

Simulink® Release Notes



MATLAB® & SIMULINK®



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Simulink[®] Release Notes

© COPYRIGHT 2000–2022 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

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

Patents

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

Simulink Editor	1-2
Simulink Fundamentals course	1-2
Learn tab provides access to in-product trainings	1-2
Drop-down menu for Simulink Editor pane interactions	1-3
Compact Boundaries for Simscape Blocks Improve Model Layout and Readability	1-4
Highlight reduced nonvirtual blocks	1-4
Add State Reader, State Writer, and Parameter Writer blocks from Simulink toolstrip	1-5
Related block highlighting for Variant Connector and Parameter Writer blocks	1-8
Port toolbar menu customizations to custom tab in toolstrip	1-10
View and trace callbacks	1-10
Use modelfinder to find Simulink models	1-10
Unified breakpoint list in the Simulink Editor for debugging	1-11
Toolstrip-based UI for Model Advisor	1-11
Functionality being removed or changed	1-12
 Simulation Analysis and Performance	 1-14
Local solver	1-14
Zero-Crossing detection using Fixed-Step Solver	1-14
Change visualization for a subplot more easily in the Simulation Data Inspector	1-14
XY, sparklines, and map visualization enhancements	1-14
Interactive comparison report enhancements	1-16
External debugger integration for debugging custom C/C++ code	1-16
Parameter Combinations in Multiple Simulations panel	1-17
Functionality being removed or changed	1-17
 Component-Based Modeling	 1-18
Mask Editor streamlines masking workflow for Parameter Constraint, Port Constraint, and Parameter Promotion	1-18
Specify port constraints on masked block	1-23
Bus Editor enhancements	1-23
Message Triggered Subsystem and Message Polling Subsystem blocks: Process messages by executing subsystem when message is available	1-24
Reinitialize ports and Reinitialize Function block: Reinitialize referenced model or subsystem states during simulation	1-25
Function ports and port-scoped Simulink Function blocks: Call and define functions across peer models	1-26
Code reuse support in code generation	1-26

Use the new match filter with find_system to find both active and inactive blocks in a model	1-26
Find variant control variables used by variant parameters	1-26
Compile code conditionally for all variant parameter values with different dimensions	1-27
Reuse, control representation, and import variant parameter values in generated code using Simulink.Parameter variables	1-28
Use enumerated values to improve code readability of variant blocks and variant parameters with startup activation time	1-29
Mask parameter enhancements	1-30
Retrieve list of options and evaluated value of a parameter	1-30
Constraint Manager integrated with Mask Editor	1-30
Control Subsystem Reference programmatically	1-32
Variable-size signals in array of buses	1-33
Designation of export-function model	1-33
Redesigned Simulation Target Pane of Model Configuration Parameters	1-33
Interactively update legacy code in a System object	1-35
Half-precision data type support for System objects	1-36
MATLAB class object is now a non-tunable Parameter	1-36
String support for messages in MATLAB System block	1-36
Enhancements to co-simulation numerical compensation user interface ..	1-36
Multi-instance Simulink Function blocks	1-37
New example of parsing NMEA GPS Text Messages Using Simulink and Python.	1-37
New example of modeling a distributed wireless tire pressure monitoring system	1-37
Functionality being removed or changed	1-37
Project and File Management	1-38
Projects: Reduce test runtime in continuous integration workflows using the dependency cache	1-38
Project API: Export top-level and referenced project to previous version using Simulink.exportToVersion	1-38
Dependency Analyzer: Save dependency graph as image	1-38
Projects Examples: Determine optimal order for resolving conflicts	1-38
Text Comparison: Save results as HTML report	1-38
Folder Comparison: Compare folders in MATLAB Online	1-38
Functionality being removed or changed	1-39
Data Management	1-40
Import Simulink Design Verifier sldvData data into Signal Editor user interface	1-40
Signal Editor user interface tabular improvements	1-40
Root Import Mapper updates	1-40
Relax data consistency checking across model hierarchies	1-40
Model argument enhancements for Simulink.LookupTable and Simulink.Breakpoint objects	1-41
Block Enhancements	1-42
C Function block can interface with C++ classes	1-42
Enhanced Lookup Table Editor	1-42
Lookup Table blocks updated for better sharing and reuse	1-43

Configure Dashboard Scope plots more easily using subplot menu	1-43
New Matrix Operations blocks	1-44
New Logic and Bit Operations library blocks	1-44
Trigonometric Function block updates	1-44
New Message Triggered Subsystem and Message Polling Subsystem blocks	1-45
New Reinitialize Function block and reinitialize ports for Model and Subsystem blocks	1-45
Port-scoped Simulink Function blocks	1-45
New Function Element and Function Element Call blocks	1-45
New capabilities for Parameter Writer block	1-45
C Function block can access custom code global ND arrays	1-46
C Caller and C Function blocks support 64-bit integer types as int64 and uint64	1-46
C Function block can specify code to initialize conditions when subsystem states reset	1-46
Selector and Assignment blocks support out-of-range index checking in accelerator and rapid accelerator modes	1-46
For Each Subsystem block accepts variable-size signals	1-46
Specify origin for value bar and needle on customizable dashboard blocks	1-46
Resize and reposition foreground image for customizable dashboard blocks	1-47
Change scale direction for customizable dashboard blocks	1-48
Lamp block in Customizable Blocks library supports specifying state values as ranges	1-48
View and edit each state for customizable Lamp block in design mode . .	1-48
Change opacity of state colors for customizable Lamp block	1-49
From Spreadsheet block Range Selection tool updates	1-49
Unit Conversion block now accepts multidimensional signals	1-49
Simulink.ImageType data type for image signals	1-50
Discrete PID Controller Blocks: Improve execution time of generated code	1-51
Functionality being removed or changed	1-51
Connection to Hardware	1-54
Support added for 64-bit Android applications	1-54
Support added to integrate algorithm to existing Android application . . .	1-54
Better visual display and auto rotate applications on Android devices . . .	1-54
Support added for Arduino compatible ESP32-WROOM boards	1-54
Hardware support extended for real-time execution of generated code on Arduino boards	1-55
Select serial communication port for external and PIL modes on Arduino boards	1-55
Support extended for executing multitasking models for Arduino Nano 33 BLE Sense board	1-55
Control pan and tilt motions using PCA9685-based PWM driver	1-55
Adjust and set video and camera parameters using V4L2 Video Capture block in Raspberry Pi	1-56
Improved build and deployment speeds for Simulink Support Package for Raspberry Pi Hardware	1-56
Manually set XCP-based polling time for Simulink model running on Raspberry Pi	1-56
Higher data acquisition rates using Connected I/O with Arduino boards	1-56

Read distance to target object using VL53L0X sensor	1-56
Read linear acceleration, angular velocity, magnetic field, and temperature using ICM-20948 sensor	1-56
Read eCO2 and eTVOC using CCS811 sensor	1-57
Read linear acceleration using ADXL34x family of accelerometers	1-57
MATLAB Function Blocks	1-58
Cell Array Type for Non-Tunable Parameters	1-58
Variable Size property applies only to output variables	1-58
Symbols pane replaces Ports and Data Manager	1-58
New Code Generation Readiness Tool: View more information and navigate through readiness results more easily	1-59
coder.ScreenerInfo object: Access code generation readiness information programmatically	1-60
Code generation for more MATLAB functions	1-60
Code generation for more toolbox functions	1-60
Modeling Guidelines	1-61
High-Integrity Systems Modeling Checks: Improve quality and compliance with guidelines	1-61
S-Functions	1-62
Programmatically configure S-functions using the S-Function Builder block	1-62
Functionality being removed or changed	1-62

