you call MATLAB functions in a Stateflow chart, scalar inputs are no longer cast automatically to data of type `double`. This behavior applies when you use the `ml` operator to call a built-in or custom MATLAB function. (For details, see ml Namespace Operator.)

Compatibility Considerations

Previously, Stateflow generated code for simulation would automatically cast scalar inputs to data of type `double` when calling MATLAB functions in a chart. This behavior has changed. Stateflow charts created in earlier versions now generate errors during simulation if they contain calls to external MATLAB functions that expect scalar inputs of type `double`, but the inputs are of a different data type.

To prevent these errors, you can change the data type of a scalar input to `double` or add an explicit cast to type `double` in the function call. For example, you can change a function call from `ml.`*`function_name`*`(i)` to `ml.`*`function_name`*`(double(i))`.

Ability to Specify Continuous Update Method for Output Data

In R2008b, you can set the **Update Method** of output data in continuous-time charts to `Continuous`. In previous releases, only local data could use a continuous update method.

Use of Output Data with Change Detection Operators Disallowed for Initialize-Outputs-at-Wakeup Mode

If you enable the option **Initialize Outputs Every Time Chart Wakes Up** in the Chart properties dialog box, do not use output data as the first argument of a change detection operator. When this option is enabled, the change detection operator returns `false` if the first argument is an output data. In this case, there is no reason to perform change detection. (For details, see Detecting Changes in Data Values.)

Compatibility Considerations

Previously, Stateflow software would allow the use of output data with change detection operators when you enable the option **Initialize Outputs Every Time Chart Wakes Up**. This behavior has changed. Stateflow charts created in earlier versions now generate errors during parsing to prevent such behavior.

Parsing a Stateflow Chart Without Simulation No Longer Detects Unresolved Symbol Errors

To detect unresolved symbol errors in a chart, you must start simulation or update the model diagram. When you parse a chart without simulation or diagram updates, the Stateflow parser does

not have access to all the information needed to check for unresolved symbols, such as exported graphical functions from other charts and enumerated data types. Therefore, the parser now skips unresolved symbol detection to avoid generating false error messages. However, if you start simulation or update the model diagram, you invoke the model compilation process, which has full access to the information needed, and unresolved symbols are flagged.

For more information, see Parsing Stateflow Charts and How to Check for Undefined Symbols.

Generation of a Unique Name for a Copied State Limited to States Without Default Labels

If you copy and paste a state in the Stateflow Editor, a unique name is generated for the new state only if the original state does not use the default ? label. For more information, see Copying Graphical Objects.

New Configuration Set Created When Loading Nonlibrary Models with an Active Configuration Reference

When you load a nonlibrary model with an active configuration reference for Stateflow charts or Truth Table blocks, a copy of the referenced configuration set is created and attached to your model. The new configuration set is marked active, and the configuration reference is marked inactive. This behavior does not apply to library models.

For information about using configuration references, see Manage a Configuration Reference.

Compatibility Considerations

In previous releases, you could load a nonlibrary model with an active configuration reference for Stateflow charts or Truth Table blocks. In R2008b, the configuration reference becomes inactive after you load the model, and a warning message appears to explain this change in behavior. To restore the configuration reference to its original active state, follow the instructions in the warning message.

For more information, see Configuration References for Models with Older Simulation Target Settings.

R2008a+

Version: 7.1.1

Bug Fixes

R2008a

Version: 7.1

New Features

Bug Fixes

Compatibility Considerations

Support for Data with Complex Types

Stateflow charts support data with complex data types. You can perform basic arithmetic (addition, subtraction, and multiplication) and relational operations (equal and not equal) on complex data in Stateflow action language. You can also use complex input and output arguments for Embedded MATLAB functions in your chart.

For more information, see Using Complex Data in Stateflow Charts.

Support for Functions with Multiple Outputs

You can specify more than one output argument in graphical functions, truth table functions, and Embedded MATLAB functions. Previously, you could specify only one output for these types of functions.

For more information, see Graphical Functions for Reusing Logic Patterns and Iterative Loops, Truth Table Functions for Decision-Making Logic, and Using MATLAB Functions in Stateflow Charts.

