

# Simulink<sup>®</sup> Release Notes

---

The Chapter 1, “Simulink 6.3 Release Notes” describe changes since Simulink 6.2. It includes the following sections:

- “New Features” on page 1-2
- “Model Referencing Limitations” on page 1-10
- “Major Bug Fixes” on page 1-13
- “Known Software and Documentation Problems” on page 1-14
- “Upgrading from a Previous Release” on page 1-15

The Simulink Release Notes also provide information about recent versions of the product, in case you are upgrading from a version that was released prior to Simulink 6.2.

- Chapter 2, “Simulink 6.2 Release Notes”
- Chapter 3, “Simulink 6.1 Release Notes”
- Chapter 4, “Simulink 6.0 Release Notes”
- Chapter 5, “Simulink 5.1 Release Notes”
- Chapter 6, “Simulink 5.0.1 Release Notes”
- Chapter 7, “Simulink 5.0 Release Notes”
- Chapter 8, “Simulink 4.1 Release Notes”
- Chapter 9, “Simulink 4.0 Release Notes”

**Printing the Release Notes.** If you would like to print the Release Notes, you can link to a PDF version.



## Simulink 6.3 Release Notes

1

<b>New Features</b> .....	<b>1-2</b>
Model Reference Enhancements .....	1-2
Block Enhancements .....	1-3
Modeling Enhancements .....	1-5
Simulation Enhancements .....	1-7
User Interface Enhancements .....	1-8
<b>Model Referencing Limitations</b> .....	<b>1-10</b>
Referencing and Referenced Model Limitations .....	1-10
Referencing Model Limitations .....	1-10
Referenced Model Limitations .....	1-11
<b>Major Bug Fixes</b> .....	<b>1-13</b>
<b>Known Software and Documentation Problems</b> .....	<b>1-14</b>
<b>Upgrading from a Previous Release</b> .....	<b>1-15</b>
New File Extension for MEX-Files on Windows .....	1-15
Fixed-Point Functions No Longer Supported for Use in Signal Objects .....	1-15
Parameter Object Expressions No Longer Supported in Dialog Boxes .....	1-15

## Simulink 6.2 Release Notes

2

<b>New Features</b> .....	<b>2-2</b>
<b>Model Referencing Limitations</b> .....	<b>2-3</b>
Referencing and Referenced Model Limitations .....	2-3

Referencing Model Limitations .....	2-3
Referenced Model Limitations .....	2-4
<b>Upgrading from a Previous Release .....</b>	<b>2-6</b>

## Simulink 6.1 Release Notes

### 3

<b>Changes from the Previous Release .....</b>	<b>3-2</b>
Changed Source Dialog Box Behavior .....	3-2
Changed Model Explorer Source Behavior .....	3-2
Affected Blocks .....	3-3
<b>Upgrading from a Previous Release .....</b>	<b>3-5</b>
Model Load Warnings .....	3-5
<b>Major Bug Fixes .....</b>	<b>3-6</b>
Wrong Discrete-Time Integrator Reset Behavior .....	3-6

## Simulink 6.0 Release Notes

### 4

<b>New Features .....</b>	<b>4-2</b>
Model Explorer .....	4-2
Configuration Sets .....	4-2
Model Referencing .....	4-3
Model Workspaces .....	4-4
Implicit Fixed-Step Solver .....	4-4
The Signal and Scope Manager .....	4-4
Data Object Type Enhancements .....	4-4
Block Enhancements .....	4-5
Signal Enhancements .....	4-9
Rate Transition Enhancements .....	4-9
Execution Context Enhancements .....	4-10
Algebraic Loop Minimization .....	4-10
Level-2 M-File S-Functions .....	4-10

Panning Model Diagrams .....	4-11
<b>Model Referencing Limitations</b> .....	<b>4-12</b>
Referencing and Referenced Model Limitations .....	4-12
Referencing Model Limitations .....	4-12
Referenced Model Limitations .....	4-13
<b>Major Bug Fixes</b> .....	<b>4-15</b>
<b>Upgrading from an Earlier Release</b> .....	<b>4-16</b>
Changes in MATLAB Data Type Conversions .....	4-16
Signal Object Resolution Changes .....	4-16
Loading Models Containing Non-ASCII Characters .....	4-17
Change in Sample Time Behavior of Unary Minus Block ..	4-17
Initial Output of Conditionally Executed Subsystems ....	4-17
Execution Context Default Changes .....	4-18

## Simulink 5.1 Release Notes

# 5

<b>New Features</b> .....	<b>5-2</b>
Sample Time Parameters Exposed .....	5-2
Enhanced Debugger .....	5-2
Context-Sensitive Data Typing of Tunable Parameters ...	5-5
Conditional Execution Behavior .....	5-7
Function-Call Subsystem Enhancements .....	5-10
External Increment Option Added To For Iterator Block ..	5-10
<b>Performance Improvements</b> .....	<b>5-11</b>
<b>Major Bug Fixes</b> .....	<b>5-13</b>
<b>Upgrading from an Earlier Release</b> .....	<b>5-14</b>

## Simulink 5.0.1 Release Notes

### 6

<b>Major Bug Fixes</b> .....	<b>6-2</b>
<b>Upgrading from an Earlier Release</b> .....	<b>6-3</b>
Backwards Compatibility of Tunable Parameters for Unified Fixed-Point Blocks .....	<b>6-3</b>

## Simulink 5.0 Release Notes

### 7

<b>New Features</b> .....	<b>7-2</b>
Block Enhancements .....	<b>7-2</b>
Simulation Enhancements .....	<b>7-6</b>
Modeling Enhancements .....	<b>7-8</b>
<b>Major Bug Fixes</b> .....	<b>7-11</b>
<b>Platform Limitations for HP and IBM</b> .....	<b>7-12</b>
<b>Upgrading from an Earlier Release</b> .....	<b>7-14</b>
BlockInstanceData Function Deprecated .....	<b>7-14</b>

## Simulink 4.1 Release Notes

### 8

<b>New Features</b> .....	<b>8-2</b>
Simulink Editor .....	<b>8-2</b>
Modeling Enhancements .....	<b>8-3</b>
Simulink Debugger .....	<b>8-7</b>
Block Library .....	<b>8-8</b>
<b>Bug Fixes</b> .....	<b>8-11</b>

<b>Upgrading from an Earlier Release</b> .....	<b>8-13</b>
Running Simulink 4.1 Models in Simulink 4.0 .....	<b>8-13</b>
Simulink Block Library Reorganization .....	<b>8-13</b>
Direct Feedthrough Compensation Deprecated .....	<b>8-13</b>
S-Functions Sorted Like Built-In Blocks .....	<b>8-14</b>
Added Latched Triggered Subsystems .....	<b>8-14</b>
Self-Triggering Subsystems Are No Longer Allowed .....	<b>8-15</b>
Improved Invalid Model Configuration Diagnostics .....	<b>8-15</b>

## Simulink 4.0 Release Notes

# 9

<b>New Features</b> .....	<b>9-2</b>
Simulink Editor .....	<b>9-2</b>
Modeling Enhancements .....	<b>9-5</b>
Simulink Debugger .....	<b>9-6</b>
Block Library .....	<b>9-6</b>
SB2SL .....	<b>9-9</b>
<b>Upgrading from an Earlier Release</b> .....	<b>9-11</b>
Port Name Property .....	<b>9-11</b>





# Simulink 6.3 Release Notes

---

New Features (p. 1-2)	Simulink features introduced in this release.
Model Referencing Limitations (p. 1-10)	Limitations on referencing models in this release
Major Bug Fixes (p. 1-13)	How to get information on important bugs that this release fixes.
Known Software and Documentation Problems (p. 1-14)	How to get information on software and documentation problems known to exist in this release.
Upgrading from a Previous Release (p. 1-15)	Changes in this release that may affect models developed in previous releases.

## New Features

### Model Reference Enhancements

#### Function-Call Models

This release allows you to use a block capable of emitting a function-call signal, such as a Function-Call Generator or a custom S-function, in one model to control execution of another model during the current time step. See “Function-Call Models” for more information.

#### Using Noninlined S-Functions in Referenced Models

This release adds limited support for use of noninlined S-functions in models referenced by other models. For example, you can simulate a model that references models containing noninlined S-functions. However, you cannot use the Real-Time Workshop to generate a standalone executable (Real-Time Workshop target) for the model. See “Referenced Model Limitations” on page 1-11 for information on other limitations.

#### Referenced Models Without Root I/O Can Inherit Sample Times

Previous releases of Simulink do not allow referenced models without root-level input or output ports to inherit their sample time. This release removes this restriction.

#### Referenced Models Can Use Variable Step Solvers

Previous releases of Simulink do not allow models to reference models that require variable-step solvers. This release removes this restriction.

#### Model Dependency Graphs Accessible from the Tools Menu

This release adds a **Model Reference Dependency Graph** item to the Model Editor’s **Tools** menu. The item displays a graph of the models referenced by the model displayed in the Model Editor. You can open any model in the dependency graph by clicking its node. See “Browsing Model Reference Dependencies” for more information.

## Command That Converts Atomic Subsystems to Model References

This release introduces a MATLAB command that converts an atomic subsystem to a model reference. See `Simulink.SubSystem.convertToModelReference` for more information.

## Model Reference Demos

This release has the following model reference demo changes:

- Model reference demo names are now prepended with `sldemo_`. For example, the demo `mdlref_basic.mdl` is now `sldemo_mdlref_basic.mdl`.
- You can no longer use the `mdlrefdemos` command from the MATLAB command prompt to access model reference demos. Instead, you can navigate to the Simulink demos tab either through the Help browser, or by typing `demos` at the command prompt, then navigating to the Simulink demos category and browsing the demos.

## Block Enhancements

### Variable Transport Delay, Variable Time Delay Blocks

This release replaces the Variable Transport Delay block of previous releases with two new blocks. The Variable Transport Delay block of previous releases implemented a variable time delay behavior, which is now implemented by the Variable Time Delay block introduced in this release. This release changes the behavior of the Variable Transport Delay block to model variable transport delay behavior, e.g., the behavior of a fluid flowing through a pipe.

### Additional Reset Trigger for Discrete-Time Integrator Block

This release adds a `sampled_level` trigger option for causing the Discrete-Time Integrator to reset. The new reset trigger is more efficient than the `level` reset option, but may introduce a discontinuity when integration resumes.

---

**Note** In Simulink 6.2 and 6.2.1, the level reset option behaves like the sampled level option in this release. This release restores the level reset option to its original behavior.

---

## Input Port Latching Enhancements

This release includes the following enhancements to the signal latching capabilities of the Inport block.

**Label Clarified for Triggered Subsystem Latch Option.** The dialog box for an Inport block contains a check box to latch the signal connected to the system via the port. This check box applies only to triggered subsystems and hence is enabled only when the Inport block resides in a triggered subsystem. In this release, the label for the check box that selects this option has changed from **Latch (buffer) input** to **Latch input by delaying outside signal**. This change is intended to make it clear what the option does, i.e., cause the subsystem to see the input signal's value at the previous time step when the subsystem executes at the current time step (equivalent to inserting a Memory block at the input outside the subsystem). The Inport block's icon displays <Lo> to indicate that this option is selected.

**Latch Option Added for Function-Call Subsystems.** This release adds a check box labeled **Latch input by copying inside signal** to the Inport block's dialog box. This option applies only to function-call subsystems and hence is enabled only if the Inport block resides in a function-call subsystem. Selecting this option causes Simulink to copy the signal output by the block into a buffer before executing the contents of the subsystem and to use this copy as the block's output during execution of the subsystem. This ensures that the subsystem's inputs, including those generated by the subsystem's context, will not change during execution of the subsystem. The Inport block's icon displays <Li> to indicate that this option is selected.

## Improved Function-Call Inputs Warning Label

In previous releases, the dialog box for a function-call subsystem contains a check box labeled **Warn if function-call inputs arise inside called context**. This release changes the label to **Warn if function-call inputs are context-specific**. This change is intended to indicate more clearly the

warning's purpose, i.e., to alert you that some or all of the function-call inputs come from the function-call subsystem's context and hence could change while the function-call subsystem is executing.

---

**Note** In this release, you can avoid this function-call inputs problem by selecting the **Latch input by copying inside signal** option on the subsystem's Inport blocks (see “Latch Option Added for Function-Call Subsystems” on page 1-4).

---

## Modeling Enhancements

### Annotations

This release introduces the following enhancements to model annotations:

- Annotation properties dialog box (see “Annotations Properties Dialog Box”)
- Annotation callback functions (see “Annotation Callback Functions”)
- Annotation application programming interface (see “Annotations API”)

### Custom Signal Viewers and Generators

This release allows you to add custom signal viewers and generators so that you can manage them in the Signal & Scope Manager. See “Adding Custom Viewers and Generators to the Signal & Scope Manager” for further details.

### Model Explorer Search Option

This release adds an Evaluate Property Values During Search option to the Model Explorer. This option applies only for searches by property value. If enabled, the option causes the Model Explorer to evaluate the value of each property as a MATLAB expression and compare the result to the search value. If disabled (the default), the Model Explorer compares the unevaluated property value to the search value.

### Using Signal Objects to Assign Signal Properties

Previous releases allow you to use signal objects to check signal property values assigned by signal sources. This release allows you, in addition, to use

signal objects to assign values to properties not set by signal sources. See `Simulink.Signal` for more information.

### **Bus Utility Functions**

This release introduces the following bus utility functions:

- `Simulink.Bus.save`
- `Simulink.Bus.createObject`
- `Simulink.Bus.cellToObject`

### **Fixed-Point Support in Embedded MATLAB**

In this release, the Embedded MATLAB Function block supports many Fixed-Point Toolbox functions. This allows you to generate code from models that contain fixed-point M functions. See “Using the Fixed-Point Toolbox with Embedded MATLAB” in the Fixed-Point Toolbox documentation for more information.

---

**Note** You must have a Simulink Fixed Point license to use this capability.

---

### **Embedded MATLAB Function Editor**

The Embedded MATLAB Editor has a new tool, the Ports and Data Manager. This tool helps you manage your block inputs, outputs, and parameters. The Ports and Data Manager uses the same Model Explorer dialogs for manipulating data, but restricts the view to the block you are working on. You can still access the Model Explorer via a menu item to get the same functionality as in previous releases.