R2021b

Simulink Editor	2-2
Create custom Simulink Toolstrip tabs	2-2
Miniature map provides context when viewing part of a block diagram	2-2
Get names of built-in Simulink Toolstrip actions and icons	2-3
Import, export, reset, and reorder favorite commands in Simulink	2-3
View how to pan and zoom in the Simulink Editor by pressing and holding the space bar	2-3
Signal Tracing Command Line API	2-4
Improved search experience in quick insertion of block	2-4
Dock Diagnostic Viewer to a model	2-4
Move Simulink Editor tab to a new window	2-5
Quick action search menu provides default results	2-6
Visually identify line or port as primary selection	2-6
Display of port label information and automatic editing of port labels on Subsystem blocks	2-7
Display related blocks in open diagrams or new tabs	2-7
Propagate tag change from Goto block to related From and Goto Tag Visibility blocks	2-7
Enhancements to right-click context menu	2-7

Simulation Analysis and Performance	2-9
Run multiple simulations for different scenarios from the Simulink Editor	2-9
Solver Switching with Fast Restart	2-9
Configure comparison constraints using the Simulation Data Inspector UI	2-9
Copy, cut, and paste plotted signal selection from one run to another in the Simulation Data Inspector	2-10
View multidimensional signal data on an array plot in the Simulation Data Inspector	2-11
Export video data to an MP4 file using the Simulation Data Inspector ..	2-13
Easily swap x and y data on XY plots in the Simulation Data Inspector ..	2-14
Simulation Data Inspector UI enhancements	2-15
Interactive comparison report enhancements	2-17
Simulation Data Inspector snapshot supports capturing a map visualization in a MATLAB figure	2-17
SIMD hardware acceleration support for Just-in-Time (JIT) simulation ..	2-17
Simulation enhancements for referenced models with multiple normal-mode instances	2-17
Extract data from multiple elements of a Simulink.SimulationData.Dataset object and put the data into timetables using the extractTimetable function	2-18
Use runtime callbacks to set inport data and outport data at the root level of a Simulink model	2-18
Functionality being removed or changed	2-18
Component-Based Modeling	2-20
'startup' Variant Activation Time: Change active choice before simulation or in the generated code during model initialization	2-20
Convert input and output ports to bus ports at root-level of model	2-20
Support for code generation workflows in Variant Reducer	2-21
Use new field in CompiledVariantInfo parameter to identify execution status of a block	2-21
Prevent accidental edits to subsystem reference using the read-only property	2-21
Open test harness automatically while running it on a subsystem	2-22
Schedule Editor: Use a global scoped event to combine multiple instances of an event	2-22
Function-Call Subsystem: Call periodic function-call subsystem intermittently using Function-Call Generator block	2-23
Guard header files in generated code for S-Function blocks in variant models	2-24
Allow message root-level In Bus Element block to provide receive queue properties	2-24
Variant Parameters: Enhanced simulation speed, improved code layout, and option to select active choice while running executable	2-24
Project and File Management	2-28
Model Comparison: Compare Simulink models in MATLAB Online	2-28
Projects: Collaborate using Projects in MATLAB Online	2-28
Text Comparison: Compare and merge text files with improved usability, appearance, and syntax highlighting	2-28
OPC format for MDL files	2-29

Project Preferences: Detect project-wide references when renaming Simulink buses and bus elements	2-30
Source Control: Work with files under Git source control in MATLAB Online	2-30
Functionality being removed or changed	2-30
Data Management	2-31
Specify signal properties based on application-specific value types such as wind velocity	2-31
Signal Editor user interface now accepts simple MATLAB expressions in tabular area	2-31
Signal Editor adds timeseries data by default	2-31
Simulink.io.FileType has new methods	2-32
New Simulink.io.ModelWorkspace class	2-32
Configure complexity for Simulink.Parameter objects	2-32
Define complex model arguments without specifying a default value	2-32
Import struct and enum types without typedef from custom C code	2-32
Block Enhancements	2-34
New Matrix Operations library	2-34
New button blocks in the Customizable Blocks library	2-34
New switch blocks in the Customizable Blocks library	2-34
New Rotary Switch block in the Customizable Blocks library	2-35
New Lamp block in the Customizable Blocks library	2-35
Upgrade Signal Builder blocks to Signal Editor blocks with Upgrade Advisor	2-35
Upgrade Advisor: Check output dimensions of MATLAB Function blocks	2-36
Lookup table viewer updates	2-36
Parameter Tunability Enhancements for Continuous Blocks	2-36
Signal Builder Import File dialog box updated	2-37
Visualize and edit n-dimensional table data in mask dialog control using lookup table editor	2-37
Control initial size of column in a custom table	2-38
Specify Row and AlignPrompts for containers in System objects blocks ..	2-38
Specify descriptions with In Bus Element and Out Bus Element blocks ..	2-39
String support for MATLAB System Block	2-39
Push Button block enhancements	2-39
FMU Import updates	2-39
Functionality being removed or changed	2-39
Connection to Hardware	2-42
Deploy customizable Gauge and Knob Dashboard blocks on Android device	2-42
Generate code from Simulink model to run directly on Android devices using PIL simulation	2-42
Read and write commands to TalkBack ID on ThingSpeak TalkBack for Android devices	2-42
Send and receive data using Bluetooth low energy protocol on Arduino boards	2-42
Create interactive display of Dashboard Push Button block on Arduino boards	2-42

Deploy Input Capture and SD Card File Read blocks on Arduino Nano 33 BLE Sense boards	2-42
Measure rotation of motor interfaced with rotary encoder on Arduino boards	2-43
Read and write from ThingSpeak data channels simultaneously on Arduino boards	2-43
Encode and decode data packets based on any messaging format for Arduino-based peripheral communication	2-43
SocketCAN support added for CAN Transmit and CAN Receive blocks on Raspberry Pi boards	2-43
Deploy customizable Gauge and Knob Dashboard blocks on Raspberry Pi boards	2-43
Read and write commands to TalkBack ID on ThingSpeak TalkBack on Raspberry Pi boards	2-43
MATLAB Function Blocks	2-44
MATLAB Function Block Editor in Simulink window	2-44
Output column vectors as two-dimensional or one-dimensional data in MATLAB Function blocks	2-45
Expanded code generation for tables and timetables	2-45
Code generation for more MATLAB functions	2-45
Code generation for more toolbox functions	2-45
MATLAB Function blocks support implicit expansion	2-46
Modeling Guidelines	2-47
Use modeling guidelines to verify compliance of your models	2-47
S-Functions	2-49
S-Function Builder: Support for array of buses, nested array of buses, and single-word fixdt data types	2-49

R2021a

Simulink Editor	3-2
Schedule Editor: View connections between Stateflow charts and corresponding partitions with Schedule Connectors	3-2
Find and replace text in Simulink	3-2
Add comments to blocks for collaboration	3-2
Improved experience in quick insertion of blocks	3-3
Pause on block diagram to view and edit block names	3-4
Select and interact with ports	3-5
Edit port labels on subsystem blocks	3-6
Updates to compatible port connection cues	3-8
Edit enumerated block parameters on block icons	3-8
Move multiple lines and blocks together	3-10
Enhancements to favorite commands in Simulink	3-11
Keyboard shortcuts for selecting ports and outermost elements	3-11
New context menu for closing Simulink Editor tabs	3-12