Bidirectional Traceability for Navigating Between Generated Code and Stateflow Objects

In previous releases, Real-Time Workshop Embedded Coder software provided bidirectional traceability only for Simulink blocks. In R2008a, bidirectional traceability works between generated code and Stateflow objects.

For embedded real-time (ERT) based targets, you can choose to include traceability comments in the generated code. Using the enhanced traceability report, you can click hyperlinks to go from a line of code to its corresponding object in the model. You can also right-click an object in your model to find its corresponding line of code.

For more information, see Traceability of Stateflow Objects in Generated Code.

New Temporal Logic Notation for Defining Absolute Time Periods

You can use a keyword named `sec` to define absolute time periods based on simulation time of your chart. Use this keyword as an input argument for temporal logic operators, such as `after`.

For more information, see Using Temporal Logic in State Actions and Transitions.

New temporalCount Operator for Counting Occurrences of Events

You can use the `temporalCount` operator to count occurrences of explicit or implicit events. This operator can also count the seconds of simulation time that elapse during chart execution.

For more information, see Using Temporal Logic in State Actions and Transitions and Counting Events.

Using a Specific Path to a State for the in Operator

When you use the `in` operator to check state activity, you must use a specific path to a state. The operator performs a localized search for states that match the given path by looking in each level of

the Stateflow hierarchy between its parent and the chart level. The operator does not do an exhaustive search of all states in the entire chart. If there are no matches or multiple matches, a warning message appears and chart execution stops. The search algorithm must find a unique match to check for state activity.

For more information, see Checking State Activity.

Compatibility Considerations

Previously, you could use a non-specific path to a state as the argument of the `in` operator, because the operator performed an exhaustive search for all states in the chart that match the given path. In the case of multiple matches, a filtering algorithm broke the tie to produce a unique state for checking activity. This behavior has changed. Stateflow charts created in earlier versions may now generate errors if they contain an `in` operator with a non-specific path to a state.

Enhanced MISRA C Code Generation Support

Stateflow Coder software detects missing `else` statements in `if-else` structures for generated code. This enhancement supports MISRA C rule 14.10.

Enhanced Folder Structure for Generated Code

Code files for simulation and code generation targets now reside in the `slprj` folder. Previously, generated code files resided in the `sfprj` folder.

For more information, see Generated Code Files for Targets You Build.

Code Optimization for Simulink Blocks and Stateflow Charts

In R2008a, Real-Time Workshop code generation is enhanced to enable cross-product optimizations between Simulink blocks and Stateflow charts.

New fitToView Method for Zooming Objects in the Stateflow Editor

You can use the API method `fitToView` to zoom in on graphical objects in the Stateflow Editor.

For more information, see Zooming a Chart Object with the API.

Generation of a Unique Name for a Copied State

If you copy and paste a state in the Stateflow Editor, a unique name automatically appears for the new state.

For more information, see Copying Graphical Objects.

New Font Size Options in the Stateflow Editor

In the Stateflow Editor, the font sizes in the **Edit > Set Font Size** menu now include 2-point, 4-point, and 50-point. These font options are also available by right-clicking a text item and choosing **Font Size** from the context menu.

For more information, see Specifying Colors and Fonts in a Chart.

New Fixed-Point Details Display in the Data Properties Dialog Box

The Data Type Assistant in the Data properties dialog box now displays status and details of fixed-point data types.

For more information, see Showing Fixed-Point Details.

"What's This?" Context-Sensitive Help Available for Simulink Configuration Parameters Dialog

R2008a introduces "What's This?" context-sensitive help for parameters that appear in the Simulink Configuration Parameters dialog. This feature provides quick access to a detailed description of the parameters, saving you the time it would take to find the information in the Help browser.

To use the "What's This?" help, do the following:

1 Place your cursor over the label of a parameter.

2 Right-click. A **What's This?** context menu appears.

For example, the following figure shows the **What's This?** context menu appearing after a right-click on the **Start time** parameter in the **Solver** pane.