### **Input Trigger and Function-call Output Support in Embedded MATLAB**

Embedded MATLAB now supports input triggers and function-call outputs. See “Ports and Data Manager” for more information.

## Find Options Added to Data Object Wizard

This release adds find options to the **Data Object Wizard**. The options enable you to restrict the search for model data to specific kinds of objects. See “Data Object Wizard” for more information.

## Simulation Enhancements

### Viewing Logged Signal Data

This release can display logged signal data in the MATLAB **Times Series Tools** viewer on demand or whenever a simulation ends or you pause a simulation. See “Viewing Logged Signal Data” in the Simulink documentation for more information.

### Importing Time-Series Data

In this release, root-level Inport blocks can import data from time-series (see `Simulink.Timeseries`) and time-series array (see `Simulink.TSArray`) objects residing in the MATLAB workspace. See “Importing Data from the MATLAB Workspace” for more information. From Workspace blocks can also import time-series objects. The ability to import time-series objects allows you to use data logged from one simulation as input to another simulation.

### Using a Variable-Step Solver with Rate Transition Blocks

Previous releases of Simulink generate an error if you try to use a variable-step solver to solve a model that contains Rate Transition blocks. This release allows you to use variable-step as well as fixed-step solvers to simulate a model. Note that you cannot generate code from a model that uses a variable-step solver. However, you may find it advantageous, in some cases, to use a variable-step solver to test aspects of the model not directly related to code generation. This enhancement allows you to switch back and forth between the two types of solver without having to remove and reinsert Rate Transition blocks.

### Additional Diagnostics

This releases adds the following simulation diagnostics:

- “Enforce sample times specified by Signal Specification blocks”

- “Extraneous discrete derivative signals”
- “Detect read before write”
- “Detect write after read”
- “Detect write after write”

### **Data Integrity Diagnostics Pane Renamed, Reorganized**

This release changes the name of the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box to the **Data Validity** pane. It also reorganizes the pane into groups of related diagnostics. See “Data Validity Diagnostics” for more information.

### **Improved Sample-Time Independence Error Messages**

When you enable the Ensure sample time independent solver constraint (see “Periodic sample time constraint” for more information), Simulink generates several error messages if the model is not sample-time independent. In previous releases, these messages were not specific enough for you to determine why a model failed to be sample-time independent. In this release, the messages point to the specific block, signal object, or model parameter that causes the model not to be sample-time independent.

## **User Interface Enhancements**

### **Model Viewing**

This release adds the following model viewing enhancements:

- A command history for pan and zoom commands (see “View Command History”)
- Keyboard shortcuts for panning model views (see “Model Viewing Shortcuts”)

### **Customizing the Simulink User Interface**

This release allows you to use M-code to perform the following customizations of the standard Simulink user interface:



- Add custom commands to the Model Editor's **Tools** menu (see “Adding Custom Commands to the Tools Menu”)
- Disable, or hide widgets on Simulink dialog boxes (see “Disabling and Hiding Dialog Box Controls”)

## Model Referencing Limitations

This release imposes some limitations on the use of model referencing. For example, in this release, models must meet certain conditions to reference other models or be referenced by other models. This release also restricts the use of model referencing with some features of Simulink and products based on Simulink. This section summarizes some of the major limitations that this release places on model referencing. See Simulink-related product information for additional information on the limitations summarized here.

### Referencing and Referenced Model Limitations

The following limitations apply both to models that reference other models (referencing models) and models referenced by other models.

- The simulation start time of both referencing and referenced models must be 0.
- The model's inline parameters optimization must be selected. If the optimization is off for the top model, this release displays an error message. If it is off for referenced models, this release turns it on while generating model reference targets.
- The model must use `Simulink.Parameter` objects to specify the tunability of model parameters.

This release ignores tunable parameter information specified by the model's **Model Parameter Configuration** dialog box. This release provides a utility function, `tunablevars2parameterobjects`, to facilitate conversion of parameter tunability information from dialog to `Simulink.Parameter` object form.

### Referencing Model Limitations

A model can reference other models if it meets the following conditions:

- The Model Browser does not display Model blocks in its tree view.  
Use the Model Explorer to browse models referenced by a model.
- Tools that require access to a model's internal data or configuration, including the Model Coverage Tool, Report Generator, the Simulink and Stateflow debuggers, and the Profiler, do not work when invoked from a

top model on models that the top model references. This is because the referenced models appear as black boxes to the top model. For example, you cannot use the Simulink Debugger to step from a top model into a referenced model.

- You cannot print a referenced model from a top model.
- If you want to initialize the states of a model that references other models with states, you must specify the initial states in structure format.
- To Workspace and Scope blocks in models referenced by a top model do not log data when you simulate or run code generated from a top model.
- A continuous sample time cannot be propagated to a Model block that is sample-time independent.
- Linearization is not supported for models containing Model blocks.
- Right clicking on a subsystem to build an S-function from it works for subsystems containing Model blocks only if the model is configured to use `ert.tlc`.

## Referenced Model Limitations

A model can be referenced by other models if it meets the following conditions:

- The model must specify that it can be referenced (see “Total number of instances allowed per top model”).
- This release places some restrictions on the sample times that a referenced model can inherit from the model that references it. See “Model Block Sample Times” in the Simulink documentation for more information.
- This release places some limitations on I/O connections in referenced models.
- A referenced model must use `Simulink.Bus` objects to specify any buses that it inputs and outputs (see “Bus I/O Limitations”).
- A referenced model can input or output only those user-defined data types that are fixed-point or defined by `Simulink.DataType` or `Simulink.Bus` objects.
- Function-Call and Goto/From blocks cannot cross model reference boundaries.

- The Real-Time Workshop cannot generate standalone executable (Real-Time Workshop targets) for models that reference models that include noninlined S-functions.
- A referenced model cannot use noninlined S-functions in the following cases:
  - The model uses a variable-step solver.
  - The model is referenced more than once in the model reference hierarchy. The workaround in this case is to make copies of the referenced model, assign different names to the copies, and reference the copies in the model reference hierarchy.
  - The S-function was generate by the Real-Time Workshop.
  - The S-function supports use of fixed-point numbers as inputs, outputs, or parameters.
- This release ignores custom code settings in the **Configuration Parameter** dialog box and custom code blocks when generating the simulation target for a referenced model.
- This release does not include Stateflow target custom code in simulation targets generated for referenced models.
- This release does not support referenced models that have asynchronous rates.
- A referenced model can input or output indices. However, Simulink may not be able to detect a 0-based index that is connected to a model port expecting or outputting a 1-based index, or vice versa. See “Index I/O Limitations” in the Simulink documentation for more information.
- Model blocks referencing models that contain assignment blocks that are not in an iterator subsystem, cannot be placed in an iterator subsystems.
- The Real-Time Workshop’s S-function target and grt\_malloc-based target do not support model reference.
- When generating a simulation target for a referenced model that contains an S-function with a TLC file, Simulink inlines the S-function only if the S-function sets the SS\_OPTION\_USE\_TLC\_WITH\_ACCELERATOR flag.
- This release places some restrictions on the use of custom storage classes in referenced models. See The Real-Time Workshop documentation for details.

## Major Bug Fixes

---

**Note** If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

---

To view major bug fixes made in R14SP3 and R14SP2. for Simulink, use the Bug Reports interface on the MathWorks Web site.

---

**Note** If you are not already logged in to Access Login, when you link to the Bug Reports interface (see below), you will be prompted to log in or create an Access Login account.

---

After you are logged in, use the link, Bug Reports, to view the bug reports for Simulink. The reports are sorted with fixed bugs listed first, and then open bugs.

For descriptions of bugs fixed before R14SP2, see “Major Bug Fixes” on page 3-6 in the Simulink 6.1 release notes.

## **Known Software and Documentation Problems**

See Bug Reports on the MathWorks Web site for descriptions of open bugs in R14SP3.

## Upgrading from a Previous Release

### **New File Extension for MEX-Files on Windows**

In this release, the extension for files created by the MATLAB `mex` command on Windows has changed from `dll` to `mexw32` or `mexw64`. If you have implemented any S-functions in C, Ada, or Fortran or have models that reference other models, you should

- recreate any `mexopts.bat` files (other than the one in your MATLAB preferences directory) that you use to build S-functions and model reference simulation targets
- rebuild your S-functions

### **Fixed-Point Functions No Longer Supported for Use in Signal Objects**

Previous releases allowed you to use fixed-point data type functions, such as `sfix`, to specify the value of the `DataType` property of a `Simulink.Signal` object. This release allows you to use only builtin data types and `Simulink.NumericType` objects to specify the data types of `Simulink.Signal` objects. See the `Simulink.Signal` documentation for more information.

### **Parameter Object Expressions No Longer Supported in Dialog Boxes**

Previous releases allow you to specify a `Simulink.Parameter` object as the value of a block parameter by entering an expression that returns a parameter object in the parameter's value field in the block's parameter dialog box. In this release, you must enter the name of a variable that references the object in the MATLAB or model workspace.





# Simulink 6.2 Release Notes

---

## New Features

This release introduces the following enhancements to the Signal & Scope Manager's Simulink Scope Viewer:

- Viewers can now display multiple signals on a single set of axes.
- Viewers can now log the signals that they display to the MATLAB workspace.
- Viewers can now display a legend that identifies signal traces.
- Viewers can now display tic labels both inside and outside scope axes.
- You can open a viewer's parameters dialog box by right-clicking on the viewer scope.

See the Simulink 6.2 documentation for more information on these enhancements.

## Model Referencing Limitations

This release imposes some limitations on the use of model referencing. For example, in this release, models must meet certain conditions to reference other models or be referenced by other models. This release also restricts the use of model referencing with some features of Simulink and products based on Simulink. This section summarizes some of the major limitations that this release places on model referencing. See Simulink-related product information for additional information on the limitations summarized here.

### Referencing and Referenced Model Limitations

The following limitations apply both to models that reference other models (referencing models) and models referenced by other models.

- The simulation start time of both referencing and referenced models must be 0.
- The model's inline parameters optimization must be selected. If the optimization is off for the top model, this release displays an error message. If it is off for referenced models, this release turns it on while generating model reference targets.
- The model must use `Simulink.Parameter` objects to specify the tunability of model parameters.

This release ignores tunable parameter information specified by the model's **Model Parameter Configuration** dialog box. This release provides a utility function, `tunablevars2parameterobjects`, to facilitate conversion of parameter tunability information from dialog to `Simulink.Parameter` object form.

### Referencing Model Limitations

A model can reference other models if it meets the following conditions:

- The Model Browser does not display Model blocks in its tree view.  
Use the Model Explorer to browse models referenced by a model.
- Tools that require access to a model's internal data or configuration, including the Model Coverage Tool, Report Generator, the Simulink and Stateflow debuggers, and the Profiler, do not work when invoked from a

top model on models that the top model references. This is because the referenced models appear as black boxes to the top model. For example, you cannot use the Simulink Debugger to step from a top model into a referenced model.

- You cannot print a referenced model from a top model.
- If you want to initialize the states of a model that references other models with states, you must specify the initial states in structure format.
- To Workspace and Scope blocks in models referenced by a top model do not log data when you simulate or run code generated from a top model.
- A continuous sample time cannot be propagated to a Model block that is sample-time independent.
- Linearization is not supported for models containing Model blocks.
- Right clicking on a subsystem to build an S-function from it works for subsystems containing Model blocks only if the model is configured to use `ert.tlc`.

## Referenced Model Limitations

A model can be referenced by other models if it meets the following conditions:

- The model must use a fixed-step solver (see "Choosing a Fixed-Step Solver").
- The model must specify that it can be referenced (see "Total number of instances allowed per top model").
- This release places some restrictions on the sample times that a referenced model can inherit from the model that references it. See "Model Block Sample Times" in the Simulink documentation for more information.
- This release places some limitations on I/O connections in referenced models.
- A referenced model must use `Simulink.Bus` objects to specify any buses that it inputs and outputs (see "Bus I/O Limitations").
- A referenced model can input or output only those user-defined data types that are fixed-point or defined by `Simulink.DataType` or `Simulink.Bus` objects.

- Function-Call and Goto/From blocks cannot cross model reference boundaries.
- A referenced model cannot contain noninlined S-functions.
- This release ignores custom code settings in the **Configuration Parameter** dialog box and custom code blocks when generating the simulation target for a referenced model.
- This release does not include Stateflow target custom code in simulation targets generated for referenced models.
- This release does not support referenced models that have asynchronous rates.
- A referenced model can input or output indices. However, Simulink may not be able to detect a 0-based index that is connected to a model port expecting or outputting a 1-based index, or vice versa. See "Index I/O Limitations" in the Simulink documentation for more information.
- Model blocks referencing models that contain assignment blocks that are not in an iterator subsystem, cannot be placed in an iterator subsystems.
- The Real-Time Workshop's S-function target and grt\_malloc-based target do not support model reference.
- When generating a simulation target for a referenced model that contains an S-function with a TLC file, Simulink inlines the S-function only if the S-function sets the SS\_OPTION\_USE\_TLC\_WITH\_ACCELERATOR flag.
- This release places some restrictions on the use of custom storage classes in referenced models. See The Real-Time Workshop documentation for details.

## Upgrading from a Previous Release

If you save a model with rootlevel input ports in this release and load it in a previous release, you will get the following warning:

```
Warning: model, line xxx block_diagram does not have a parameter  
named 'SignalName'.
```

You can safely ignore this warning.

# Simulink 6.1 Release Notes

---

## Changes from the Previous Release

In this release, Simulink no longer provides the user with the ability to change the values of source block parameters through either a dialog box or the Model Explorer while a simulation is running.

*\*Changes described in this section reflect Simulink reprogramming implemented to comply with a court decision regarding patent litigation.*

### Changed Source Dialog Box Behavior

In this release, opening the dialog box of a source block with tunable parameters causes a running simulation to pause. While the simulation is paused, you can edit the parameter values displayed on the dialog box. However, you must close the dialog box to have the changes take effect and allow the simulation to continue. Similarly, starting a simulation causes any open dialog boxes associated with source blocks with tunable parameters to close.

Since you can no longer change source block parameters while a simulation is running, this release removes the **Apply** button from the dialog boxes of source blocks.