Simulation Analysis and Performance	3-13
Log data to the workspace and to a file using the Record block	3-13
Event Animation: Access event animation via the Simulink toolstrip	3-14
Event Inspection: Log and visualize events using the Sequence Viewer tool	3-14
Use hardware acceleration to improve simulation execution performance by leveraging SIMD instructions	3-15
Multithread co-simulation changes	3-16
For Each subsystem multicore execution	3-16
Canceling model compilation is now supported	3-16
Control maximum size of logged data	3-16
Log variable-size signals using signal logging	3-17
Data access callback supports nonvirtual buses	3-17
Sparklines visualization in the Simulation Data Inspector	3-17
Configure colors for plot area, grid, and tick labels on XY plots in the Simulation Data Inspector	3-19
Simulation Data Inspector user interface enhancements	3-20
Import ULG files into the Simulation Data Inspector	3-20
Programmatic interface for Simulation Data Inspector cursors	3-21
Programmatically specify Simulation Data Inspector time plot axis limits	3-21
Manage multiple panels using tabs	3-21
Generate C++ code for half-precision floating-point data types in Simulink	3-23
Run custom C/C++ code in a separate process during model simulation	3-23
Report Generation for Simulink Profiler	3-24
Discrete State Linearization in Referenced Models	3-24
Dynamic Switching of the Algebraic Loop Solver	3-24
Functionality being removed or changed	3-24
Component-Based Modeling	3-26
Simulink Code Importer: Import C code as reusable Simulink libraries ..	3-26
Bus ports support nonvirtual buses, arrays of buses, and case-insensitive searches	3-26
Messages: Use Message Merge block to combine multiple message lines into a single message line	3-27
Override, edit, and reset configuration parameters using configuration reference functions	3-28
Specify data mode for virtual bus elements at component interface	3-28
Get list of choices in Variant Subsystems using VariantChoices parameter	3-29
Promote mask icon of active variant to Variant Subsystem automatically	3-29
Reuse block parameters with different values by using variant parameters	3-30
Limit scope of variant control variables for Inline Variant blocks and Variant Connector blocks	3-31
Model reference simulation targets now support C++	3-31
Allow commenting out from and commenting through blocks in model, block, and mask initialization	3-32
Support for variant conditions on function-call signals through Variant Source and Variant Subsystem blocks	3-33

Transfer existing test harnesses to subsystem block diagram while converting subsystem to subsystem reference	3-33
Promote block parameters to system mask in subsystem reference	3-33
Replace continuous block with discrete versions inside Variant Subsystem	3-33
Support for variant condition propagation to root-level In Bus Element, Out Bus Element blocks and Simulink.Bus objects	3-33
Variant Configuration Analysis tool now supports the Variant Connector block	3-35
Functionality being removed or changed	3-35
Project and File Management	3-38
Model Comparison: Compare and merge models using rich highlighting	3-38
Model Comparison: Improved model comparison report	3-38
Project API: List all referenced projects of the current project	3-38
Project API: List impacted project files	3-39
Upgrade Advisor Check: Turn off file compression to optimize storage under source control	3-39
Dependency Analyzer: Find required add-ons	3-39
Functionality Being Removed or Changed	3-39
Data Management	3-40
Share data objects with Simulink block library users by using a linked data dictionary	3-40
Model argument enhancements	3-40
Signal Editor user interface changes	3-40
Register Simulink.io.FileType objects from a different name space	3-41
Block Enhancements	3-42
Data Stores: Access Data Store Memory elements dynamically via index input ports with Data Store Read/Write blocks	3-42
Highlighting to indicate related blocks	3-42
New Message Merge Block	3-43
New customizable Knob, Vertical Slider, and Horizontal Slider blocks	3-43
Choose from a set of icons or specify a custom icon for the Lamp block	3-44
Dashboard gauge blocks support foreground, background, and font color	3-44
Tune matrix and structure elements using dashboard blocks	3-44
Dashboard Scope block enhancements	3-45
Change dashboard block orientation and scale direction	3-45
From Workspace block supports loading data for array of bus	3-46
Sqrt and Dot Product blocks support half-precision data type	3-46
Trigonometric Function block disables input range checking for acos and asin functions	3-46
Trigonometric Function block supports lookup table algorithms for multiple functions	3-46
1-D Lookup Table block supports full precision interpolation algorithm for linear interpolation	3-46
Switch, MinMax, and Multiport Switch blocks have new output data type option	3-46
Math Function block reciprocal function can now use Newton-Raphson method	3-47

Tunability of Rate Limiter block parameters	3-47
Simulink.LookupTable and Simulink.Breakpoint classes are able to be subclassed	3-47
FMU block is now more customizable	3-47
Signal Editor block now supports array of buses	3-47
Signal Builder block menu and toolbar options to export data to Signal Editor block	3-47
MATLAB System Block now supports message input and output	3-48
New upgrade advisor check to integrate with sysobjupdate	3-48
C Function and C Caller blocks in For Each subsystem, with continuous time, and optimized in conditional input branch execution	3-48
Control inherited block output data type for half-precision	3-48
Tunable mask enumeration in generated code	3-49
Show or hide columns in custom table	3-49
Sparse Matrix Support for State-Space Block	3-49
Parameter Tunability Enhancements for Continuous Blocks	3-49
Functionality being removed or changed	3-51
Connection to Hardware	3-53
Deploy deep learning applications on Android devices	3-53
Create interactive display of Dashboard blocks on Android devices	3-53
Read proximity, gesture, ambient light, and color data using APDS9960 I2C sensor	3-53
Support added for LCD and TFT display using Display Configuration app	3-53
Profile real-time execution of generated code on Arduino hardware	3-53
Support added for Arduino Engineering Kit Rev2	3-53
Support added for Arduino Nano 33 BLE Sense board	3-54
Measure rotation of motor interfaced with rotary encoder	3-54
New functionalities added to Raspberry Pi Resource Monitor app	3-54
Support for MODBUS TCP/IP data communication protocol	3-54
Support for adding multiple SDL Display blocks to Simulink model deployed on Raspberry Pi	3-55
Trigger downstream function-call subsystem	3-55
MATLAB Function Blocks	3-56
Programmatically access MATLAB function reports	3-56
64-bit integer type support for parameters	3-56
Half-precision data type support for MATLAB Function blocks	3-56
Error reporting from MATLAB Function blocks in rapid accelerator simulation mode	3-56
Expanded code generation for categorical arrays	3-57
Expanded code generation for tables and timetables	3-57
Code generation for more MATLAB functions	3-58
Code generation for more toolbox functions	3-58
Modeling Guidelines	3-59
Use modeling guidelines to verify compliance of your models	3-59