3 Click **What's This?** A context-sensitive help window appears showing a description of the parameter.

Specifying Scaling Explicitly for Fixed-Point Data

When you define a fixed-point data type in a Stateflow chart, you must specify scaling explicitly in the **General** pane of the Data properties dialog box. For example, you cannot enter an incomplete specification such as `fixdt(1,16)` in the **Type** field. If you do not specify scaling explicitly, you will see an error message when you try to simulate your model.

To ensure that the data type definition is valid for fixed-point data, perform one of these steps in the **General** pane of the Data properties dialog box:

• Use a predefined option in the **Type** drop-down menu.

• Use the Data Type Assistant to specify the **Mode** as fixed-point.

For more information, see Defining Data.

Compatibility Considerations

Previously, you could omit scaling in data type definitions for fixed-point data. Such data types were treated as integers with the specified sign and word length. This behavior has changed. Stateflow

charts created in earlier versions may now generate errors if they contain fixed-point data with no scaling specified.

Use of Data Store Memory Data in Entry Actions and Default Transitions Disallowed for Execute-at-Initialization Mode

If you enable the option **Execute (enter) Chart At Initialization** in the Chart properties dialog box, you cannot assign data store memory data in state `entry` actions and default transitions that execute the first time that the chart awakens. You can use data store memory data in state `during` actions, inner transitions, and outer transitions without any limitations.

Previously, assigning data store memory in state `entry` actions and default transitions with this option enabled would cause a segmentation violation.

Enhanced Warning Message for Target Hardware That Does Not Support the Data Type in a Chart

If your target hardware does not support the data type you use in a Stateflow chart, a warning message appears when you generate code for that chart. This message appears only if the unsupported data type is present in the chart.

Previously, a warning message appeared if the target hardware did not support a given data type, even when the unsupported data type was not actually used in the chart.

Detection of Division-By-Zero Violations When Debugger Is Off

Stateflow simulation now detects division-by-zero violations in a chart, whether or not you enable the debugger.

Previously, disabling the debugger would prevent detection of division-by-zero violations, which caused MATLAB sessions to crash.

R2007b+

Version: 7.0.1

Bug Fixes

R2007b

Version: 7.0

New Features

Bug Fixes

Compatibility Considerations

Enhanced Continuous-Time Support with Zero-Crossing Detection

Using enhanced support for modeling continuous-time systems, you can:

- Detect zero crossings on state transitions, enabling accurate simulation of dynamic systems with modal behavior.
- Support the definition of continuous state variables and their derivatives for modeling hybrid systems as state charts with embedded dynamic equations

For more information, see Modeling Continuous-Time Systems in Stateflow Charts.

Compatibility Considerations

Previously, Stateflow charts implemented continuous time simulation without maintaining mode in minor time steps or detecting zero crossings. Accurate continuous-time simulation requires several constraints on the allowable constructs in Stateflow charts. Charts created in earlier versions may generate errors if they violate these constraints.

New Super Step Feature for Modeling Asynchronous Semantics

Using a new super step property, you can enable Stateflow charts to take multiple transitions in each simulation time step. For more information, see Execution of a Chart with Super Step Semantics.

Support for Inheriting Data Properties from Simulink Signal Objects Via Explicit Resolution

You can use a new data property, **Data Must Resolve to Simulink signal object**, to allow local and output data to explicitly inherit the following properties from `Simulink.Signal` objects of the same name that you define in the base workspace or model workspace:

- Size
- Type
- Complexity
- Minimum value
- Maximum value
- Initial value
- Storage class (in Real-Time Workshop generated code)

For more information, see Resolving Data Properties from Simulink Signal Objects.

Compatibility Considerations

Stateflow software no longer performs implicit signal resolution, a feature supported for output data only. In prior releases, Stateflow software attempted to resolve outputs implicitly to inherit the size, type, complexity, and storage class of `Simulink.Signal` objects of the same name that existed in the base or model workspace. No other properties could be inherited from Simulink signals.