---

**Note** In this release, as in previous releases, if you enable the **Inline parameters** option, Simulink does not pause the simulation when you open a source block's dialog box because all of the parameter fields are disabled and can be viewed but cannot be changed.

---

### Changed Model Explorer Source Behavior

In this release, the parameter fields in both the list view and the dialog pane of the Model Explorer have been disabled and the **Apply** button has been removed for source blocks with tunable parameters while a simulation is running. As a result, you can no longer use the Model Explorer to change source block parameters while a simulation is running.



## Affected Blocks

Blocks affected are all source blocks with tunable parameters, including the following blocks.

- Simulink source blocks, including
  - Band-Limited White Noise
  - Chirp Signal
  - Constant
  - Pulse Generator
  - Ramp
  - Random Number
  - Repeating Sequence
  - Signal Generator
  - Sine Wave
  - Step
  - Uniform Random Number
- User-developed masked subsystem blocks that have one or more tunable parameters and one or more output ports, but no input ports.
- S-Function and M-file (level 2) S-Function blocks that have one or more tunable parameters and one or more output ports but no input ports.
- Source blocks in other MathWorks products, including:
  - CDMA Reference Blockset
  - Communications Blockset
  - Embedded Target for TI C6000 DSP
  - Signal Processing Blockset (formerly DSP Blockset)
  - Simulink Fixed Point (formerly Fixed-Point Blockset)
  - System Identification Toolbox
  - xPC Target
  - xPC TargetBox

See the release notes for each product for a list of that product's source blocks affected by the changes in this release.

## Upgrading from a Previous Release

This section discusses problems that may occur when upgrading to this release from previous releases of Simulink.

### **Model Load Warnings**

If you open a model in Simulink 6.0 that was created or saved with Simulink 6.1, Simulink 6.0 displays warnings that the following parameters are undefined:

- `StrictBusMsg`
- `MdlSubVersion`

Depending on the model, Simulink 6.0 may also display a warning that the parameter `BusObject` is not defined. You can safely ignore these warnings.

## Major Bug Fixes

---

**Note** If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

---

Simulink 6.1 includes the following bug fixes made since Version 6.0. If you are upgrading from a version of Simulink earlier than Version 6.0, you should also see the Major Bug Fixes summary for Simulink 6.0. The following bug fix occurred in Version 6.0 but was not documented in the Version 6.0 release notes.

### **Wrong Discrete-Time Integrator Reset Behavior**

In Simulink 5.0 (Release 13) and 5.1 (Release 13SP1), if the Discrete-Time Integrator block's **External reset** option is set to level1, the block incorrectly resets itself not only when the reset signal is not zero, but also when the signal is falling or rising. In Simulink 6.0 (Release 14) and subsequent releases, the integrator correctly resets itself when the reset signal is not zero.

# Simulink 6.0 Release Notes

---

## New Features

This section summarizes the changes and enhancements introduced in Simulink 6.0. The features and enhancements include:

- “Model Explorer” on page 4-2
- “Configuration Sets” on page 4-2
- “Model Referencing” on page 4-3
- “Model Workspaces” on page 4-4
- “Implicit Fixed-Step Solver” on page 4-4
- “The Signal and Scope Manager” on page 4-4
- “Data Object Type Enhancements” on page 4-4
- “Block Enhancements” on page 4-5
- “Signal Enhancements” on page 4-9
- “Rate Transition Enhancements” on page 4-9
- “Execution Context Enhancements” on page 4-10
- “Algebraic Loop Minimization” on page 4-10
- “Level-2 M-File S-Functions” on page 4-10
- “Panning Model Diagrams” on page 4-11

### Model Explorer

The Model Explorer is a new tool that lets you quickly navigate, view, create, configure, search, and modify all data and properties of a Simulink model or Stateflow chart. See "The Model Explorer" in the online Simulink help for more information.

### Configuration Sets

This release introduces configuration sets. A configuration set is a named set of values for simulation parameters, such as solver type and simulation start or stop time. Every new model is created with a configuration set that is initialized from a global default configuration set. You can create additional configuration sets for a given model and designate any of them as the active

set with the click of a mouse button. See "Configuration Sets" in the online Simulink documentation for more information.

### **Configuration Parameters Dialog Box**

This release replaces the **Simulation Parameters** dialog box with the **Configuration Parameters** dialog box. The **Configuration Parameters** dialog box allows you to set a model's active configuration parameters. You can also use the Model Explorer to set the active configuration parameters as well as inactive parameters. See "The Configuration Parameters Dialog Box" for more information.

### **Model Referencing**

This release introduces model referencing, a feature that lets a model include other models as modular components. You include other models in a model by using Model blocks to reference the included models. Like subsystems, model referencing allows you to organize large models hierarchically, with Model blocks representing major subsystems. However, model referencing has significant advantages over subsystems in many applications. The advantages include:

- Modular development

You can develop the referenced model independently from the models in which it is used.

- Inclusion by reference

You can reference a model multiple times in another model without having to make redundant copies. Multiple models can also reference the same model.

- Incremental loading

The referenced model is not loaded until it is needed, speeding up model loading.

- Incremental code generation

Simulink and the Real-Time Workshop create binaries to be used in simulations and standalone applications to compute the outputs of the included blocks. Code generation occurs only for models that have changed.

See "Referencing Models" in the online Simulink documentation for more information. For a demonstration of a way to automate conversion of an existing model's subsystems to model references, execute `mdlref_conversion` at the MATLAB Command Line. For a summary of limitations on the use of model referencing in this release, see "Model Referencing Limitations" on page 4-12.

### **Model Workspaces**

In this release, Simulink provides each model with its own workspace for storing data. Models can access data in their own workspaces as well as data in models that reference them and in the base (i.e., MATLAB) workspace. Model workspaces allow you to create data for one model without fear of inadvertently altering another model's data. See "Working with Model Workspaces" for more information.

### **Implicit Fixed-Step Solver**

This release includes a new fixed-step solver named `ode14x`. This is an implicit, extrapolating fixed-step solver whose extrapolation order and number of Newton's method iterations can be specified via Simulink configuration parameters. The `ode14x` solver is faster than Simulink's explicit fixed-step solvers for certain types of stiff systems that require a very small step size to avoid unstable solutions.

### **The Signal and Scope Manager**

The Signal and Scope Manager is a new Simulink feature that enables you to globally manage signal generators and viewers. See "The Signal & Scope Manager" in the online Simulink help for more information.

### **Data Object Type Enhancements**

This release introduces the following types of objects for specifying the properties of model signals and parameters (i.e., model data):



<b>Object Class</b>	<b>Purpose</b>
<code>Simulink.AliasType</code>	Specify another name for a data type.
<code>Simulink.NumericType</code>	Define a custom data type.
<code>Simulink.StructType</code>	Define a data structure, i.e., a type of signal or parameter comprising data of different types.
<code>Simulink.Bus</code>	Define a signal bus.

See "Working with Data Type Objects" and "Data Object Classes" in the Simulink online documentation for more information.

This release also adds the following properties to `Simulink.Signal` class:

- `Dimensions`
- `SampleTime`
- `SamplingMode`
- `DataType`
- `Complexity`

Simulink checks the consistency of these properties against the values set on the ports/dwork elements associated with each signal object.

---

**Note** If an attribute is set as auto / -1 (not specified), then no consistency checking is done.

---

## Block Enhancements

This release includes the following block-related enhancements.

### New Blocks

This release introduces the following blocks.

- The Signal Conversion block enables you to convert virtual buses to nonvirtual buses, and vice versa.
- The Environment Controller block's output depends on whether the model is being used for simulation or code generation.
- The Bias block adds a specified bias value to its input and outputs the result.
- Embedded MATLAB Function block enables you to include MATLAB code in models from which you intend to generate code, using the Real-Time Workshop.
- The Model block allows you to include other models in a model (see "Model Referencing" on page 4-3).

### **Fixed-Point-Capable Blocks**

This release adds fixed-point data capability to many existing Simulink blocks and includes fixed-point blocks previously available only with the Fixed-Point Blockset. To use the fixed-point data capability of these blocks, you must install the Simulink Fixed-Point product on your system. See "Fixed-Point Data" in the online Simulink documentation for more information.

### **Port Values Display**

This release of Simulink can display block outputs as data tips on a block diagram while a simulation is running. This allows you to observe block outputs without having to insert Scope or Display blocks. See "Displaying Block Outputs" in the online Simulink documentation for more information.

### **User-Specifiable Sample Times**

This release expands the number of blocks with user-specifiable sample times to include most builtin Simulink blocks. In previous releases, most builtin blocks inherited their sample times directly or indirectly from the blocks to which they were connected. In this release, most blocks continue to inherit their sample times by default. However, you can override this default setting in most cases to specify a nondefault sample time, using either the block's parameter dialog box or a `set_param` command. This avoids the need to use Signal Specification blocks to introduce nondefault sample times at specific points in a model.

## Improved Initial Output Handling

In previous Simulink releases, the Constant, Initial Condition, Unit Delay, and other blocks write out their initial output values in their `mdlStart` method. This behavior can cause unexpected block output initialization. For example, if a Constant block in an enabled subsystems feeds an Outputport block whose IC is set to [ ], the Constant value appears even when the enabled subsystem is not enabled.

It is desirable in some cases for a block to write its initial output value in its `mdlStart` method. For example, discrete integrator block may need to read the value from its external IC port to setting the initial state in `mdlInitialize` method.

This release addresses these problems by implementing a hand-shaking mechanism for handling block initial output. Under this mechanism, a block only computes its initial output value when it is requested by its downstream block. For example, if a Constant block feeds the external IC port of a Discrete Integrator block, the discrete integrator block's external IC port requests the Constant block to compute its initial output value in its `mdlStart` method.

## Bus-Capable Nonvirtual Blocks

In previous releases, Simulink propagated buses only through virtual blocks, such as subsystems. In this release, Simulink also propagates buses through the following nonvirtual blocks:

- Memory
- Merge
- Switch
- Multiport Switch
- Rate Transition
- Unit Delay
- Zero-Order Hold

Some of these blocks impose constraints on bus propagation through them. See the documentation for the individual blocks for more information.

### **Duplicate Input Ports**

This release allows you to create duplicates of Inport blocks in a model. A model can contain any number of duplicates of an original Inport block. The duplicates are graphical representations of the original intended to simplify block diagrams by eliminating unnecessary lines. The duplicate has the same port number, properties, and output as the original. Changing a duplicate's properties changes the original's properties and vice versa. See the Inport block documentation for more information.

### **Inport/Output Block Display Options**

Inport and Output blocks can now optionally display their port number, signal name, or both the number and the name. See the online documentation for the Inport and Output blocks for more information.

### **Zero- and One-Based Indexing**

In this release, some blocks that use indices provide the option for indices to start at 0 or 1. The default is one-based indexing to maintain compatibility with previous releases. Blocks that now support zero- or one-based indexing include

- Selector
- For Iterator
- Assignment

### **Runtime Block API**

This release introduces an application programming interface (API) that enables programmatic access to block data, such as block inputs and outputs, parameters, states, and work vectors, while a simulation is running. You can use this interface to develop MATLAB programs capable of accessing block data while a simulation is running or to access the data from the MATLAB command line. See "Accessing Block Data During Simulation" for more information.

### **Command-Line API to Signal Builder Block**

This release provides a command, `signalbuilder`, for creating and accessing Signal Builder blocks in a model.

## Signal Enhancements

This release includes the following signal-related enhancements.

### Test Point Indicators

This release can optionally use indicators on a block diagram to indicate signals that are test points. See "Test Point Indicators" in the online documentation for more information.

### Edit-Time Signal Label Propagation

In this release, when you change a signal label, Simulink automatically propagates the change to all downstream instances of the label. You do not have to update the diagram as in previous releases.

### Bus Editor

The new Bus Editor enables you to create and modify bus objects in Simulink's base (MATLAB) workspace. See "Bus Editor" for more information.

## Rate Transition Enhancements

This release provides the following enhancements to the handling of rate transitions in models.

### Rate Transition Block Determines Transition Type Automatically

The Rate Transition block now determines the type of transition that occurs between the source and destination block (i.e., fast-to-slow or slow-to-fast). Therefore, this release eliminates the transition type option on the block's parameter dialog.

### Automatic Insertion of Rate Transition Blocks

This release introduces an option to insert hidden rate transition blocks automatically between blocks that operate at different rates. This saves you from having to insert rate transition blocks manually in order to avoid illegal rate transitions. The inserted blocks are configured to ensure that data is transferred deterministically and that data integrity is maintained during the transfer. See "Fixed-Step Solver Options" in the online Simulink documentation for more information.

### **User-Specifiable Output Sample Time**

The Rate Transition Block's parameter dialog box contains a new parameter: **Output Port Sample Time**. This parameter allows you to specify the output rate to which the input rate is converted. If you do not specify a rate, the Rate Transition block inherits its output rate from the block to which its output is connected.

### **Execution Context Enhancements**

This release introduces the following enhancements to execution context propagation.

#### **Enabling Execution Context Propagation**

This release allows you to specify whether to permit execution contexts to be propagated across a conditionally executed subsystem's boundary. See the documentation for the Subsystem block for more information.

#### **Execution Context Indicator**

This release optionally displays a bar across each input port and output port of a subsystem that does not permit propagation of the subsystem's execution context. To enable this option, select **Block Displays->Execution context indicator** from the model editor's **Format** menu.

### **Algebraic Loop Minimization**

This release can eliminate some types of algebraic loops involving atomic or enabled subsystems or referenced models. See "Eliminating Algebraic Loops" in the online Simulink documentation for more information.

### **Level-2 M-File S-Functions**

This release introduces a new application programming interface (API) for creating custom Simulink blocks based on M code. In contrast to the previous API, designated Level 1, which supported a restricted set of block features, the new API, designated Level 2, supports most standard Simulink block features, including support for matrix signals and nondouble data types. See "Writing Level-2 M-file S-functions" in the online documentation for more information.

## **Panning Model Diagrams**

You can now use the mouse to pan around model diagrams that are too large to fit in the model editor's window. To do this, position the mouse over the diagram and hold down the left mouse button and the P or Q key on the keyboard. Moving the mouse now pans the model diagram in the editor window.