Simulink Editor	4-2
Simulink Online: Use Simulink through your web browser	4-2
Enhancements to the Simulink quick access toolbar	4-2
Keyboard shortcuts to select and move objects in a diagram	4-2
Drag or paste text into Simulink Editor canvas to create annotations	4-3
Optionally resize blocks to fit parameter values	4-3
Align centers while resizing blocks and annotations	4-4
Use half and string data types in the mask datatypestr parameter	4-4
Display auto-complete options for mask parameter values while using set_param	4-4
Refer between containers in the same library	4-4
Improved performance for the add_block function	4-5
Function to check if Simulink is started	4-5
 Simulation Analysis and Performance	 4-6
FMU Export: MultiThreadCoSim renamed to MultithreadedSim	4-6
Streaming enhancements and triggers in the Simulation Data Inspector	4-6
Simulation Data Inspector enhancements for importing Microsoft Excel files	4-7
Create a custom data reader to import data into the Simulation Data Inspector	4-7
Select colors to use for comparison results in the Simulation Data Inspector	4-7
Simulation Data Inspector zoom enhancements	4-8
Specify a units system to use to display data in the Simulation Data Inspector	4-9
Simulink logging enhancements	4-9
Enhanced Simulation Data Inspector interactive comparison report	4-9
Read and write data using Simulation Data Inspector functions in deployed applications	4-10
Intel C and C++ toolchain support for Windows	4-10
Functionality Being Removed or Changed	4-10
 Component-Based Modeling	 4-12
Schedule Partitions to Execute Based on Events Broadcast from Stateflow in the Schedule Editor	4-12
Support for virtual buses as component interfaces in message-based communication	4-12
For Each subsystem: Call a utility from repeated executions using the Function Caller block	4-12
Receive protected models with their dependencies in a project	4-13
Restore and push edited and parameterized library blocks in the model using the improved Library Links Manager tool	4-13
Analyze all Variant choices using Variant activation time	4-14
Automatic variant switching for simulation and code generation workflows	4-15
Use true and false as conditions in Variant blocks	4-17

Suppress digital signature verification of protected model programmatically	4-17
Perform commonly used subsystem reference tasks using the toolstrip menu	4-18
Simulate subsystem block diagrams using test harness from the subsystem context menu	4-19
Schedule Editor Supports Partitioning with Variable-Step Solvers	4-20
Parameter CompiledVariantInfo	4-20
MatchFilter option with find_system	4-21
find_system enhancements with Variants	4-21
Variant Configuration Analysis Tool: Specify variant control values	4-22
Variant Manager: Improved handling of variant control variables defined in InitFcn	4-23
Variant Connector support in Variant Manager	4-23
Subsystem Reference support in Variant Reducer	4-23
Set simulation target language to build custom code	4-23
Enhanced support for different types of variables in Variant expressions	4-24
Limit scope of Variant control variables from mask and model workspace of Variant subsystems	4-24
Handle Variant-related modeling issues using new diagnostic	4-25
SIMD optimization for simulation targets	4-26
Disable fading effect of inactive blocks using Variant Fading option	4-26
Improved version control for Simulink models	4-27
Project and File Management	4-28
Automerge Tool: Customize source control tools to automerge models ..	4-28
Project API: Export to previous version using Simulink.exportToVersion	4-28
Dependency Analyzer: Export to archive, generate dependency report, and create a project from the dependency graph	4-28
Project API: Change project definition file type and preserve source control history	4-28
Model Templates: Edit Simulink Model Templates	4-28
Source Control: Improved workflow to set up Git source control	4-29
Text Comparison: Compare text files in MATLAB Online	4-29
Clear Simulink History Programmatically	4-29
Model File Preferences: Notify when opening a model in an unopened project	4-29
polyspacePackNGo Function : Generate and package Polyspace option files from a Simulink model	4-29
Save and close referenced subsystems and models in a model hierarchy	4-30
Simulink cache functions for querying info and unpacking contents	4-30
Functionality Being Removed or Changed	4-30
Data Management	4-32
Export signal data in custom formats from the Signal Editor	4-32
Default value for model arguments	4-32
Property dialog boxes for data objects use separate tabs for design and code generation properties	4-32
Functionality being removed or changed	4-32
Block Enhancements	4-34

Global variables in C Caller block	4-34
New PWM block	4-34
Property Inspector and toolstrip enhancements for custom dashboard blocks	4-35
Name property added for panels	4-35
Simulink toolstrip support for dashboard blocks	4-35
Add a foreground image to the Horizontal Gauge, Vertical Gauge, and Circular Gauge blocks	4-36
Background Color, Foreground Color, and Opacity properties added for several dashboard blocks	4-36
IC block supports bus signals	4-36
1-D, 2-D, and n-D Lookup Table blocks have new input ports	4-36
FMU block changes	4-36
Simulate models containing string blocks in rapid accelerator mode from the MATLAB command line	4-36
MATLAB System block supports data types with slope bias scaling	4-36
Simulink.LookupTable property dialog box updated	4-37
New SimStruct functions for string support	4-37
PID Controller Simulink Blocks: Option to provide output saturation limits as external inputs	4-37
Highlighting for related Goto and From blocks	4-38
Specify name of subsystem created by Simulink.BlockDiagram.createSubsystem	4-39
C Function block improvements	4-39
Limit the range of data to process in a For Each subsystem	4-40
Model blocks refresh automatically	4-40
Updated block parameter names for buses	4-40
New Variable Pulse Generator Block	4-40
Functionality Being Removed or Changed	4-41
Connection to Hardware	4-43
Communicate with Android devices in Normal mode simulation using Connected I/O	4-43
Create an interactive display of Dashboard blocks on Android devices ..	4-43
Measure frequency and duty cycle of an external signal	4-43
Measure linear acceleration, angular rate, and magnetic field using LSM9DS1 IMU sensor	4-44
Support for MODBUS RS485 serial data communication protocol	4-44
Support added for Nano 33 IoT board	4-44
Publish data to WebSocket server using Arduino hardware	4-44
Read data from WebSocket server using Arduino hardware	4-44
Obtain Arduino library locations	4-45
Support for I2C serial data communication	4-45
Support for parallel compilation of generated code and driver source code	4-45
Communicate with Raspberry Pi hardware in Normal mode simulation using Connected I/O	4-45
Base rate synchronization between ALSA Audio blocks and Simulink models	4-46
MATLAB Function Blocks	4-47
Access fields of a structure array in a MATLAB Function block	4-47
Expanded code generation for datetime and duration arrays	4-47
Expanded code generation for tables and timetables	4-48

Expanded code generation for categorical arrays	4-49
Code generation for more MATLAB functions	4-50
Code generation for more toolbox functions	4-50
Modeling Guidelines	4-55
High-Integrity System Modeling guidelines to verify compliance with DO, IEC, EN, and ISO safety standards	4-55
S-Functions	4-56
S-Function Builder changes	4-56

R2020a

Simulink Editor	5-2
Port enhancements	5-2
Port name propagation	5-2
Simplified line segment moving	5-3
Improved in-canvas text editing	5-3
Put the block name on any side of the block	5-3
Simulation Analysis and Performance	5-4
Save Simulation Data Inspector comparison results in a shareable, interactive HTML report	5-4
Display message payload as port value labels	5-4
Filter and scan errors and warnings during edit-time in diagnostic viewer.	5-5
View streamed data on a scrolling display in the Simulation Data Inspector	5-5
Simulation Data Inspector programmatic interface enhancements	5-6
View and inspect multiple series on an XY visualization in the Simulation Data Inspector	5-7
Shade signal regions using cursors in the Simulation Data Inspector	5-8
Configure programmatic Simulation Data Inspector comparisons and save comparison results	5-10
Enhanced Simulation Data Inspector support for parallel computing	5-11
Enhanced task-based sorting algorithm for determining block execution order	5-12
Navigate from blocks to tasks in the Execution Order viewer	5-12
Updated Simulink Profiler	5-14
ode1be implicit fixed step solver	5-15
Implicit solver support for odeN solver	5-16
Project tab Tool-Coupling FMU option name change	5-17
Intel C and C++ toolchain support for Windows	5-17
Functionality being removed or changed	5-18
Component-Based Modeling	5-21
Simulink Units: Create and load custom units	5-21