Now, local as well as output data can inherit additional properties from `Simulink.Signal` objects, but you must enable signal resolution explicitly. In models developed before Version 7.0 (R2007b) that

rely on implicit signal resolution, Stateflow charts may not simulate or may generate code with unexpected storage classes. In these cases, Stateflow software automatically disables implicit signal resolution for chart outputs and generates a warning at model load time about possible incompatibilities. Before loading such a model, make sure you have loaded into the base or model workspace all `Simulink.Signal` objects that will be used for explicit resolution. After loading, resave your model in Version 7.0 (R2007b) of Stateflow software.

Common Dialog Box Interface for Specifying Data Types in Stateflow Charts and Simulink Models

You can use the same dialog box interface for specifying data types in Stateflow charts and Simulink models. For more information, see Setting Data Properties in the Data Dialog Box.

Support for Animating Stateflow Charts in Simulink External Mode

When running Simulink models in external mode, you can now animate states, and view Stateflow test points in floating scopes and signal viewers. For more information, see Animating Stateflow Charts.

These Real-Time Workshop targets support Stateflow chart animation in external mode:

Real-Time Workshop Target	External Mode Support	Support for Stateflow Chart Animation in External Mode
GRT (generic real-time)	R10	Yes
VxWorks® / Tornado®	R10	Yes
RTWin (Real-Time Windows)	R11	Yes
Simulink Real-Time™	R12 *	No **
ERT (embedded real-time)	R13	Yes
RSim (rapid simulation)	R13	Yes
MPC5xx	R2007a	No
C166®	R2007a	No
TI's C6000™	R2007a	Yes
TI's C2000™	R2007b	No
Rapid Accelerator	R2007b	Yes
dSPACE® RTI	R12.1 ***	No

Note

*   Simulink Real-Time supported parameter download only from release R12 through R14sp3. As of release R2006a, Simulink Real-Time supports signal upload as well.

**   Simulink Real-Time has documented support for `xpcexplr` to display the boolean value of test point Stateflow states. You can also retrieve the state value via the Simulink Real-Time command-line API. There is no documented support for animating a Stateflow chart that is running in Simulink external mode.

*** dSPACE RTI supports parameter download only.

Support for Target Function Library

Stateflow Coder code generation software supports the Target Function Library published by Real-Time Workshop Embedded Coder software, allowing you to map a subset of built-in math functions and arithmetic operators to target-specific implementations. For more information, see Replacing Operators with Target-Specific Implementations and Replacement of C Math Library Functions with Target-Specific Implementations.

Support for Fixed-Point Parameters in Truth Table Blocks

You can now define fixed-point parameters in Truth Table blocks.

Support for Using Custom Storage Classes to Control Stateflow Data in Generated Code

You can use custom storage classes to control Stateflow local data, output data, and data store memory in Real-Time Workshop generated code.

For more information, see Custom Storage Classes in the Real-Time Workshop Embedded Coder documentation.

Loading 2007b Stateflow Charts in Earlier Versions of Simulink Software

If you save a Stateflow chart in release 2007b, you will not be able to load the corresponding model in earlier versions of Simulink software. To work around this issue, save your model in the earlier version before loading it, as follows:

1 In the Simulink model window, select **File > Save As**.
2 In the **Save as type** field, select the version in which you want to load the model.

   For example, if you want to load the model in the R2007a version of Simulink software, select **Simulink 6.6/R2007a Models (#.mdl)**.

Bug Fixed for the History Junction

In previous releases, there was a bug where a default transition action occurred more than once if you used a history junction in a state containing only a single substate. The history junction did not remember the state's last active configuration unless there was more than one substate. This bug has been fixed.

R2007a+

Version: 6.6.1

Bug Fixes

R2007a

Version: 6.6

New Features

Bug Fixes

New Operators for Detecting Changes in Data Values

You can use three new operators for detecting changes in Stateflow data values between time steps:

- `hasChanged`
- `hasChangedFrom`
- `hasChangedTo`

For more information, see Detecting Changes in Data Values.

Elimination of "goto" Statements from Generated Code

The code generation process automatically eliminates `goto` statements from generated code to produce structured, readable code that better supports MISRA C rules.