## Model Referencing Limitations

This release imposes some limitations on the use of model referencing. For example, in this release, models must meet certain conditions to reference other models or be referenced by other models. This release also restricts the use of model referencing with some features of Simulink and products based on Simulink. This section summarizes some of the major limitations that this release places on model referencing. See Simulink-related product information for additional information on the limitations summarized here.

### Referencing and Referenced Model Limitations

The following limitations apply both to models that reference other models (referencing models) and models referenced by other models.

- The simulation start time of both referencing and referenced models must be 0.
- The model's inline parameters optimization must be selected. If the optimization is off for the top model, this release displays an error message. If it is off for referenced models, this release turns it on while generating model reference targets.
- The model must use `Simulink.Parameter` objects to specify the tunability of model parameters.

This release ignores tunable parameter information specified by the model's **Model Parameter Configuration** dialog box. This release provides a utility function, `tunablevars2parameterobjects`, to facilitate conversion of parameter tunability information from dialog to `Simulink.Parameter` object form.

### Referencing Model Limitations

A model can reference other models if it meets the following conditions:

- The Model Browser does not display Model blocks in its tree view.  
Use the Model Explorer to browse models referenced by a model.
- Tools that require access to a model's internal data or configuration, including the Model Coverage Tool, Report Generator, the Simulink and Stateflow debuggers, and the Profiler, do not work when invoked from a



top model on models that the top model references. This is because the referenced models appear as black boxes to the top model. For example, you cannot use the Simulink Debugger to step from a top model into a referenced model.

- You cannot print a referenced model from a top model.
- You can initialize the states of a top model from the workspace only if you use structure format or you use array format and the top model does not reference any models that have states. You cannot use the workspace to initialize the states of models that the top model references.
- To Workspace and Scope blocks in models referenced by a top model do not log data when you simulate or run code generated from a top model.
- This release does not let you use signal logging to log, or floating scopes to view, the data of Stateflow charts residing in referenced models.
- An enabled or action subsystem cannot reference a model that uses absolute time.
- A continuous sample time cannot be propagated to a Model block that is sample-time independent.
- Linearization is not supported for models containing Model blocks.
- Right clicking on a subsystem to build an S-function from it works for subsystems containing Model blocks only if the model is configured to use `ert.tlc`.

## Referenced Model Limitations

A model can be referenced by other models if it meets the following conditions:

- The model must use a fixed-step solver (see "Choosing a Fixed-Step Solver").
- The model must specify that it can be referenced (see "Total number of instances allowed per top model").
- You cannot initialize the states of a referenced model.
- This release places some restrictions on the sample times that a referenced model can inherit from the model that references it. See "Model Block Sample Times" in the Simulink documentation for more information.

- This release places some limitations on I/O connections in referenced models.
- A referenced model must use `Simulink.Bus` objects to specify any nonvirtual buses that it inputs and outputs (see "Bus I/O Limitations").
- A referenced model can input or output only those user-defined data types that are fixed-point or defined by `Simulink.DataType` or `Simulink.Bus` objects.
- Function-Call, Data Store Memory, Goto/From blocks cannot cross model reference boundaries.
- A referenced model cannot contain noninlined S-functions.
- This release ignores custom code settings in the **Configuration Parameter** dialog box and custom code blocks when generating the simulation target for a referenced model.
- This release does not include Stateflow target custom code in simulation targets generated for referenced models.
- This release does not support referenced models that have asynchronous rates.
- A referenced model can input or output indices. However, Simulink may not be able to detect a 0-based index that is connected to a model port expecting or outputting a 1-based index, or vice versa. See "Index I/O Limitations" in the Simulink documentation for more information.
- Model blocks referencing models that contain assignment blocks that are not in an iterator subsystem, cannot be placed in an iterator subsystems.
- The Real-Time Workshop's S-function target and `grt_malloc`-based target do not support model reference.
- When generating a simulation target for a referenced model that contains an S-function with a TLC file, Simulink inlines the S-function only if the S-function sets the `SS_OPTION_USE_TLC_WITH_ACCELERATOR` flag.
- This release places some restrictions on the use of custom storage classes in referenced models. See The Real-Time Workshop documentation for details.

## Major Bug Fixes

Simulink 6.0 includes several bug fixes made since Version 5.1. This section describes the particularly important Version 6.0 bug fixes.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

If you are upgrading from a version of Simulink earlier than Version 5.1, you should also see “Major Bug Fixes” on page 5-13 for Simulink 5.1.

## Upgrading from an Earlier Release

---

**Note** If you are upgrading from Version 5.0 or earlier, then you should see “Upgrading from an Earlier Release” on page 6-3 in the Simulink 5.0.1 Release Notes.

---

### Changes in MATLAB Data Type Conversions

Release 14 introduces changes in the way MATLAB handles conversions from double to standard MATLAB nondouble data types (e.g., `int8`, `uint8`, etc.) and from one nondouble data type to another. Previous releases of MATLAB use truncation to convert a floating point value to an integer value, e.g., `int8(1.7) = 1`. Release 14 uses rounding, e.g., `int8(1.7) = 2`. See "New Nondouble Mathematics Features" in the Release 14 MATLAB Release Notes for a complete description of the changes in data type conversion algorithms introduced in Release 14.

Such changes could affect the behavior of models that rely on nondouble data type conversions of signals and block parameters. For example, a Gain parameter entered as `int8(3.7)` ends up as 4 in this release as opposed to 3 in previous releases and this difference could change the simulation results. Therefore, if the simulation results for your model differ in Release 14 from previous releases, you should investigate whether the differences result from the changes in data type conversion algorithms, and, if so, modify your model accordingly.

### Signal Object Resolution Changes

In previous releases, Simulink attempted to resolve every named signal to a `Simulink.Signal` object of the same name in the MATLAB workspace. In this release, Simulink lets you specify whether a named signal or discrete state should resolve to a signal object, using the **Signal Properties** dialog box and the **State Properties** of blocks that have discrete states, such as the Discrete-Time Integrator. By default, Simulink attempts to resolve every named signal or state to a signal object regardless of whether the model specifies that the signal or state should resolve to a signal object. If the model does not specify resolution for a signal or state and it does resolve, Simulink displays a warning. You can also specify that Simulink attempt to resolve all

named signals or states without warning of implicit resolutions (the behavior in previous releases) or that it only resolve signals and states that the model specifies should resolve (explicit resolution).

Explicit signal resolution is the recommended approach for doing signal resolution as it ensures that signals that should be resolved are resolved and signals that should not resolve are not resolved. This release includes a script that facilitates converting models that use implicit signal resolution to use explicit resolution. Enter `help disableimplicitsignalresolution` at the MATLAB command line for more information.

## Loading Models Containing Non-ASCII Characters

Release 14 of MATLAB introduces Unicode support. This enhancement allows MATLAB and Simulink to support character sets from different encoding systems. However, this change causes Simulink to behave differently from previous releases when loading a model containing non-ASCII characters. Previous releases load such models regardless of whether the non-ASCII characters are compatible with the current encoding system used by MATLAB. In Release 14, Simulink checks the characters in the model against the current encoding setting of MATLAB. If they are incompatible, Simulink does not load the model. Instead, it displays an error message that prompts you to change to a compatible MATLAB encoding setting, using the `slCharacterEncoding` command.

## Change in Sample Time Behavior of Unary Minus Block

Release 14 changes the sample time behavior of the Unary Minus block. In Release 13, if the sample time of this block's input is continuous, the sample time of the block and its output is fixed in minor time step. this block is fixed in minor step and the output signal is fixed in minor step when the input is a continuous sample time signal. In Release 14, if the input is continuous, the block and output sample time are continuous also.

## Initial Output of Conditionally Executed Subsystems

In previous releases, if the **Initial output** parameter of an Output block in a conditionally executed subsystem specified `[]` as the initial output, the initial output of this port was the initial output of the block driving the Output

block. In this release, the initial output is undefined if the **Initial output** port specifies [ ].

### **Execution Context Default Changes**

In R13 sp1 and DACORE2, execution contexts propagate across conditionally executed subsystem boundaries by default. In R14, execution context propagation does not cross a conditionally executed subsystem boundaries by default. You need to choose the **Propagate execution context across subsystem boundary** option in the subsystem's parameter dialog box.

# Simulink 5.1 Release Notes

---

## New Features

This section summarizes changes and enhancements introduced in Simulink 5.1. Those features are

- "Sample Time Parameters Exposed" (see below)
- "Enhanced Debugger" (see below)
- "Context-Sensitive Data Typing of Tunable Parameters" on page 5-5
- "Conditional Execution Behavior" on page 5-7
- "Function-Call Subsystem Enhancements" on page 5-10
- "External Increment Option Added To For Iterator Block" on page 5-10

### Sample Time Parameters Exposed

Sample time parameters of most Simulink built-in library blocks have been exposed to the user. That is, the sample time parameter of these blocks has been made accessible via the block's dialog box or `set_param`. This means that most nonvirtual blocks in the Simulink library have a user settable sample time parameter. Prior to this exposure, these blocks had an internal inherited sample time with the exception of the Constant block, which had a constant (`inf`) sample time. By providing access to the sample time parameter, you no longer need to use the Signal Specification block to apply a nondefault sample times to these blocks.

### Enhanced Debugger

This release includes enhancements to the Simulink debugger that enable you to step through a simulation showing information not visible in previous releases. The enhancements include

- An expanded command set that now enables you to step a simulation method by method. Previous releases showed only output methods.
- An expanded toolbar that gives you push button access to new debugger commands
- A Simulation Loop pane that shows the current state of the simulation at a glance



---

**Note** Methods are functions that Simulink uses to solve a model at each time step during the simulation. Blocks are made up of multiple methods. "Block execution" in this documentation is shorthand notation for "block methods execution." Block diagram execution is a multi-step operation that requires execution of the different block methods in all the blocks in a diagram at various points during the process of solving a model at each time step during simulation, as specified by the simulation loop.

---

These changes allow you to pinpoint problems in your model with greater accuracy. The following sections briefly describe the debugger enhancements. See the Simulink documentation for a detailed description of the new features and their usage.

## Enhanced Debugger Commands

This release enhances the following debugger commands:

- `step`

In previous releases, this command advanced the simulation from the current block Outputs method over any intervening methods to the next block Outputs method. In this release, `step` advances the simulation method by method, or into, over, or out of methods, from the first method executed during the simulation to the last. This allows you to determine the result of executing any model, subsystem, or block method executed during the simulation, including block Outputs, Update, and Derivative methods as well as solver methods.

- `next`

In previous releases, this command advanced the simulation to the first block Outputs method executed during the next time step. In this release, it advances the simulation over the next method to be executed, executing any methods invoked by the next method.

- `break`

In previous releases, this command set a breakpoint at the Outputs method of a specified block. In the current release, it sets a breakpoint at any specified method or on all the methods of a specified block.

- `bafter`

In previous releases, this command set a breakpoint after the Outputs method of a specified block. In this release, it sets a breakpoint after a specified method or after each of the methods of a specified block.

- `minor`

In previous releases, this command enabled or disabled stepping across Outputs methods in minor time steps. In the current release, it enables or disables in minor time steps breakpoints set by block for all methods.

### **New Debugger Commands**

This release introduces the following debugger commands:

- `elist`

Displays the method execution lists for the root system and the nonvirtual subsystems of the model being debugged.

- `etrace`

Causes the debugger to display a message in the MATLAB Command Window every time a method is entered or exited while the simulation is running.

- `where`

Displays the call stack of the method at which the simulation is currently suspended.

### **Enhanced Debugger Toolbar**

The debugger toolbar has been expanded to include buttons for the following versions of the step command: `step into`, `step over`, `step out`, and `step top`.

### **Simulation Loop Pane**

This release adds a **Simulation Loop** pane to the debugger GUI that displays by method the point in the simulation loop at which the simulation is currently suspended. The debugger updates the pane after each `step`, `next`, or `continue` command, enabling you to determine at a glance the point to which the command advanced the simulation. The pane also allows you to

set breakpoints on simulation loop methods and to navigate to the block at whose method the simulation is currently suspended.

### Sorted List Pane

This release renames the **Block Execution List** pane of the debugger GUI to the **Sorted List** pane to reflect more accurately what the pane contains. The Sorted List pane displays for the root system and each nonvirtual subsystem of the model being debugged a sorted list of the subsystem's blocks. The sorted lists enable you to determine the block IDs of a model's blocks.

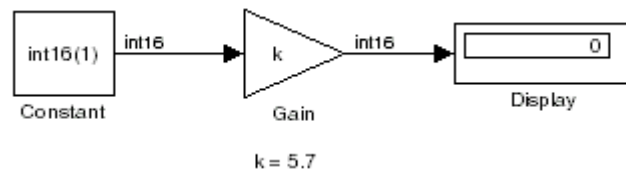
### Context-Sensitive Data Typing of Tunable Parameters

In this release, if a model's **Inline parameters** setting is selected, Simulink regards the data type of a tunable parameter as context-sensitive if the data type is not specified. In particular, this release allows the block that uses the parameter to determine the parameter's data type. By contrast, Release 13 regards the type of the parameter to be double regardless of where it is used.

### Change in Simulink Behavior

This change affects the behavior of Simulink in two cases. First, in Release 13, if a tunable parameter's data type is unspecified and a block that uses it needs to convert its type from double to another type, Simulink by default stops and displays an error message when you update or simulate the model. The error alerts the user to the fact that the type conversion is a downcast and hence could result in a loss of precision. In this release, by contrast, a typecast never occurs because the block itself determines the appropriate type for the parameter. Hence, in this release, Simulink never generates a downcast error for tunable parameters of unspecified data type.

The following model illustrates the difference in behavior between this release and Release 13 in this case.