Simulate referenced models with unsaved changes in accelerator mode	5-21
Simulate protected models from a previous release in accelerator mode	5-21
find_mdrefs option to exclude top model in results	5-21
Messages support fixed-point data type	5-21
Event actions support calling a scoped Simulink Function block	5-21
Verify digital signatures on protected models	5-22
Improved workflow of variants and masking	5-22
Convert Configurable Subsystem blocks to Variant Subsystem blocks programmatically	5-22
Improved performance of Variant Manager: Control Variables usage changed to on-demand from automatic	5-23
Improved code generation with Variant Subsystem	5-24
C++ code generation with Inline Variants	5-24
Support for C++ class interfaces for explicitly partitioned models	5-24
Subsystem reference enhancements	5-24
Search, sort, and group items in the forwarding table dialog	5-24
add_block command uses forwarding table to add appropriate block ...	5-25
Schedule Editor API: Programmatically perform operations on the schedules of the model	5-25
Functionality being removed or changed	5-25
Project and File Management	5-26
Dependency Analyzer: Unified App for analyzing project dependencies and model hierarchies	5-26
Project Checks: Run all project checks programmatically	5-27
Model Comparison: Create PDF model comparison reports	5-27
Model Comparison: Color legend for comparison reports	5-27
Project API: Get latest Git revision programmatically	5-28
Simulink Model Files: Determine update release number used to save SLX models	5-28
Simulink Template: Specify template group in the template dialog box ..	5-28
Case-sensitive model and library names	5-28
Data Management	5-29
Signal Editor user interface updates	5-29
Import and export custom format variables using Simulink.io.FileType objects	5-29
createInputDataset function allows signal format specification	5-29
Specify value of Simulink.LookupTable argument by using simple numeric value	5-29
64-bit integer data types	5-30
External Data node for model in Model Explorer	5-30
Simulink.Bus object enhancements	5-31
Improved usability for specifying model arguments and data type block parameters	5-31
Name change for properties that allow access to base workspace	5-32
View referenced dictionary hierarchy by using the Dependency Analyzer	5-32
Block Enhancements	5-33
C Function Block: Integrate and call C code from the model	5-33

Delay and Unit Delay blocks support strings	5-33
Delay block now accepts bus input for initial condition	5-33
Label enhancements for In Bus Element and Out Bus Element blocks . . .	5-33
Bus Assignment block input signals stay connected to corresponding ports when reordered	5-34
Programmatically refresh Model blocks	5-34
New string blocks for pattern matching	5-34
String Find and String Length blocks now support output data type	5-35
String blocks and rapid accelerator mode	5-35
Direct Lookup Table (n-D) block has new parameter	5-35
Simulink toolstrip Lookup Table tab	5-35
Signal Editor block supports timetable objects	5-35
Step block now supports single data type	5-35
PID Controller Simulink blocks: Support for asynchronous interrupts with external targets	5-35
Search Simulink blocks using internationalized keywords	5-36
Dashboard library Display block supports vector, 2-D matrix, and complex signals, and has additional format options	5-36
New custom Horizontal Gauge and Vertical Gauge blocks in the Dashboard library	5-37
Root-level Inport blocks support interpolation for fixed-point data types and loading matlab.io.datastore.sd datastore data	5-38
From Workspace block supports loading timetable data	5-38
Rate Transition block deterministic mode support for concurrent execution	5-39
For Each Subsystem block supports test-pointed signals	5-39
Expanded DWork buffer length	5-39
Hit Crossing block outputs function-call events	5-39
Functionality being removed or changed	5-39
Connection to Hardware	5-40
Create display of gauges on Android devices	5-40
Improved build and deployment speeds of Simulink Support Package for Arduino Hardware	5-40
Set PWM frequency on Arduino hardware	5-40
Capture data using Arduino sensor blocks	5-41
Support added for Raspberry Pi 4B model board	5-41
Use Connected I/O to communicate with Raspberry Pi in Normal mode simulation	5-41
Read data from web socket server using Raspberry Pi hardware	5-41
Pixy2 Vision Sensor Block	5-41
Enhancements to Parrot Flight Control Interface	5-42
MATLAB Function Blocks	5-43
Call scoped Simulink functions from MATLAB Function block	5-43
64-bit integer type support for MATLAB Function block	5-43
Enhanced error reporting from MATLAB Function blocks	5-43
Code generation for datetime arrays	5-43
Code generation for timetables	5-44
Code generation for more MATLAB functions	5-44
Code generation for more toolbox functions	5-44
Functionality being removed or changed	5-48
Modeling Guidelines	5-49

Support for MathWorks Advisory Board (MAB) v5.0 modeling guidelines	5-49
High-Integrity System Modeling guidelines to verify compliance with DO, IEC, EN, and ISO safety standards	5-56

R2019b

Simulink Editor	6-2
Simulink Toolstrip: Access and discover Simulink capabilities when you need them	6-2
Find actions to perform using a Simulink Editor search tool	6-24
Animation for messages and Simulink functions	6-25
Invoke text editing for a selection using a keyboard shortcut	6-25
Update line routing affected by scripts	6-25
Create conditionally executed subsystems from empty selections	6-26
Align edges while resizing blocks and annotations	6-26
Simulation Analysis and Performance	6-27
Simulation Manager: Configure simulation visualization and save the simulation sessions	6-27
Dashboard Panels: Test and debug your models using a dashboard that is accessible throughout the model hierarchy	6-28
XY Plots: Visualize relationships between signals in the Simulation Data Inspector	6-30
Connect Dashboard blocks in your model during simulation	6-33
Connecting a Dashboard block to a signal does not mark the signal for logging	6-33
Save Simulink.SimulationData.Dataset objects to a MAT-file you can use in earlier releases	6-33
Constant signal logging enhancement	6-34
Programmatically plot data for any simulation output object in the Simulation Data Inspector	6-34
Save selected Simulink.sdi.Run objects to an MLDATX-file	6-35
Additional styling controls for time plots in the Simulation Data Inspector	6-35
Simulation Data Inspector signal property enhancements	6-36
Additional information in Simulink.sdi.DiffRunResult and Simulink.sdi.DiffSignalResult objects	6-36
Hide Simulation Data Inspector Getting Started pop-up	6-37
Data logging and signal viewer block support for export function models	6-37
FMU Import block model reference support	6-37
Component-Based Modeling	6-38
Subsystem Reference: Save subsystem contents in a file that you can reference in a model	6-38
Messages: Model and generate C++ code for software compositions with message-based communication	6-39
Simulink Cache: Supports code generation and Stateflow	6-40