R2006b

Version: 6.5

New Features

Bug Fixes

Support for Mealy and Moore Charts

You can use a new chart property to constrain finite state machines to use either Mealy or Moore semantics. You can create Stateflow charts that implement pure Mealy or Moore semantics as a subset of Stateflow chart semantics. Mealy and Moore charts can be used in simulation and code generation of C and hardware description language (HDL). See Building Mealy and Moore Charts.

New Structure Data Type Provides Support for Buses

You can use a structure data type to interface Simulink bus signals with Stateflow charts and truth tables, and to define local and temporary structures. You specify Stateflow structure data types as `Simulink.Bus` objects. See Working with Structures and Bus Signals in Stateflow Charts.

Note Signal logging is not available for Stateflow structures.

Custom Integer Sizes

Integers are no longer restricted in size to 8, 16, or 32 bits. You can now enter word lengths of any size from one to 32 bits.

R2006a+

Version: 6.4.1

No New Features or Changes

R2006a

Version: 6.4

New Features

Option to Initialize Outputs When Chart Wakes Up

You can use a new chart option **Initialize Outputs Every Time Chart Wakes Up**. Use this to initialize the value of outputs every time a chart wakes up, not only at time 0 (see Setting Properties for a Single Chart in the online documentation). When you enable this option, outputs are reset whenever the chart is triggered, whether by a function call, edge trigger, or clock tick. The option ensures that outputs are defined in every chart execution and prevents latching of outputs.

Ability to Customize the Stateflow User Interface

You can use MATLAB code to perform the following customizations of the standard Stateflow user interface:

- Add items and submenus that execute custom commands in the Stateflow Editor
- Disable or hide menu items in the Stateflow Editor

Using the MATLAB Workspace Browser for Debugging Stateflow Charts

The MATLAB Workspace Browser is no longer available for debugging Stateflow charts. To view Stateflow data values at breakpoints during simulation, use the MATLAB command line or the Browse Data window in the Stateflow Debugger.

Chart and Truth Table Blocks Require C Compiler for 64-Bit Windows Operating Systems

No C compiler ships with Stateflow software for 64-bit Windows operating systems. Because Stateflow software performs simulation through code generation, you must supply your own MEX-supported C compiler if you wish to use Stateflow Chart and Truth Table blocks. The C compilers available at the time of this writing for 64-bit Windows operating systems include the Microsoft Platform SDK and the Microsoft Visual Studio development system.

R14SP3

Version: 6.3

New Features

Data Handling

Sharing Global Data Between Simulink Models and Stateflow Charts

This release provides an interface that gives Stateflow charts access to global variables in Simulink models. A Simulink model implements global variables as *data stores*, created either as data store memory blocks or instances of `Simulink.Signal` objects. Now Stateflow charts can share global data with Simulink models by reading and writing data store memory symbolically using the Stateflow action language. See Sharing Global Data with Multiple Charts.

Enhancements to Data Properties Dialog Box

The Stateflow data properties dialog box has been enhanced to:

- Accommodate fixed-point support
- Support parameter expressions in data properties

Stateflow charts now accept Simulink parameters or parameters defined in the MATLAB workspace for the following properties in the data properties dialog box:

- Initial Value
- Minimum
- Maximum

Entries for these parameters can be expressions that meet the following requirements:

- Expressions must evaluate to scalar values.
- For library charts, the expressions for these properties must evaluate to the same value for all instances of the library chart. Otherwise, a compile-time error appears.

See Defining Data.

Truth Table Enhancements

Using Embedded MATLAB Action Language in Truth Tables

You can now use the Embedded MATLAB action language in Stateflow truth tables. Previously, you were restricted to the Stateflow action language. The Embedded MATLAB action language offers the following advantages:

- Supports the use of control loops and conditional constructs in truth table actions
- Provides direct access to all MATLAB functions

See Truth Table Functions for Decision-Making Logic.

Embedded MATLAB Truth Table Block in Simulink Models

A truth table function block is now available as an element in the Simulink library. With this new block, you can call a truth table function directly from your Simulink model. Previously, there was a level of indirection. Your Simulink model had to include a Stateflow block that called a truth table function.