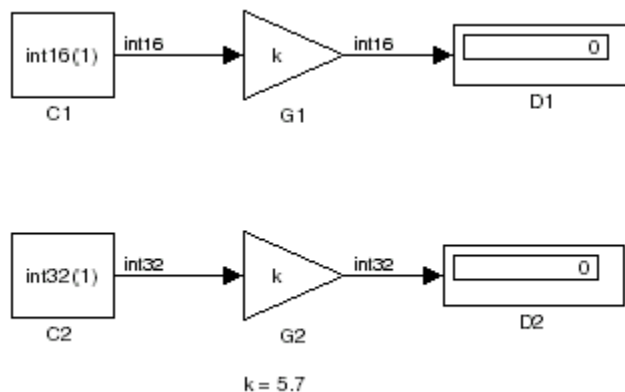
Assume that the model's **Inline parameters** setting is selected (thereby making parameters nontunable by default) and the model declares `k` as a tunable parameter on the **Model Configuration Parameters** dialog box. Also assume that the user has specified the value of `k` on the MATLAB command line as follows:

```
>> k = 5.7
```

In other words, the user has specified a value for `k` but not a data type. In this case, this release regards the type of `k` to be `int16`, the type required by the Gain block to compute its output. By contrast, Release 13 regards the type of `k` to be `double` and hence assumes that the Gain block must downcast `k` to compute its output. Release 13 therefore stops and displays an error message by default in this case when you update or simulate the model.

The behavior of this release also differs from Release 13 in the case where a model uses a tunable parameter of unspecified data type in more than one place in the model and the required data type differs in different places. This case creates a conflict under the assumption that the block in which the parameter is used determines the parameter's data type. This assumption requires Simulink to assign different data types to the same parameter, which is impossible. Therefore, in this release, Simulink signals an error to alert the user to the conflict. By contrast, in Release 13, Simulink does not throw an error because the data type of the parameter is `double` regardless of where it is used. You can avoid the conflicting data types error in Release 13SP1 by specifying the tunable parameter's data type.

The following model illustrates this change in behavior.



The two Gain blocks in this model both use `k`, a tunable parameter of unspecified type, as their gain parameter. Computing the outputs of the blocks requires that the gain parameter be of types `int16` and `int32`, respectively. In Release 13, Simulink regards the data type of `k` to be `double` and the Gain blocks use typecasts to convert `k` to the required type in each case. Simulink simulates the model without error (if the parameter downcasting diagnostic is set to `none` or `warning`). By contrast, this release signals an error because this model requires `k` to be both type `int16` and `int32`, an impossibility. You can avoid this error by explicitly specifying `k`'s data type; for example:

```
k = int16(6);
```

## Conditional Execution Behavior

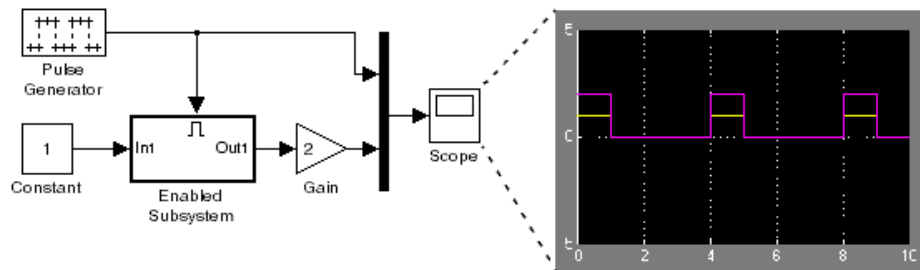
This release augments the conditional input branch behavior of the previous release with a more generalized behavior called *conditional execution (CE)* behavior. The new behavior speeds simulation of models by eliminating unnecessary execution of blocks connected to Switch, Multiport Switch, and conditionally executed blocks.

---

**Note** The Simulink documentation has not yet been updated to reflect the new behavior. Consequently, the remainder of this release note provides a detailed explanation of how the behavior works.

---

As with the conditional input branch behavior available in the previous release, the new behavior ensures that the block methods that make up an input branch of a Switch or Multiport Switch block execute only when the model selects the corresponding switch input. In addition, the new behavior option generalizes this behavior to conditionally executed subsystems. Consider, for example, the following model.



Simulink computes the outputs of the Constant block and Gain Block only when the Enabled Subsystem executes (i.e., at time steps 0, 4, 8, and so on). This is because the output of the Constant block is required and the input of the Gain block changes only when the Enabled Subsystem executes. When CE behavior is off, Simulink computes the outputs of the Constant and Gain blocks at every time step, regardless of whether the outputs are needed or change.

In this example, Simulink regards the Enabled Subsystem as defining an execution context for the Constant and Gain blocks. Although the blocks reside in the model's root system, their block methods are executed as if the blocks reside in the Enabled Subsystem.

In general, Simulink defines an *execution context* as a set of blocks to be executed as a unit. At model compilation time, Simulink associates an execution context with the model's root system and with each of its nonvirtual subsystems. Initially, the execution context of the root system and each nonvirtual subsystem is simply the blocks that it contains. Simulink examines whether a block's output is required only by a conditionally executed subsystem or whether the block's input changes only as a result of the execution of a conditionally executed subsystem. If so, Simulink moves the block into the execution context of the conditionally executed system. This

ensures that the block methods are executed during the simulation loop only when the corresponding conditionally executed subsystem executes.

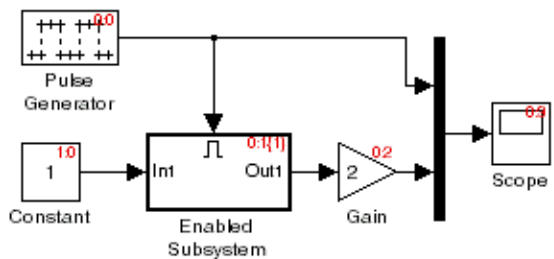
---

**Note** This behavior treats the input branches of a Switch or Multipoint Switch block as invisible, conditionally executed subsystems, each of which has its own execution context that is enabled only when the switch's control input selects the corresponding data input. As a result, switch branches execute only when selected by switch control inputs.

---

To determine the execution context to which a block belongs, select **Sorted order** from the model window's **Format** menu. Simulink displays the sorted order index for each block in the model in the upper right corner of its icon. The index has the format  $s:b$ , where  $s$  specifies the subsystem to whose execution context the block,  $b$ , belongs.

Simulink also expands the sorted order index of conditionally executed subsystems to include the system ID of the subsystem itself in curly brackets as illustrated in the following figure.



In this example, the sorted order index of the enabled subsystem is  $0:1\{1\}$ . The 0 indicates that the enable subsystem resides in the model's root system. The first 1 indicates that the enabled subsystem is the second block on the root system's sorted list (zero-based indexing). The 1 in curly brackets indicates that the system index of the enabled subsystem itself is 1. Thus any block whose system index is 1 belongs to the execution context of the enabled subsystem and hence executes when it does. For example, the constant block's

index, 1:0, indicates that it is the first block on the sorted list of the enabled subsystem, even though it resides in the root system.

## Function-Call Subsystem Enhancements

This release adds the following function-call subsystem-related parameters to the Trigger block:

- The **States when enabling** parameter specifies whether a function-call enable trigger causes Simulink to reset the states of the subsystem containing this Trigger block to their initial values.
- The **Sample time type** parameter specifies whether the function-call subsystem containing the Trigger block is invoked periodically.
- The **Sample time** parameter specifies the rate at which the function-call subsystem containing the Trigger block is invoked.

See the Trigger block documentation for additional information.

## External Increment Option Added To For Iterator Block

This release adds an external increment option to the For Iterator block. Selecting this option causes the block to display an input port for the external increment. The value of this input port at the current time step is used as the value of the block's iteration variable at the next iteration. You can select this option by checking the **Set next i (iteration variable)** externally option on the block's parameter dialog box or by setting its ExternalIncrement parameter to 'on'. See the documentation for the For Iterator block for more information.

---

**Note** This enhancement is not backward compatible with R13. Loading models containing For Iterator blocks with this option selected in R13 produces a warning message. Simulating such models in R13 can produce incorrect results.

---



## Performance Improvements

Release R13SP1 includes many performance improvements that were designed to particularly benefit large models (containing on the order of 100,000 blocks and/or more than a few megabytes of parameter data). Speed has been improved and memory consumption reduced for model loading, compilation, code generation, and closing. The various improvements span the Simulink, Stateflow, and Real-Time Workshop products and include:

- Increased speed and decreased memory consumption through improved incremental loading of library blocks that contain Stateflow blocks.
- Increased speed and decreased memory usage through the introduction of a redesigned Signal Specification block. Models saved with the old version of the Signal Specification block should automatically start using the new block when you load the model with this release.
- Increased speed in datatype and sample time propagation during the compile phase of certain models.
- Increased speed in the Stateflow build process for both simulation and Real-Time Workshop targets.
- Increased speed and decreased memory consumption when using N-D Lookup Table blocks that utilize large parameter data.
- Increased speed and decreased memory usage when generating code with Real-Time Workshop or the Simulink Accelerator for models with large parameter sets. This improvement involves writing out parameter references instead of the entire parameter data into the RTW file for parameters whose size exceeds 10 elements. The parameter values for such references are retrieved directly from Simulink during the code generation process.
- Decreased memory usage during various phases of code generation process in Real-Time Workshop or the Simulink Accelerator.
- Improved speed during model close through streamlining of the close process.

Other minor improvements have also been made to improve performance. Your models should experience corresponding speed and memory

improvements, to the extent that these changes apply to your specific models and usage scenarios.

## Major Bug Fixes

Simulink 5.1 includes several bug fixes made since Version 5.0.1. This section describes the particularly important Version 5.1 bug fixes.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

## Upgrading from an Earlier Release

---

**Note** If you are upgrading from a version earlier than 5.0.1 (or 5.0.2, which did not significantly differ from 5.0.1), then you should see “Upgrading from an Earlier Release” on page 6-3 in the Simulink 5.0.1 Release Notes.

---

# Simulink 5.0.1 Release Notes

---

## Major Bug Fixes

Simulink 5.0.1 includes several bug fixes made since Version 5.0. This section describes the particularly important Version 5.0.1 bug fixes.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

If you are upgrading from a release earlier than Release 13, then you should also see “Major Bug Fixes” on page 7-11 of the Simulink 5.0 Release Notes.

## Upgrading from an Earlier Release

Below is an upgrade issue involved in upgrading from Simulink 5.0 to Version 5.0.1.

If you are upgrading from a version earlier than 5.0, then you should see “Upgrading from an Earlier Release” on page 7-14 in the Simulink 5.0 Release Notes.

### **Backwards Compatibility of Tunable Parameters for Unified Fixed-Point Blocks**

Unified fixed-point blocks with tunable parameters have compatibility problems under certain conditions in Release 13. The problem arises only if a tunable parameter is mapped to a built-in integer or single data type. When tunable parameters are mapped to built-in integers or single, the code generated by Real Time Workshop will be different for unified blocks than it was for Fixed-Point Blockset blocks in prior releases. There are no compatibility problems if a tunable parameter maps to a nonbuilt-in data type, such as a scaled fixed-point integer.

Tunable parameters are entered in a Simulink model by specifying the name of a MATLAB variable in a block’s dialog. This variable can be either a plain MATLAB variable or a Simulink parameter object. In either case, a numerical value will be defined for this tunable parameter by doing an assignment in MATLAB. MATLAB supports several numerical data types including the eight Simulink built-in numerical data types: double, single, int8, uint8, int16, uint16, int32, and uint32. One of these eight data types can be used when a value is defined for a MATLAB variable. The effect of the data type of the MATLAB variable is significantly different depending on how the tunable parameter is used in Simulink.

For Simulink built-in blocks, the legacy rule is to fully respect the data type used for the value of a MATLAB variable. Whatever data type is used in MATLAB when assigning a value to a variable is also be used when declaring that parameter in code generated by Real Time Workshop. The use of that parameter by a block may require the value to be represented using a different data type. If so, additional code is generated to convert the parameter every time it is used by the block. To get the most efficient code

for a given block, the value of the MATLAB variable should use the same data type as is needed by the block.

For Fixed-Point Blockset blocks, the legacy rule is to expect no data type information from the MATLAB variable used for the tunable parameter. A fundamental reason for this is that MATLAB does not have native support for fixed-point data types and scaling, so the Simulink built-in legacy rule could not be directly extended to the general fixed-point case. Many fixed-point blocks automatically determine the data type and scaling for parameters based on what leads to the most efficient implementation of a given block. However, certain blocks such as Constant, as well as blocks that use tunable parameters in multiplication, do not imply a unique best choice for the data type and scaling of the parameter. These blocks have provided separate parameters on their dialogs for entering this information.

In Release 13, many Simulink built-in blocks and Fixed-Point Blockset blocks were unified. The Saturation block is an example of a unified block. The Saturation block appears in both the Simulink Library and in the Fixed-Point Blockset Library, but regardless of where it appears it has identical behavior. This identical unified behavior includes the treatment of tunable parameters. The dissimilarity of the legacy rules for tunable parameters has led to a shortcoming in the unified blocks. Unified blocks obey the Simulink legacy rule sometimes and the Fixed-Point Blockset legacy rule at other times. If the block is using the parameter with built-in Simulink data types, then the Simulink legacy rule applies. If the block is using the parameter with nonbuilt-in data types, such as scaled fixed-point data types, then the Fixed-Point Blockset legacy rule applies. This gives full backwards compatibility with one important exception.

The backwards compatibility issue arises when a model created prior to R13 uses a Fixed-Point Blockset block with a tunable parameter, and the data type used by the block happens to be a built-in data type. If the block is unified, it will now handle the parameter using the Simulink legacy rule rather than the Fixed-Point Blockset legacy rule. This can have a significant impact. For example, suppose the tunable parameter is used in a Saturation block and the data type of the input signal is a built-in `int16`. In prior releases, the Fixed-Point Blockset block would have declared the parameter as an `int16`. For legacy fixed-point models, the MATLAB variables used for tunable parameters invariably gave their value using floating-point `double`. The unified Saturation block would now declare the tunable parameter in the



generated code as double. This has several negatives. The variable takes up six more bytes of memory as a double than as an `int16`. The code for the Saturation block now includes conversions from double to `int16` that execute every time the block executes. This increases code size and slows down execution. If the design was intended for use on a fixed-point processor, the use of floating-point variables and floating-point conversion code is likely to be unacceptable. It should be noted that the numerical behavior of the blocks is not changed even though the generated code is different.

For an individual block, the backwards compatibility issue is easily solved. The solution involves understanding that the Simulink legacy rule is being applied. The Simulink legacy rule preserves the data type used when assigning the value to the MATLAB variable. The problem is that an undesired data type will be used in the generated code. To solve this, you should change the way you assign the value of the tunable parameter. Determine what data type is desired in the generated code, then use an explicit type cast when assigning the value in MATLAB. For example, if `int16` is desired in the generated code and the initial value is 3, then assign the value in MATLAB as `int16(3)`. The generated code will now be as desired.