Variant Configuration Analysis Tool: Enhanced functionalities for better usability	6-40
Execution Order Viewer	6-40
Simulink Units Updates	6-40
Global data on parameters are guarded	6-40
Functionality being removed or changed	6-41
Project and File Management	6-42
Model Comparison: Compare and three-way merge MATLAB Function blocks	6-42
Projects: Delete project definition files	6-42
Compare Git Branches: Show differences and save copies	6-42
Simulink Models Files: Unset compression to reduce repository size	6-42
Data Management	6-43
Simulink data dictionary version handling	6-43
Root Inport Mapper Updates	6-43
createInputDataset function supports bus elements	6-43
Signal Editor user interface updates	6-43
Edit stored integer values for fixed-point data objects	6-43
Create variable from numeric expression for block parameters	6-44
Access model workspace and data dictionary from model canvas	6-44
Model Data Editor enhancements	6-44
Breakpoint set type must match specification for Simulink.LookupTable objects	6-44
Convert variables to Simulink.Parameter objects from the context menu	6-45
Functions Simulink.findIntEnumType and Simulink.clearIntEnumType Applied to User-Defined Enumeration Classes	6-45
Block Enhancements	6-46
Blockset Designer: Streamline the process of blockset authoring and file management	6-46
New Messages & Events library	6-46
Sequence Viewer Block: Visualize function calls, message communication, and Stateflow state changes and event activity over time	6-46
n-D Lookup Table, 1-D Lookup Table, and 2-D Lookup Table blocks support modified Akima cubic Hermite interpolation	6-47
Enhanced enumerated data type support for the Rotary Switch block and Combo Box block	6-47
Configure Display block appearance	6-48
Math Function block pow function changes	6-48
Additional data type and parameter support with mask parameter constraints	6-49
Evaluate Combo box mask parameter during simulation	6-49
Create a hierarchical list of data on mask dialog box	6-49
Store mask images in the SLX file	6-49
New interface for managing signal viewers and generators	6-49
Playback control behavior changed for scopes in referenced models	6-50
Bus Selector and Bus Assignment block usability improvements	6-50
Control inherited sample time for System objects included in a MATLAB System block	6-51
Delay block enhanced support for symbolic dimensions	6-51

String blocks updates	6-51
Configure C Caller block properties using command line API	6-52
Select behavior for undefined function handling in C Caller block	6-52
New Entity Transport Delay Block	6-52
First Order Hold block	6-52
Simulink block data type support table	6-52
Expanded block I/O buffer length	6-53
MinMax Running Resettable update	6-53
Memory block update	6-53
Priority set on If Action Subsystem block ignored	6-53
Functionality being removed or changed	6-53
Connection to Hardware	6-55
New blocks for communicating with Android devices	6-55
Support for MATLAB Arduino IO functions inside a MATLAB function block	6-55
Support for Arduino MKR CAN Shield: Run Simulink models on Arduino boards	6-55
Support for Arduino MKR WIFI 1010 Hardware: Run Simulink models on Arduino boards	6-55
Support for Arduino MKR Zero: Run Simulink models on Arduino boards	6-56
Read data from files in SD Card	6-56
New blocks to communicate with CAN bus module connected to Raspberry Pi	6-56
Support for Grabber and Cannon modules of Parrot minidrone	6-56
Hardware Toolstrip: Build, deploy, and start Simulink models on supported hardware	6-56
MATLAB Function Blocks	6-59
Simplify interfacing with row-major data by using row-major array layout	6-59
Simulink string support in MATLAB Function block	6-59
Configure MATLAB Function block properties using command line APIs	6-59
Code generation for tables	6-59
Code generation for duration arrays	6-59
Code generation for hexadecimal and binary literals	6-60
Code generation for more MATLAB functions	6-60
Code generation for more toolbox functions	6-61
Modeling Guidelines	6-65
High-Integrity System Modeling guidelines to verify compliance with DO, IEC, EN, and ISO safety standards	6-65

Simulation Analysis and Performance	7-2
--	------------

Replay simulations in the Simulation Data Inspector	7-2
Signal Editor user interface supports signal draw and insert	7-3
Timing Legend: Interactively identify different rates and their types through highlighting	7-3
odeN Solver (Nonadaptive): Quickly simulate systems with zero-crossing detection	7-3
FMU Export: Package a model as an FMU for co-simulation	7-6
Locate source of Simulink diagnostics	7-6
Simulation Data Inspector comparison enhancements	7-6
Show or hide plot grid in the Simulation Data Inspector	7-7
Export data to Microsoft Excel from the Simulation Data Inspector	7-7
Convert signal data types in the Simulation Data Inspector	7-8
Process 1-D and 2-D matrix data during simulation	7-8
Bus signal logging enhancements	7-8
FMU Import Block: Rapid accelerator support	7-9
Save and restore a simulation snapshot using the renamed ModelOperatingPoint object	7-9
Return simulation outputs as a single simulation output by default for new Simulink models	7-9
Zero-Crossing Explorer for Solver Profiler: Visualize and interactively explore zero-crossing events.	7-9
Unit test constraint for comparing time series data	7-11
Simulink Editor	7-12
Simulink Onramp: Self-paced, interactive tutorial for getting started with Simulink	7-12
Arrange Automatically: Improve layout and appearance using a menu command	7-13
Smart Guides: Evenly space blocks and place blocks in suggested zones	7-14
Quick Insert Keywords: Add library blocks to model using keyword	7-14
Collapsible Panes: Save and restore settings	7-14
Annotations: Specify a transparent background and connect to areas ...	7-14
Image Annotations: Drag images to canvas	7-15
Model Window Size Increased: 32-bit coordinates supported	7-15
No-Selection Action Bar: Draw to create area or subsystem	7-15
Simulink.BlockDiagram.expandSubsystem: Expand subsystems without creating areas	7-15
add_block: Copy block or subsystem from library without library link ...	7-15
Simulink.BlockDiagram.routeLine: Route existing lines of a model	7-16
Functions added for displaying errors and warnings	7-16
Simulink Toolstrip Tech Preview replaces menus and toolbars in the Simulink Desktop	7-17
Component-Based Modeling	7-18
Schedule Editor: Explicitly schedule the execution of your model components	7-18
Model Reference: Open referenced models in the same window as their parent models	7-18
Protected Models: Use models protected in previous releases for normal mode simulation	7-19
Buses: Simplify model interfaces with bus element port blocks	7-19
Buses: Specify multirate buses and signal attributes at bus ports	7-20

Variant Configuration Analysis Tool: Difference tool to compare variant configurations	7-21
Control usage of arithmetic operations in Variant conditions	7-21
Global data on signals and states are guarded	7-22
Code generation for multi-instance referenced models with variant Simulink Functions	7-23
Propagate variant conditions with no active variant controls in Variant Subsystem	7-23
Rate transition improvements in models configured for concurrent execution	7-24
Functionality being removed or changed	7-25
Project and File Management	7-27
Projects: Use projects in MATLAB	7-27
Model Comparison: Get more accurate Simulink block matches when comparing models	7-27
Project Files: Find and add files to the project in bulk	7-27
Project Dependency Analysis: Navigate required files with improved highlighting and filtering	7-27
Project Startup: Control Simulink starting and running customizations when opening projects	7-28
Project Archives: Share projects in a single file and check for required products	7-28
Functionality being removed or changed	7-28
Data Management	7-29
Simplified workflow for specifying values for model arguments in a nested model reference hierarchy	7-29
Componentized management of global data in a model reference hierarchy	7-29
Root Import Mapper Updates	7-29
Signal Editor user interface supports improved fixed-point data type proposals	7-30
Override parameter values in a configuration reference	7-30
Simplified configurations section in the Model Explorer	7-30
View Simulink data dictionary version information	7-31
Enable model to access data directly from base workspace and data dictionary	7-31
Enhanced user interface for edit-time checks	7-31
Create missing data stores during edit and compile time	7-31
Edit-time fix provided for renamed variables in a data dictionary	7-32
Enhanced name conflict management in Model Explorer	7-32
Support for uint64 and int64 as built-in Simulink data types	7-32
New data type propagation rules for Sum, Gain, and Product blocks	7-32
Support for bus element ports in Model Data Editor	7-33
Block Enhancements	7-34
Call Simulink Functions from MATLAB System block	7-34
Improved results with Quick Block Insert	7-34
Effective identification of leaf and container blocks	7-34
Mask Parameter Constraints: Create custom error messages	7-35
set_param command on mask parameter executes callback code	7-35
Mask parameter callbacks execute in temporary workspace	7-36