The Simulink truth table block supports the Embedded MATLAB language subset only. You must have a Stateflow software license to use the Truth Table block in Simulink models.

See Truth Table Functions for Decision-Making Logic.

API Enhancements

Retrieving Object Handles of Selected Stateflow Objects

A new function `sfgco` retrieves the object handles of the most recently selected objects in a Stateflow chart.

Default Case Handling in Generated Code

Stateflow Coder software now implements a default case in generated switch statements to account for corrupted memory at runtime. In this situation, the default case performs a recovery operation by calling the child entry functions of the state whose variable is out of bounds. Reentering the state resets the variable to a valid value.

This recovery operation is *not* performed if a Stateflow chart contains any of the following elements:

- Local events
- Machine-parented events
- Implicit events, such as state entry, state exit, and data change

If any of these conditions exist in a chart, state machine processing can become recursive, causing variables to temporarily assume values that are out of range. However, when processing finishes, the variables return to valid values.

Greater Usability

Specifying Execution Order of Parallel States Explicitly

You can specify the execution order of parallel states explicitly in Stateflow charts. Previously, the execution order of parallel states was governed solely by implicit rules, based on geometry. A disadvantage of implicit ordering is that it creates a dependency between design layout and execution priority. When you rearrange parallel states in your chart, you may inadvertently change order of execution and affect simulation results. Explicit ordering gives you more control over your designs. See Execution Order for Parallel States.

Hyperlinking Simulink Subsystems from Stateflow Events

You can now directly hyperlink the Simulink subsystem connected to a Stateflow output event by using the context menu option **Explore** for any state or transition broadcasting event. See Accessing Simulink Subsystems Triggered By Output Events.

Warnings for Transitions Looping Out of Logical Parent

A common modeling error is to create charts where a transition loops out of the logical parent of the source and destination objects. The *logical parent* is either a common parent of the source and destination objects, or if there is no common parent, the nearest common ancestor.

Consider the following example:
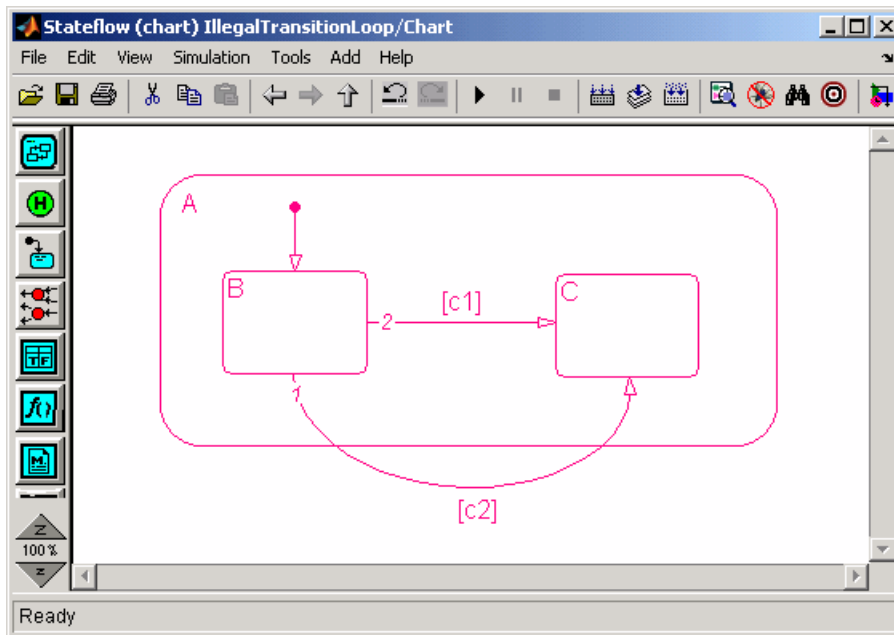
In this chart, transition 1 loops outside of logical parent A, which is the common parent of transition source B and destination C.