A preliminary step to solving this issue with tunable parameters is identifying which blocks are affected. In most cases, the treatment of the parameter will involve a downcast from double to a smaller data type. On the **Diagnostics** tab of the **Simulation Parameters** dialog is a line item called Parameter downcast. Setting this item to Warning or None will help identify the blocks whose tunable parameters require reassignment of their variables.

In R13, the solution described above did not work for three unified blocks: Switch, Look-Up Table, and Lookup Table (2-D). These blocks caused errors when the value of a tunable parameter was specified using integer data types. This was a false error and has been removed. Using an explicit type cast when assigning a value to the MATLAB variable now solves the issue of generating code with the desired data types.



# Simulink 5.0 Release Notes

---

## New Features

---

**Note** Simulink 5.0 incorporates changes introduced in Simulink 4.1.1, which was initially released in Web-downloadable form after Release 12.1 was released, but before Release 13. These Release Notes describe those changes, as well as other changes introduced after Version 4.1.1.

---

Simulink 5.0 introduces features and enhancements in the following areas:

- “Block Enhancements” on page 7-2
- “Simulation Enhancements” on page 7-6
- “Modeling Enhancements” on page 7-8

If you are upgrading from a release earlier than Release 12.1, then you should also see “New Features” on page 8-2 in the Simulink 4.1 Release Notes.

### Block Enhancements

Simulink 5.0 includes the following block-related enhancements:

- “Fixed-Point Block Library” on page 7-3
- “Look-Up Table Editor” on page 7-3
- “Model Verification Block Library” on page 7-4
- “Signal Builder Block” on page 7-4
- “DocBlock” on page 7-4
- “Rate Transition Block” on page 7-4
- “Block Library Reorganization” on page 7-4
- “Model Linearization Blocks” on page 7-4
- “Data Read/Write Block Navigation” on page 7-5
- “Enhanced S-Function Builder” on page 7-5
- “Miscellaneous Block Enhancements” on page 7-5

## Fixed-Point Block Library

Simulink now includes the latest version (4.0) of the Fixed-Point Blockset. The library was previously available only as a separately installed option. You must have a Fixed-Point Blockset license to run models containing fixed-point blocks in fixed-point mode. However, you can open, edit, and run such models in floating-point mode, regardless of whether you have a Fixed-Point Blockset license. This change facilitates sharing of fixed-point models in large organizations by eliminating the need for all users in a group to have a Fixed-Point Blockset license in order to run or modify models containing fixed-point blocks. See "Installation and Licensing" in the Fixed-Point Blockset release notes for information on how to run models containing fixed-point blocks when you do not have a Fixed-Point Blockset license.

This release also unifies many core Simulink and Fixed-Point Blockset blocks that have similar functionality. For example, the Sum block in the Simulink Math Operations library and the Sum block in the Fixed-Point Blockset Math library are now the same block. As a result, you no longer have to replace any of the unified blocks when switching from built-in to fixed-point data types and vice versa. You can change the data types of the blocks simply by selecting the appropriate settings on their parameter dialog boxes. See "Unified Simulink and Fixed-Point Blockset Blocks" in the Fixed-Point Blockset release notes for more information and for a list of blocks that this release unifies.

---

**Note** When you open an existing model, Simulink 5.0 updates the model to use the unified version of a standard or Fixed-Point Blockset block wherever an instance of that block occurs in the model. Simulink sets the parameters of the unified block to preserve the behavior of the original block. For example, wherever your existing model contains a Sum block from the Fixed-Point Blockset library, Simulink replaces the Fixed-Point Blockset version with a unified Sum block set to operate as a fixed-point block. This automatic updating ensures that your existing model runs the same in Simulink 5.0 as it did in previous releases of Simulink.

---

## Look-Up Table Editor

The Look-Up Table Editor allows you to find and edit the contents of look-up tables used by look-up table blocks. See "Look-Up Table Editor" in the online Simulink documentation for more information.

### **Model Verification Block Library**

Simulink now includes a library of model verification blocks that enable you to create self-validating models. For example, you can use the blocks to test that signals do not exceed specified limits during simulation. When you are satisfied that a model is correct, you can turn error-checking off by disabling the model verification blocks. You do not have to physically remove them from the model. The library includes set of blocks preconfigured to check for common types of errors, for example, signals that exceed a specified upper or lower bound. See "Model Verification" in the online Simulink documentation for more information.

### **Signal Builder Block**

The new Signal Builder block allows you to create interchangeable groups of signal sources and quickly switch the groups into and out of a model. The Signal Builder block's signal editor allows you to define the waveforms of the signals output by the block. You can specify any waveform that is piecewise linear. Signal groups can greatly facilitate testing a model, especially when used in conjunction with Simulink assertion blocks and the optional Model Coverage Tool. See "Working with Signal Groups" for more information.

### **DocBlock**

The new DocBlock block allows you to create text that documents a model and save that text with the model.

### **Rate Transition Block**

Simulink now includes a Rate Transition block that allows you to specify the data transfer mechanism between two rates of a multirate system. See Rate Transition in the online Simulink block reference for more information.

### **Block Library Reorganization**

The Simulink Block Library has been reorganized to simplify accessing blocks with related functionality. See for more information.

### **Model Linearization Blocks**

This release introduces two blocks that generate linear models from a Simulink model at various times during a simulation. The Time-Based

Linearization block generates linear models at specified time steps. The Trigger-Based Linearization block generates models when triggered by events appearing at its trigger port.

## Data Read/Write Block Navigation

To come

## Enhanced S-Function Builder

The S-Function Builder has been enhanced to generate S-functions with the following additional capabilities

- Multiple ports
- Support for all builtin datatypes
- Support for 2-D signals
- Support for complex signals

See "Building S-Functions Automatically" for more information.

## Miscellaneous Block Enhancements

This release introduces the following enhancements to Simulink blocks.

**Math Function Block.** This release significantly speeds up the simulation of the Math Function block's exponential math functions. All functions now support both double- and single-precision floating-point inputs and outputs. The `mod` and `rem` functions also support inputs and outputs of all integer types. The `transpose` and `hermitian` functions support all data types. When optimizations are enabled, the conjugate operation on a real signal invokes the block reduction optimization, as that case is a no-op. In-place multiplies for the  $\text{magnitude}^2$  operation are used for reused block I/O on real signals.

**Gain Block.** The Gain block now performs block reduction when block reduction is on, `inline parameters=ON`, and the gain is both nontunable and unity.

**Width Block.** The Width block now includes a parameter to specify the datatype of the output.

**Real Data Type Support.** The following blocks now operate on both double precision and single precision floating point signals:

- Dot Product
- Trigonometric
- Matrix Inversion

### **Block Data Type Table**

To view a table that summarizes the data types supported by the blocks in the Simulink and Fixed-Point block libraries, execute the following command at the MATLAB command line:

```
showblockdatatypetable
```

### **Simulation Enhancements**

Simulink 5.0 includes the following new features and enhancements to simulation of Simulink models.

- “Invalid Loop Highlighting” on page 7-6
- “Algebraic Loop Highlighting” on page 7-7
- “Conditional Input Branch Execution” on page 7-7
- “Reorganized Simulation Diagnostics” on page 7-7
- “Enhanced Diagnostic Viewer” on page 7-7

### **Invalid Loop Highlighting**

Simulink now detects and highlights several kinds of invalid loops:

- Loops that create invalid function-call connections or an attempt to modify the input/output arguments of a function call
- Loops containing non-latched triggered subsystems
- Self-triggering subsystems
- Loops containing action subsystems in a cycle



This makes it is easier to identify and fix the loop. See "Avoiding Invalid Loops" for more information.

### **Algebraic Loop Highlighting**

Simulink now optionally highlights algebraic loops when you update or simulate a model. See "Highlighting Algebraic Loops" for more information. The `ashow debug` command without any arguments now lists all of a model's algebraic loops in the MATLAB command window.

### **Conditional Input Branch Execution**

This release introduces a new optimization called conditional input branch execution. Previously, when simulating models containing Switch or Multiport Switch blocks, Simulink executed all blocks required to compute all inputs to each switch at each time step. In this release, Simulink, by default, executes only the blocks required to compute the control input and the data input selected by the control input at each time step. Similarly, code generated from the model by Real-Time Workshop executes only the code needed to compute the control input and the selected data input. This optimization speeds simulation and execution of code generated from the model. See "Optimizations" for more information.

### **Reorganized Simulation Diagnostics**

The **Diagnostics Pane** of the **Simulation Parameters** dialog box now groups diagnostics by functionality. This makes it easier to find and configure related diagnostics.

### **Enhanced Diagnostic Viewer**

This release introduces an enhanced Diagnostic Viewer. Improvements include

- Identical appearance on UNIX and Windows
- Hyperlinks to Simulink, Stateflow, and Real-Time Workshop objects that caused the errors displayed in the viewer
- Sortable error list

Clicking a column head sorts the error list by the contents of that column.

- Configurable content

The **View** menu allows you to choose which information to display in the viewer.

- Selectable font size

The **FontSize** menu allows you to choose the size of the font used to display error messages.

See "Simulation Diagnostic Viewer" for more information.

## Modeling Enhancements

The following enhancements facilitate creation of Simulink models.

- “Enhanced Mask Editor” on page 7-8
- “Production Hardware Characteristics” on page 7-9
- “Including Symbols and Greek Letters in Block Diagrams” on page 7-9
- “True Color Support” on page 7-9
- “Print Details” on page 7-9
- “Boolean Logic Signals” on page 7-9
- “Model Discretizer” on page 7-10

## Enhanced Mask Editor

This release introduces changes to the Mask Editor designed to improve usability. Changes include

- Block parameter information moves from the **Initialization** pane to a new pane entitled **Parameters**.
- The **Parameters** pane allows you to specify a callback function to be called when the value of a parameter changes.
- The **Parameters** pane allows you to specify via check boxes whether a parameter is visible on the masked block’s dialog box and whether a parameter is tunable.

- The **Icon** pane provides a list of examples of all the types of drawing commands that can be used to draw the block's icon.

See "Creating Masked Subsystems" in the online Simulink documentation for more information.

## Including Symbols and Greek Letters in Block Diagrams

This release allows you to include symbols, Greek letters, and other formatting in annotations, masked subsystem port labels, and masked subsystem icon text. You do this by including TeX formatting commands in the annotation, port label, or icon text.

## Production Hardware Characteristics

**Production hardware characteristics** is a new setting on the **Advanced** pane of the **Simulation parameters** dialog box. This setting, intended for use in modeling, simulating, and generating code for digital systems, allows you to specify the sizes of the data types supported by the system being modeled. Simulink uses this information to automate the choice of data types for signals output by some blocks. See "The Advanced Pane" for more information.

## True Color Support

This release allows you to use any color supported by your system as the foreground or background colors of a block diagram. See "Specifying Block Diagram Colors" in the online documentation for more information.

## Print Details

This command generates an HTML report detailing the contents of the currently selected model (see "Generating a Model Report" in the online documentation for more information).

## Boolean Logic Signals

In previous releases, the Boolean logic signals optimization was off by default for new models (see "Optimizations" in the online Simulink documentation for a description of this option). In the current release, the

optimization is on by default for new models. This change does not affect existing models.

### **Model Discretizer**

The Model Discretizer tool selectively replaces continuous Simulink blocks with discrete equivalents. Discretization is critical in digital controller design for dynamic systems and for hardware in the loop simulations. You can use this tool to prepare continuous models for use with the Real-Time Workshop Embedded Coder, which supports only discrete blocks. See "Model Discretizer" in the online documentation for more information.

## Major Bug Fixes

Simulink 5.0 includes several bug fixes made since Version 4.1. This section describes the particularly important Version 5.0 bug fixes.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

If you are upgrading from a release earlier than Release 12.1, then you should also see “Bug Fixes” on page 8-11.

## Platform Limitations for HP and IBM

The following are platform limitations for Simulink 5.0 for the HP and IBM platforms that are new limitations, as of Version 5.0.

- New version of the Mask Editor
- New version of the Diagnostic Viewer
- The Model Discretizer

---

**Note** The Release 12 and 12.1 platform limitations for Simulink for the HP and IBM platforms still apply to Release 13. These are listed below.

---

The following Java-dependent Simulink features, introduced in Simulink 4.1, are *not* available on the HP and IBM platforms.

- Simulink Data Class Designer
- S-Function Builder
- Look-Up Table Editor

In addition, the following Simulink features are not supported on the HP and IBM platforms:

- Simulink Editor's **Find** dialog  
Use the `find_system` command instead.
- GUI interface to the Simulink Debugger  
Use the command-line interface instead.
- The **View Changes** dialog box for modified library links  
Instead, select the modified link and execute  
`ld=get_param(gcb, 'LinkData')` to get a structure that lists  
the parameter differences between the library and local instance of the  
block. Edit this structure and execute `set_param(gcb, 'LinkData', ld)` to  
apply the changes.
- **Parameter** dialog for the Configurable Subsystem block.

Use the `set_param` command instead to set the block's parameters.

- Model Discretizer

## Upgrading from an Earlier Release

This section describes an upgrade issue involved in moving from Simulink 4.1 to Version 5.0.

If you are upgrading from a version earlier than 4.1, then you should see “Upgrading from an Earlier Release” on page 8-13 in the Simulink 4.1 Release Notes.

### **BlockInstanceData Function Deprecated**

S-functions should no longer call the `BlockInstanceData` function. All data used by a block should be declared using data type work vectors (e.g., `DWORK`).



# Simulink 4.1 Release Notes

---

## New Features

This section introduces the new features and enhancements added in Simulink 4.1 since Simulink 4.0 (Release 12.0).

For information about Simulink features that are incorporated from recent releases, see “New Features” on page 9-2 in the Simulink 4.0 Release Notes.

This section about new Simulink features is organized into the following subsections:

- “Simulink Editor” on page 8-2
- “Modeling Enhancements” on page 8-3
- “Simulink Debugger” on page 8-7
- “Block Library” on page 8-8

### Simulink Editor

This section describes enhancements to the Simulink Editor.