Modification of mask parameters is restricted from callback	7-36
Simulink Mask: Enhance usability with additional parameters	7-36
Trigonometric Function and Sqrt block supports symbolic dimensions . .	7-38
Direct Lookup Table (n-D) block updates	7-38
Lookup table object updates	7-38
Display block updates	7-39
FMU Block Updates	7-39
Programmatically add and configure Dashboard blocks	7-39
Display Dashboard block enhancements	7-39
Enhanced enumerated data type support for the Radio Button block	7-40
Parameter Writer block	7-41
Input/Output Scope for C Caller Blocks	7-42
Signal Editor block updates	7-42
signalBuilderToSignalEditor function updates	7-42
Signal Builder block update	7-42
Connection to Hardware	7-43
Perform external mode simulation on Arduino using XCP over serial, TCP/IP, and Wi-Fi connections	7-43
Perform external mode simulation on Raspberry Pi using XCP over TCP/IP	7-43
Trigger a downstream function-call subsystem using Arduino External Interrupt block	7-43
Measure and control motor parameters using Arduino MKR motor carrier blocks	7-43
MATLAB Function Blocks	7-44
Sparse matrix inputs for more functions in MATLAB Function blocks . . .	7-44
Expanded support for the vecdim input argument in MATLAB Function blocks	7-44
MATLAB Function block support for class properties with string scalar initial values	7-44
Behavior change for character vector or string scalar input to ismethod in MATLAB Function blocks	7-45
Code generation for more MATLAB functions	7-45
Code generation for more 5G Toolbox functions	7-46
Code generation for more Audio Toolbox functions	7-46
Code generation for more Automated Driving Toolbox functions	7-47
Code generation for more Communications Toolbox functions	7-47
Code generation for more Computer Vision Toolbox functions	7-48
Code generation for more DSP System Toolbox functions	7-49
Code generation for more Image Processing Toolbox functions	7-49
Code generation for more Phased Array System Toolbox functions	7-49
Code generation for more Robotics System Toolbox functions	7-49
Code generation for more Sensor Fusion and Tracking Toolbox	7-50
Code Generation for SerDes Toolbox Functions	7-51
Code generation for more Signal Processing Toolbox functions	7-51
Code generation for more Statistics and Machine Learning Toolbox functions	7-52
Code generation for more Wavelet Toolbox functions	7-52
Code generation for more WLAN Toolbox functions	7-52
S-Functions	7-54

Thread-safe error status handling	7-54
SimState S-Function APIs renamed	7-54
Modeling Guidelines	7-55
High-Integrity System Modeling guidelines to verify compliance with DO, IEC, EN, and ISO safety standards	7-55

R2018b

Simulation Analysis and Performance	8-2
Batch Simulations: Offload execution of simulations to run in the background by using batchsim	8-2
Execution Domain Specification: Improve your solver performance and code by specifying the domains of subsystems within your model	8-2
Custom Gauge Block: Create a gauge with a fully customizable appearance	8-2
Maps in the Simulation Data Inspector: View map data that is synchronized with signal data in the Simulation Data Inspector	8-2
Descriptor State-Space Block: Author linear time invariant DAE equations directly	8-5
FMU Export: Package a model as an FMU for co-simulation	8-5
Easily connect multiple Dashboard blocks to signals in your model with connect mode	8-6
Include notes and analysis in visualizations using a text editor in the Simulation Data Inspector	8-7
Iterate model design with the Simulation Data Inspector archive	8-9
Rename signals in the Simulation Data Inspector	8-10
Import Excel files into the Simulation Data Inspector	8-11
Inspect data in the Simulation Data Inspector by viewing the window full screen and maximizing subplots	8-11
Dynamic Solver Switching: Adaptively switch solver during run-time to improve simulation performance	8-12
Step through the referenced model with Simulink Debugger	8-12
Diagnostic Viewer: Register custom compiler to report errors and warning	8-12
Functionality being removed or changed	8-12
Simulink Editor	8-15
Automatic Port Creation: Create a port by clicking or dragging the block outline	8-15
Predictive Quick Insert: Connect a recommended block to an existing block in your model, sorted by frequency of use	8-15
Edit on Block Icon: Modify block parameters without opening a dialog box	8-16
Better context awareness for SimEvents blocks	8-16
Quick Block Insert: Dynamically sort quick block insert search results ..	8-16
Collapsible Panels: Manage UI elements by hiding panels	8-16
Equations in Annotations: Insert LaTeX and MathML equations in your annotations	8-16

Group Block Rotation: Rotate or flip selected blocks as a group	8-17
Component-Based Modeling	8-19
Buses: Create buses from Simscape physical connection lines and retain arrays of buses and nonvirtual buses within virtual buses	8-19
Simulink Variants: Develop versatile variant models with improved support for buses, IRT blocks, and functions	8-19
Simulink.Bus.createObject supports more object types	8-20
Mask Constraints: Create and share mask parameter constraints	8-20
Mask Editor: Extend mask usability with additional parameters and options	8-20
Label Mode: Use Label mode to select Variant controls for modeling Variant blocks	8-20
Variant Reducer: Enhanced functionalities for better usability	8-21
Simulink Variants: Empty Simulink Variant objects not allowed in Simulink Variants	8-21
Compact Code Packaging for Variants: Generate lightweight code for Simulink Variants	8-21
Functionality being removed or changed	8-21
Project and File Management	8-23
Project References: Explore the full project reference hierarchy and associated files directly from your Simulink Project	8-23
Project-Wide Bus Renaming: Automatically update all bus references across Simulink Project when you rename a bus or bus element	8-23
Model Comparison: Use custom filters to simplify and focus model comparisons	8-23
Generated Code Traceability: View code files generated from models in the dependency graph	8-23
Git Stashes: Store uncommitted changes for later use	8-23
Source Control: Simplified workflow for creating projects from Git or SVN	8-23
Data Management	8-24
Block Parameter Autocomplete: Improve the speed and accuracy of block parameter editing by selecting from suggested variable or function names as you type	8-24
Undefined Variable Detection: Get more information about missing variables while editing your model	8-24
Model Workspace API: Read and modify variable properties in the model workspace	8-24
Model Data Editor Enhancements: Perform actions on referenced data objects from a context menu	8-25
Model Explorer Enhancements: Navigate your model using simplified interface	8-25
Root Inport Mapper updates	8-26
Signal Editor user interface updates	8-26
Underspecified dimension diagnostics	8-26
Propagated Signals: Display all propagated signal labels	8-26
Enumerated Data Types: Use functions to find, inspect, and delete enumeration classes	8-27
MATLAB variables: View dimensions and complexity in Simulink tools	8-27