This type of illegal looping causes the parent to deactivate and then reactivate. In the previous example, if transition 1 is taken, the exit action of A executes and then the entry action of A executes. Executing these actions unintentionally can cause side effects.

This situation is now detected as a parser warning that indicates how to fix the model. Here is the warning associated with the earlier example:

Differentiating Syntax Elements in the Stateflow Action Language

You can now use color highlighting to differentiate syntax elements in the Stateflow action language. Syntax highlighting is enabled by default. To specify highlighting preferences, select **Highlighting Preferences** from the chart **Edit** menu, and then click the colors you want to change. See Differentiating Syntax Elements in the Stateflow Action Language.

Stateflow Chart Notes Click Function

This release introduces enhancements to Stateflow chart notes. The chart notes property dialog box now has a **ClickFcn** section, which includes the following options:

• **Use display text as click callback** check box
• **ClickFcn** edit field

See Annotations Properties Dialog Box for a description of these new options.

Chart Viewing Enhancements

This release adds the following chart viewing enhancements:

• "View Command History" on page 43-5
• "New View Menu Viewing Commands" on page 43-5
• "New Shortcut Menu Commands" on page 43-5
• "View Command Shortcut Keys" on page 43-5

View Command History

This release enhances the chart viewing commands. You can now maintain a history of the chart viewing commands, i.e., pan and zoom, that you execute for each chart window. The history allows you to quickly return to a previous view in a window, using commands for traversing the history.

New View Menu Viewing Commands

This release adds the following viewing commands to the chart's View menu:

• **View > Back**

  Displays the previous view in the view history.

• **View > Forward**

  Displays the next view in the view history.

• **View > Go To Parent**

  Goes to the parent of the current subchart.

New Shortcut Menu Commands

The shortcut menu now has **Forward** and **Go To Parent** commands. The **Back** command has been moved to be with these new commands.

View Command Shortcut Keys

This release adds the following viewing command shortcut keys for users running the UNIX operating system or the Windows operating system:

Shortcut Key	Command
d or **Ctrl+Left Arrow**	Pan left
g or **Ctrl+Right Arrow**	Pan right
e or **Ctrl+Up Arrow**	Pan up
c or **Ctrl+Down Arrow**	Pan down
b	Go back in pan/zoom history
t	Go forward in pan/zoom history

Note These shortcut keys, together with the existing zoom shortcuts (**r** or **+** for zoom in, **v** or **-** for zoom out), allow you to pan and zoom a model with one hand (your left hand).

R14SP2

Version: 6.2

New Features

Compatibility Considerations

User-Specified Transition Execution Order

Stateflow charts now support a mode where you can explicitly specify the testing or execution order of transitions sourced by states and junctions. This mode is called the explicit mode. The implicit mode retains the old functionality, where the transition execution order is determined based on a set of rules (parent depth, triggered and conditional properties, and geometry around the source). In addition, the transition numbers, according to their execution order, now appear on the Stateflow Editor at all times, in both implicit and explicit modes.

Old models created in earlier releases load in implicit mode, which produces identical simulation results. Any new charts created use implicit mode by default. To change to explicit mode, use the Chart properties dialog box.

Enhanced Integration of Stateflow Library Charts with Simulink Models

Charts in library models do not require full specification of data type and size. During simulation, library charts can inherit data properties from the main model in which you link them.

This enhancement also affects code generation in library charts. When building simulation and code generation targets, only the library charts that you link in the main model participate in code generation.

Compatibility Considerations

In previous releases, library charts required complete specification of data properties. You had to enter these properties for both the library chart and the main model before simulation.

Stateflow Charts and Embedded MATLAB Functions Support Simulink Data Type Aliases

Data in Stateflow charts and Embedded MATLAB functions may now be explicitly typed using the same aliased types that a Simulink model uses. Also, inherited and parameterized data types in Stateflow charts and Embedded MATLAB functions support propagation of aliased types. However, code generated for Stateflow charts and Embedded MATLAB functions does not yet preserve aliased data types.

Fixed-Point Override Supported for Library Charts

You can now specify fixed-point override for Stateflow library charts.