#### Undo Move

In Simulink 4.1, the **Undo** command on the Simulink **Edit** menu restores blocks, annotations, lines, and nodes that have moved to their original locations (see "Undoing a Command" in Using Simulink).

#### Undo Subsystem Creation

In Simulink 4.1, the **Undo** command on the Simulink **Edit** menu restores blocks that have been grouped into a subsystem to their original level in the model (see "Undoing Subsystem Creation" in Using Simulink).

#### Autoconnecting Blocks

This version makes connecting blocks significantly easier. To connect a set of source blocks to a target block, simply select the source blocks, hold down the **Ctrl** key and left-click the target block. Simulink draws connecting lines between the source blocks and the destination block, neatly routing lines around intervening blocks. To connect a source block to a set of target blocks, select the target blocks, hold down the **Ctrl** key and left-click the source block.

To connect two blocks, select the source block, and left-click the destination block while holding down the **Ctrl** key. Simulink connects as many ports on the two blocks as possible (see "Autoconnecting Blocks" in *Using Simulink*).

### **Autorouting Signal Lines**

Simulink now routes signal lines around intervening blocks when you connect them either interactively (by dragging the connecting lines or using autoconnect) or programmatically via the `add_line` command's new 'autorouting' option (see "Autorouting Option Added to `add_line` Command" on page 8-4).

### **Displaying Storage Class on Lines**

This version adds an item to the **Format** menu, which toggles the display of (nonAuto) storage class on signal lines (see "RTW Storage Class" in *Using Simulink* for more information).

### **Save Models in Release 11 Format**

This release can save post-Release 11 models in Release 11 format. Simulink 3 (Release 11) can load and run converted models that do not use any post-Release 11 features of Simulink. Simulink 3 can load converted models that use post-Release 11 features but may not be able to simulate the model correctly. Use the **Save as** option from the Simulink **File** menu or the following command to save a model in Release 11 format.

```
s1saveas(SYS)
```

See "Saving a Model in Simulink 3 (R11) format" in *Using Simulink* for more information.

### **Modeling Enhancements**

This section describes enhancements to Simulink dynamic system modeling tools.

### **Autorouting Option Added to add\_line Command**

The `add_line` command now optionally routes lines around intervening blocks and annotations. For example, the following command autoroutes a connection between two blocks in the `vdp` model.

```
add_line('vdp','Product/1','Mu/1','autorouting','on')
```

The autorouting option is off by default. See `add_line` in *Using Simulink* for more information.

### **S-Function Builder**

The S-Function Builder generates an S-function from specifications that you enter in a dialog box. It provides an easy way for you to incorporate existing code into a Simulink model.

### **add\_param, delete\_param**

With this version, you can add custom parameters to your block diagrams.

```
add_param('modelName','MyParameterName','value')
delete_param('modelName','MyParameterName')
```

You can also use the model handle in place of the model name. See `add_param` and `delete_param` in *Using Simulink* for more information.

### **Connection Callbacks**

With this version, you can use `set_param` to set callbacks on ports that are triggered by changes in the ports' connectivity. The callback function parameter is named `ConnectionCallback`. When the port's connectivity changes (addition/deletion of line connected to the port, connection of new block to the port, etc.), Simulink invokes the callback function with the port handle as its argument. See "Port Callback Parameters" in *Using Simulink* for more information.

### **Saving Block User Data in Model Files**

This version adds a new block parameter, named `UserDataPersistent`, that is off by default. Setting this parameter on, e.g.,

```
set_param(block-name, 'UserDataPersistent', 'on')
```

causes Simulink to include a block's user data (i.e., the value of the block's `UserData` parameter) in the model file when you save a model. Simulink encodes the user data as ASCII characters and saves the encoded data in a new section of the model file called `MatData`. This mechanism works with all forms of MATLAB data, including arrays, structures, objects, and Simulink data objects. See "Associating User Data with Blocks" in *Using Simulink* for more information.

### **Absolute Tolerance Enhancements**

This version adds a dialog item for setting the absolute tolerance for each state in the State-Space block, the Transfer Fcn block, and the Zero-Pole block. With this enhancement, you can now specify the absolute tolerance for solving every continuous state in your model.

### **Block Reduction Enhancements**

S-functions may now request that they be eliminated from the compiled model. To do this, call `ssSetBlockReduction(true)` inside the S-function. This is an advanced feature provided for customers writing S-functions who want to optimize the generated code produced for their S-function. Graphical connectivity is now remapped during block reduction, eliminating a source of error during reduction (e.g., a memory reference error used to occur if Simulink eliminated a block connected to a scope). Block reduction is now on by default, and a Simulink preference has been added for the option.

### **Boolean Logic Signals Preference**

The Simulink Preferences dialog box now allows you to specify the use of Boolean logic signals by default. See "Setting Simulink Preferences" in *Using Simulink* for more information.

### **Subsystem Semantics Demos**

Typing `sl_subsys_semantics` at the MATLAB prompt now displays a set of models that illustrate the semantics of various types of subsystem blocks. The demos include formal definitions of function-call subsystems.

### **Enhanced Engine Model Demos**

The top and bottom dead center detection in the engine and enginecw demo models now use a reset integrator. In previous versions, the models used a triggered subsystem to detect angular position. This method resulted in inefficiencies and a slower, less accurate solution. In addition, self-triggering subsystems are now illegal in Simulink.

### **Setting Block Sorting Priority on Virtual Subsystems**

In Simulink 4.0, it was an error to specify a priority on a virtual subsystem. In Simulink 4.1, you can specify priorities on virtual subsystems.

### **Using ~ in Filenames on UNIX**

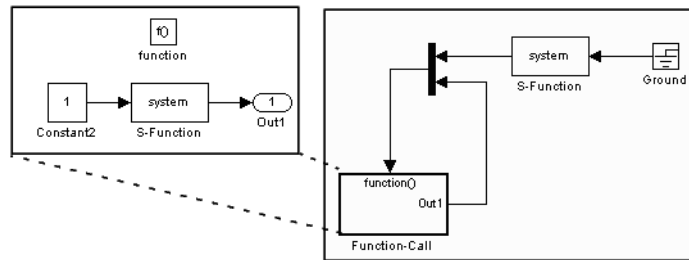
Now all filename fields in Simulink support the mapping of the ~ character in filenames. For example, in a To File block, you can specify ~/outdir/file.mat. On most systems, this will expand to /home/\$USER/outdir/file.mat.

### **Improved Warning About Slow Signals Feeding the Enable Port of an Enabled Subsystem Containing Fast Blocks**

In a multitasking environment, deterministic results cannot be guaranteed if a slow signal feeds the enable port of an enabled subsystem that contains fast blocks. In previous versions, Simulink did not issue a warning in some cases where this may occur.

### **Flagging Function-Call Subsystem Cycles**

In previous versions, Simulink allowed you to build models containing function-call-cycles, i.e., function-call subsystems that directly or indirectly call themselves.



Such models cannot be correctly simulated. Accordingly, Simulink now displays an error message when you attempt to run or update a diagram containing function-call cycles.

## Simulink Debugger

This section describes enhancements to the Simulink debugger.

### Enhancement to Sorted List Display

The Simulink debugger (`sldebug`) sorted list command, `slist`, now displays the names of the S-functions residing inside S-function blocks.

### Improved Messages in Accelerated Mode

The `trace`, `break`, `zcbreak`, `nanbreak`, and `minor` commands now indicate that they are disabled when in accelerator mode and you need to switch to normal mode to activate them. The spacing of several messages has been fixed so the text aligns correctly.

### Breakpoints on a Function-Call Subsystem

You can now put a break point on a function-call subsystem. Simulink breaks when the subsystem is executed. In Release 12, entering the `quit` command while at a breakpoint within a function-call subsystem wouldn't always quit the debugger. Now the `quit` command ends the debugging session once the initiating (calling) Stateflow chart or S-function finishes executing its time step.

## **Displaying and Probing Virtual Blocks**

The display and probe commands now work for virtual blocks.

## **Stepping Stateflow Charts**

You can now step execution of a model into a Stateflow chart.

## **Block Library**

This section describes enhancements to the Simulink block libraries.

### **Unified Pulse Generator**

This version merges the Discrete Pulse Generator block into the Pulse Generator block. The combined block has two modes: time-based and sample-based (discrete). Time-based mode varies the step size when a variable step solver is being used to ensure that simulation steps occur at pulse on/off transitions. When a fixed step solver is used, the time-based mode computes a fixed step size that ensures that a simulation step occurs at every pulse transition. The Pulse Generator block also outputs a pulse of any real data type in sample-based as well as time-based mode.

### **Control Flow Blocks**

Simulink 4.1 adds an If block and Switch Case block that can drive conditionally executed subsystems that contain instances of the new Action Port block. Action subsystems are similar to enabled subsystems, except that all blocks must run at the same rate as the If or Switch Case block.

This version also adds a For Iterator block and a While Iterator block. When placed in a subsystem, these blocks cause all of the blocks in the system to run multiple cycles during a time step. The block cycle in a For Iterator subsystem runs a specified number of times. The block cycle in a While Iterator subsystem runs until a specified condition is false. A user can limit execution of a While Iterator subsystem to a specified number of iterations to avoid infinite loops.

The new Assignment block allows a model to assign values to specified elements of a signal.



## Bus Creator

Simulink 4.1 adds a Bus Creator block that combines the output of multiple blocks into a single signal bus. A model can use the existing Signal Selector block to extract signals from the bus. The block's dialog box allows you to assign names to signals on the bus or allow the signals to inherit their names from their sources. When you double-click on a signal name in the block dialog, the source block is highlighted. There is no execution overhead in the use of bus creator/bus selector blocks.

## Sine Wave Block Enhancements

The Sine Wave block now supports a bias factor that eliminates the need to sum with a Constant block. The Sine Wave block also has a new computational mode. This mode (called sample-based) eliminates the dependence on absolute time.

## Enhanced Flip-Flop Blocks

Simulink Extras (`simulink_extras.mdl`) contains a set flip-flop blocks. These blocks now use the new triggered subsystem latching semantics. In addition, the S-R Flip-Flop block now models a physical NOR gate (i.e.,  $S=1, R=1 \Rightarrow Q=0, Q!=0$ , the undefined state).

## Additional Data Type Support

The Discrete-Time Integrator and Rounding Function blocks now handle single as well as double values. The Transport Delay, Unit Delay, Variable Transport Delay, Memory, Merge, and Outport blocks can specify nonzero initial conditions when operating on fixed-point signals.

## Simulink Block Library Reorganization

The Simulink Block Library contains a new Subsystems sublibrary. The new library contains most of the new control flow blocks as well as subsystem and subsystem-related blocks that used to reside in the Signals & Systems library. The subsystems in the new library each contain the minimum set of blocks needed to create a functioning subsystem, e.g., an input port and an output port.

## **Scope Enhancements**

The Scope block includes the following enhancements:

- A floating version of the Scope added to the Sinks block library
- Floating Scope saves the signals selected for display in the model file
- The Scope's toolbar buttons for toggling between floating/nonfloating mode, restoring saved axes, locking/unlocking axes, and displaying the **Signal Selector**

## Bug Fixes

This section lists fixes to bugs that occurred in the previous version of Simulink.

- Variable sample time S-functions

Simulink no longer crashes when an S-function with variable sample time is placed in an atomic subsystem.

- Bus selector detection of duplicated names

A bug related to the detection of a duplicated name in a bus that was feeding a Bus Selector block was fixed.

- Optimize block memory use

In Simulink 4.0, the Continuous and Discrete Transfer Function blocks and the Discrete Filter block used more memory than they needed to, particularly for the case of many poles. They now use an optimal amount of memory.

- Miscellaneous fixes to the model loader

Miscellaneous bug fixes have been performed on the model loader:

- The loader and saver now retain any comment lines (i.e., lines that begin with #) that are found at the top of the model file.
- The loader does not crash on Windows NT when file sizes are integer multiples of 4096.
- The loader does not hang on corrupt models in which blocks with duplicate names are found.

- Profiler fixes

The Simulink profiler now saves its files in the temporary directory. See the MATLAB command `tempdir`. The help was also updated.

- Chirp block fix

The Chirp block now sweeps through frequencies correctly from the initial frequency at the simulation start time to the target frequency at the target time.

- Function-call subsystem bug fixes

This version fixes several bugs related to the execution orders of function-call subsystems.

- **Sorting bug fix**

Previous versions incorrectly computed the direct feedthrough setting for nonvirtual subsystems in triggered/function-call subsystems. This resulted in incorrect execution (sorting) orders. Now all nonvirtual subsystems within triggered subsystems have their direct feedthrough (needs input) flags set for all input ports. This is needed because a nonvirtual subsystem with a triggered sample time executes both its output and update methods together within the context of the model's output method.

- **Fixed handling of grounded/unconnected inputs feeding certain blocks**

Simulink 4.0 incorrectly handled grounded or unconnected inputs to level-1 and level-2 S-functions requiring contiguous inputs and to some Matrix blocks. This has been fixed in Simulink 4.1.

## Upgrading from an Earlier Release

This section discusses upgrade issues in moving from Simulink 4.0 to Simulink 4.1.

See “Upgrading from an Earlier Release” on page 9-11 in the Simulink 4.0 Release Notes for upgrade issues involved in moving from Simulink 3.0 (Release 11.0) to Simulink 4.1.

### Running Simulink 4.1 Models in Simulink 4.0

Simulink 4.0 can run models created or saved by Simulink 4.1 as long as the models do not use features introduced in the new version, including new block types and block parameters. In particular, you should not attempt to use Simulink 4.0 to simulate or even open models that use the new Simulink control flow blocks. Opening such models cause Simulink 4.0 to crash.

### Simulink Block Library Reorganization

The Simulink Block Library contains a new Subsystems sublibrary. The new library contains most of the new control flow blocks as well as subsystem and subsystem-related blocks that used to reside in the Signals & Systems library. The subsystems in the new library each contain the minimum set of blocks needed to create a functioning subsystem, e.g., an input port and an output port.

### Direct Feedthrough Compensation Deprecated

If an S-function needs the current value of its input to compute its output, it must set its direct feedthrough flag to true. Previously, if a direct feedthrough S-function failed to do this, Simulink tried to provide a valid signal to the S-function’s `mdlOutput` (M-file `flag=3`) or `mdlGetTimeOfNextVarHit` (M-file `flag=4`) methods. This special compensation mode for S-functions was flawed. For this reason, the current version deprecates the mode, though making it available as an option. In this version, by default, if an S-function sets its direct feedthrough flag to false during initialization, Simulink sets the S-function’s input signal to NULL (or a NaN signal for M-file S-functions) during the `mdlOutput` or `mdlGetTimeOfNextVarHit` methods. Thus, in this version, models with S-function(s) may produce segmentation violations.

See `matlabroot/simulink/src/sfuntmpl_directfeed.txt` for more information.

## **S-Functions Sorted Like Built-In Blocks**

When sorting blocks, Simulink now treats S-function blocks the way it treats built-in blocks. This means that S-functions now work correctly in nonvirtual subsystems when there is a direct feedback connection (in Simulink 4.0 and prior, this wasn't the case). It also means that models compile (update diagram) faster. As a side effect, the execution order for S-functions that incorrectly set the direct feedthrough flag differs from that used in previous versions of Simulink. Consequently, models that contain invalid S-functions may produce different answers in this version of Simulink.

## **Added Latched Triggered Subsystems**

Now triggered subsystems enable you to implement software triggering, hardware triggering, or a combination of the two. Software triggering is defined as

```
if (trigger_signal_edge_detected) {
    out(t) = f(in(t));
}
```

Hardware triggering is defined as

```
if (trigger_signal_edge_detected) {
    out(t) = f(in(t-h)); // h == last step size
}
```

Previous to this version, triggered subsystems provided software triggering and a form of hardware triggering when a cycle involving triggered subsystems existed. Now, you must explicitly specify whether or not you'd like software or hardware triggering. This is done by selecting 'Latch (buffer) input' on the Inport blocks in a triggered subsystem.

Each input port of a triggered subsystem configures whether or not the input should be latched. A latched input provides the hardware-triggering semantics for that input port. Type `sl_subsys_semantics` at the MATLAB prompt for more information.

## Self-Triggering Subsystems Are No Longer Allowed

Before this version, you could define the output of a triggered subsystem to directly feed back into the trigger port of the subsystem (with potentially other additive signals). This resulted in an implicit delay. Now you must explicitly define the delay by inserting a memory block.

## Improved Invalid Model Configuration Diagnostics

This version of Simulink does a better job of detecting and flagging invalid modeling constructs in Simulink models. Consequently models that ran in previous versions of Simulink (sometimes producing incorrect results) may not run in the current release. The changes include:

- Direct feedthrough compensation no longer occurs by default for S-functions (see “Direct Feedthrough Compensation Deprecated” on page 8-13).
- S-functions are now sorted like built-in blocks (see “S-Functions Sorted Like Built-In Blocks” on page 8-14).
- Simulink no longer inserts implicit latches in triggered subsystems that directly or indirectly trigger themselves (see “Self-Triggering Subsystems Are No Longer Allowed” on page 8-15, above). Instead it signals an error when it detects a triggered subsystem loop with unlatched inputs. To avoid the error, you must select the **Latch** option on the triggered subsystem’s input ports.
- Simulink now signals an error when it detects invalid configurations of function-call subsystems. See the Subsystem Examples block in the Subsystems library for examples of illegal modeling constructs involving function-call subsystems. You can disable this diagnostic by setting the Invalid FcnCall Connection parameter on the **Diagnostics** pane of the **Simulation Parameters** dialog box to none or warning.





# Simulink 4.0 Release Notes

---

## New Features

This section introduces the new features and enhancements added in Simulink 4.0 since Simulink 3.0 (Release 11.0).

This section about new Simulink features is organized into the following subsections:

- “Simulink Editor” on page 9-2
- “Modeling Enhancements” on page 9-5
- “Simulink Debugger” on page 9-6
- “Block Library” on page 9-6
- “SB2SL” on page 9-9

### Simulink Editor

This section describes enhancements to the Simulink Editor.

#### Preferences

The Simulink **Preferences** dialog box allows you to specify default settings for many options (see "Setting Simulink Preferences" in Using Simulink).

#### Text Alignment

Simulink 4.0 allows you to choose various alignments for annotation text. To choose an alignment for an annotation, select the annotation and then select **Text Alignment** from the editor menubar or context (right-click) menu (see "Annotations" in Using Simulink).

#### UNIX Context Menus

The UNIX version of Simulink 4.0 now has context menus for block diagrams. Click the right button on your mouse to display the menu.

#### Library Link Enhancements

Simulink 4.0 optionally displays an arrow in each block that represents a library link in a model. Simulink 4.0 also allows you to modify a link in a

model and propagate the changes back to the library (see "Modifying a Linked Subsystem" in *Using Simulink*).

---

**Note** Simulink displays "Parameterized Link" on the parameter dialog box of a masked subsystem whose parameters differ from the library reference block to which the masked subsystem is linked. This feature, which is not documented in *Using Simulink*, allows you to determine quickly whether a library link differs from its reference.

---

## Find Dialog Box

The **Find** dialog box enables you to search Simulink models and Stateflow charts for objects that satisfy specified search criteria. You can use the dialog box to find annotations, blocks, signals, states, state transitions, etc. To invoke the **Find** dialog, select **Find** from the Simulink **Edit** menu (see "Searching for Objects" in *Using Simulink*).

## Model Browser

The Model Browser's toolbar includes the following new buttons:

- Show Library Links  
Shows library links as nodes in the browser tree.
- Look Under Masks  
Shows the contents of masked blocks as nodes in the browser tree.

## Single Window Mode

Simulink now provides two modes for opening subsystems. In multiwindow mode, Simulink opens each subsystem in a new window. In single-window mode, Simulink closes the parent and opens the subsystem (see "Window Reuse" in *Using Simulink*).

## Keyboard Navigation

Simulink 4.0 provides the following new keyboard shortcuts.

Key	Action
<b>Tab</b>	Selects the next block in the block diagram.
<b>Shift+Tab</b>	Selects the previous block in the block diagram.
<b>Ctrl+Tab</b>	Cycles between the browser tree pane and the diagram pane when the model browser is enabled.
<b>Enter</b>	Opens the currently selected subsystem.
<b>Esc</b>	Opens the parent of the current subsystem.

### Enhanced Library Browser

The Library Browser incorporates the following new features:

- Blocks no longer appear as browser tree nodes. Instead, they appear as icons in the preview pane.
- The preview pane has moved from beneath the library tree pane to beside the tree pane. You can create instances of blocks displayed in the preview pane by dragging them from the preview pane and dropping them in a model.
- Splitter bars now divide the browser's panes, allowing the panes to be independently resized.
- Double-clicking a block's icon opens the block's parameter dialog box with all fields disabled. This allows you to inspect, but not modify, a library block's parameters.
- Double-clicking a library block opens the library in the preview pane.
- You can now insert a block in the topmost model on your screen by right-clicking the block in the preview pane and selecting **Insert in...** from the context menu that appears. If no model is open or the topmost model is a locked library, the Library Browser offers to create a model in which to insert the block.

- The browser now contains a menu with **File**, **Edit**, and **Help** options.
- The block help text pane has moved from the bottom of the Library Browser to the top.
- Selecting **Find** from the Library Browser's **Edit** menu displays a modeless **Find** dialog box.
- The browser's search feature is much faster and supports regular expressions.

## Help Menus

Simulink 4.0 adds a Help menu to the menu bar on model and library windows. The help item on a block context menu displays a help page for the block. The help item on the model context menu displays the first page of the Using Simulink book.

## Modeling Enhancements

### Hierarchical Variable Scoping

This release extends the ability of Simulink to resolve references to variables in masked subsystems. Previously Simulink could resolve references only to variables in a block's local workspace. With this release, Simulink will resolve references to variables located anywhere within the workspace hierarchy containing the block (see "The Mask Workspace" in *Using Simulink*).

---

**Note** In some cases, hierarchical scoping will cause some models to behave differently in the current release than in previous releases of Simulink.

---

### Matrix Signals

Many Simulink blocks can now accept or output matrix signals. A matrix signal is a two-dimensional array of signal elements represented by a matrix. Each matrix element represents the value of the corresponding signal element at the current time step. In addition to matrix signals, Simulink also supports scalar (dimensionless) signals and vector signals (one-dimensional arrays of signals). Simulink can optionally thicken (select **Wide Lines** from the

**Format** menu) and display the dimensions of lines (select **Line Dimensions** from the **Format** menu) that carry vector or matrix signals. When you select the **Line Dimensions** option, Simulink displays a label of the form  $[r \times c]$  above a matrix signal line, where  $r$  is the number of rows and  $c$  is the number of columns. For example, the label  $[2 \times 3]$  indicates that the line carries a two-row by three-column matrix signal.

You can use Simulink source blocks, such as a Sine Wave or a Constant block, to generate matrix signals. For example, to create a time-invariant matrix signal, insert a Constant block in your model and set its Constant Value parameter to any MATLAB expression that evaluates to a matrix, e.g.,  $[1 \ 2; \ 3 \ 4]$ , that represents the desired signal. See "Working with Signals" in Using Simulink for more information.

### **Simulink Data Objects**

Simulink data objects allow a model to capture user-defined information about parameters and signals, such as minimum and maximum values, units, and so on (see "Working with Data Objects" in Using Simulink).

### **Block Execution Order**

Simulink now optionally displays the execution order of each block on the model's block diagram (see "Displaying Block Execution Order" in Using Simulink).

### **Simulink Debugger**

This section describes enhancements to the Simulink debugger.

### **GUI Debugger Interface**

Simulink 4.0 introduces a graphical user interface (GUI) for the Simulink Debugger. For more information, see "Simulink Debugger" in the online help for Simulink (see "Simulink Debugger" in Using Simulink).

### **Block Library**

This section describes enhancements to the Simulink block libraries.

## Product Block

The Product block now supports both element-by-element and matrix multiplication and inversion of inputs. The block's parameter dialog includes a new **Multiplication** parameter that allows you to specify whether the block should multiply or invert inputs element-by-element or matrix-by-matrix.

## Gain Block

The Gain block now supports matrix as well as element-wise multiplication of the input signal by a gain factor. Both input signals and gain factors can be matrices. The block's parameter dialog includes a new **Multiplication** parameter that allows you to choose the following options:

- $K \cdot u$  (element-wise product)
- $K * u$  (matrix product with the gain as the left operand)
- $u * K$  (matrix product with the gain as the right operand)

## Math Function Block

The Math Function block adds two new matrix-specific functions: transpose and Hermitian. The first function outputs the transpose of the input matrix. The second function outputs the complex conjugate transpose (Hermitian) of the input matrix.

## Reshape Block

Simulink 4.0 introduces the Reshape block, which changes the dimensionality of its input signals, based on an **Output dimensionality** parameter that you specify. For example, the block can change an n-element vector to a 1-by-N or N-by-1 matrix signal and vice versa. You can find the Reshape block in the Simulink Signals & Systems library.

## Multiplexing Matrix Signals

The Simulink Mux, Demux, and Bus Selector blocks have been enhanced to support multiplexing of matrix signals.

### **Function Call Iteration Parameter**

Simulink 4.0 adds a **Number of iterations** parameter to the Function Call Generator block. This parameter allows you to specify the number of times the target block is called per time step.

### **Probing Signal Dimensionality**

The Probe block now optionally outputs the dimensionality of the signal connected to its input.

### **Configurable Subsystem**

The Configurable Subsystem block has been reimplemented to make it easier to use. The configurable subsystem block now has a **Blocks** menu that allows you to choose which block the subsystem represents. To display the menu, select the configurable subsystem and then **Blocks** from the Simulink editor's **Edit** or context (right click) menu.

### **Look-Up Table Blocks**

This release provides four new Look-Up Table (LUT) blocks.

- Direct Look-Up Table (n-D)
- Look-Up Table (n-D)
- Prelook-Up Index Search
- Interpolation (n-D) Using PreLook-Up

The blocks reside in the Simulink Functions and Tables block library.

### **Polynomial Block**

The Polynomial block outputs a polynomial function of its input. The block resides in the Simulink Functions and Tables block library.

### **Signal Specification**

The Signal Specification block allows you to specify the attributes that the input signal must satisfy. If the input signal does not meet the specification, the block generates an error.



## ADA S-Functions

Simulink now supports S-functions coded in ADA. See "Creating Ada S-Functions" in *Writing S-Functions* for more information.

## Bitwise Logical Operator Block

The Bitwise Operator block is a new block that logically masks, inverts, or shifts the bits of an unsigned integer signal. See the online Simulink documentation for details.

## Atomic Subsystems

Simulink 4.0 allows you to designate subsystems as *atomic* as opposed to *virtual*. An *atomic subsystem* is a true subsystem. When simulating a model, Simulink executes all blocks contained by an atomic subsystem block before executing the next block of the containing model (or atomic subsystem).

By declaring a subsystem atomic, you guarantee that Simulink completes execution of the subsystem before executing any other blocks at the same level in the model hierarchy. See "Atomic Subsystems" in *Using Simulink* for more information.

---

**Note** Conditionally executed subsystems are inherently atomic. Simulink does not allow you to specify them as atomic or virtual.

---

## SB2SL

### SB2SL Extends Code Generation Support

SB2SL, which is included as part of Simulink, allows you to translate SystemBuild SuperBlocks to Simulink models.

For Release 12, SB2SL 2.1 has been enhanced to provide more complete support for use with the Real-Time Workshop. If you use the Real-Time Workshop 4.0 to generate code for models you have converted from SystemBuild to Simulink (using SB2SL), then code is generated for most translated blocks in the model.

The blocks that do *not* support code generation through the Real-Time Workshop 4.0 are:

- ConditionBlock
- Decoder
- Encoder
- GainScheduler
- Interp Table (Archive library)
- ShiftRegister

---

**Note** SB2SL 2.1 also includes a number of important bug fixes.

---

## Upgrading from an Earlier Release

This section describes the upgrade issues involved in moving from Simulink 3.0 (Release 11.0) to Simulink 4.0.

### Port Name Property

In previous releases, the name property of ports and lines referred to the label of the line connected to the port. In the current release, a port's name property refers to the port's (and line's) name, which, in the current release, can differ from the line's label. If you need to get the line's label, invoke

```
get_param(p, 'label')
```

where `p` is the handle of the port.