Simulink Data Dictionary Version Handling: Use and export model data saved in previous releases with models created in later releases	8-27
Legacy Code Tool: Integrate code with 64-bit variables	8-27
Configuration Parameters: View referenced configuration with a single click	8-27
Model Data Editor: View referenced variables on the Parameters tab . . .	8-28
Log Datastore Memory blocks while your model is configured for multi-tasking	8-28
Load string data with the From Workspace block	8-28
Block Enhancements	8-29
Row-Major Array Layout: Simplify integration with external C/C++ code for Lookup Table and other blocks	8-29
C Caller Block: Call external C functions directly from the model	8-29
Signal Editor Block Enhancement: Set active scenarios and signals by index	8-30
String block updates	8-30
Prelookup and Interpolation Using Prelookup blocks support virtual buses	8-30
Direct Lookup Table block supports symbolic dimension	8-30
Interpolation Using Prelookup block selection port support of enumerated data	8-30
State owner blocks: Read and write block state for Second-Order Integrator and Output blocks when running initialize, reset, and terminate functions	8-30
Add signals to Floating Scope blocks and Signal Viewers with connect mode	8-31
S-Function Builder: Option to configure the language to generate the S-function	8-31
Faster Standalone Code for Linear Algebra: Generate code that takes advantage of your own target-specific BLAS library	8-32
Connection to Hardware	8-34
Log data in MAT-files on an Android device	8-34
Retrieve data from an Android device using the Internet of Things	8-34
Measure data using Arduino sensor blocks	8-34
Publish and subscribe to messages using Raspberry Pi MQTT blocks . . .	8-34
Support added for Raspberry Pi 3 Model B+ board	8-34
MATLAB Function Blocks	8-35
Faster Standalone Code for Linear Algebra: Generate code that takes advantage of your own target-specific BLAS library	8-35
Sparse Matrices: Generate code for the backslash operation on sparse matrices	8-35
Class Support: Use objects in more functions for code generation	8-35
Code generation for more MATLAB functions	8-35
Code generation for rng 'shuffle' option	8-35
Code generation for more Communications Toolbox functions	8-36
Fuzzy Logic Toolbox Code Generation: Generate code to load and evaluate Fuzzy Inference Systems	8-36
External C code integration	8-36
Functionality being removed or changed	8-36

S-Functions	8-37
Check your S-function for undocumented APIs	8-37
Row-major code generation support for S-functions	8-37
Modeling Guidelines	8-38
High-Integrity Modeling Guidelines: Verify compliance with DO, IEC, EN, and ISO safety standards	8-38

R2018a

Simulation Analysis and Performance	9-2
Simulation Data Inspector in the Live Editor: Add, view, and edit Simulation Data Inspector plots directly in the Live Editor	9-2
Simulation Pacing: Run simulations at wall clock speed or other specified pace for improved visualization	9-2
Controllable Sample Time: Adjust the time interval for block execution during the simulation, for example, to vary the sampling frequency . . .	9-3
Subsystem Domain Specification: Improve your solver performance and code by specifying the domains of subsystems within your model	9-3
Diagnostic Handling: Directly fix errors and warnings from diagnostic messages, such as algebraic loops, as well as manage suppressed diagnostics	9-3
Simulink Execution Engine	9-4
State Logging Compression: Reduce the size of your logged states data by choosing to use a real-time states sampling algorithm	9-4
Adaptive Absolute Tolerance: Automatically fine-tune absolute tolerance values with an initial value to improve both performance and accuracy during simulation	9-4
Log parameters and variables connected to Dashboard blocks	9-4
View string data in the Simulation Data Inspector	9-5
Access logged signal data during simulation	9-5
Import CSV and MDF files into the Simulation Data Inspector	9-6
Control automatic import of parallel simulation data to the Simulation Data Inspector	9-6
Improved Simulation Data Inspector support for signal processing	9-6
Simulation Data Inspector comparison view signal grouping	9-7
Convert signal units in the Simulation Data Inspector	9-7
Log Simscape data directly to the Simulation Data Inspector	9-8
Simulink Editor	9-9
Predictive Quick Insert: Connect a recommended block to an existing block in your model	9-9
Sample Time Legend: Interactively identify rates in your model through highlighting with the new Sample Time Legend	9-9
Port Connection Cues: Highlight compatible ports as you draw a signal line	9-9
Model Notes: Add text or web links to describe any model layer	9-9

Attribute Mismatch Detection: Find mismatches between components related to data type, complexity, and dimensions while editing your model	9-10
Content Preview: Display contents on new hierarchical elements by default	9-10
Automatic Layout: Rearrange model elements from the command line	9-10
Property Inspector displays sample command-line interface for block parameters	9-10
Find Model Elements: Interact with model while search executes	9-10
Signal Tracing: Show all possible paths	9-11
Search Models Programmatically: New commands in Simulink API	9-11
Component-Based Modeling	9-12
Variant Signals in Buses: Use variant conditions to dynamically add and remove bus elements	9-12
Mask Constraints: Ensure that mask parameters are within desired specifications without writing additional code	9-12
Diagnostic Suppression Review: Manage the activity of all suppressed diagnostics in a single interface	9-12
set_param function on mask parameter reports edit-time and run-time errors	9-12
Simulink Mask: Enhance mask usability using additional parameters and options	9-13
Variant Systems: Structures as variant control is supported	9-13
Library Links: Better architecture to control multilevel variants in library links	9-13
Variant Subsystem Block: Use conditionally executed systems within the Variant Subsystem block	9-14
Enhanced Variant Manager functionalities for better usability	9-14
Project and File Management	9-15
Impact Analysis: Easily identify dependencies and impacts of block or requirement changes in your Simulink project	9-15
Simulink Project Upgrade: Streamline upgrade to a new release by checking compatibilities of MATLAB code files in your project	9-15
SVN Source Control Workflows: Browse and compare change sets, and manage repository locks	9-15
Data Dictionary Workflows: Search inside data dictionaries and check for unsaved changes	9-15
Simulink cache files (slxc) and slprj folder generated more frequently	9-15
Data Management	9-17
Model Design Data: Use multiple sources for your design data by allowing Data Dictionary to access the base workspace	9-17
Undefined Variable Detection: Get instant notifications about missing variables while editing your model	9-17
Create and edit signals using MATLAB expressions	9-18
Import Signal Builder block signals and properties to Signal Editor block	9-18
Parameter Dependencies: Explicitly model relationships between dependent and independent variables	9-18
Model Data Editor Enhancements: Configure data type and other design attributes of block parameters	9-18

Simplified Specification of Initial Values: Use consistently named programmatic parameters	9-19
Configuration Parameters: View parameters that were on the Optimization panes on the Simulation Target and new Math and Data Types panes	9-19
Load matlab.io.datastore.SimulationDatastore objects directly using root level Inport blocks	9-19
Flexible SimState Restore	9-20
Block Enhancements	9-21
Hit Crossing Messages: Create Messages for debugging and send them to Stateflow and SimEvents to indicate events in Simulink for hybrid system modeling	9-21
Strings: Design embedded systems with native support of strings	9-21
Lookup Tables: Add enumerated inputs and breakpoints and use Lagrange interpolation for more powerful lookup tables	9-22
Direct Lookup n-D block Column option	9-23
Interpolation Using Prelookup block visual affordance for subtables	9-24
Signal Editor block supports fast restart	9-24
Co-simulation data exchange	9-24
Multicore co-simulation	9-25
FMU block changes	9-25
Discrete FIR Filter block supports custom state attributes for better customization and efficiency of generated code	9-25
Run-time diagnostics	9-25
Unit System Configuration update	9-25
Controllable sample time support for MATLAB System block	9-25
System object 'Discrete Periodic' sample time renamed to 'Discrete'	9-26
Nested protected models	9-26
Log data from Scope block as timetable	9-26
Connection to Hardware	9-27
Automatically run Simulink model on Raspberry Pi after each restart	9-27
Better audio quality using Simulink Support Package for Raspberry Pi Hardware	9-27
Support added for Raspberry Pi Zero W board	9-27
Support added for ThingSpeak Read block on Raspberry Pi	9-27
Measure data using LEGO EV3 Compatible NXT Sensor blocks	9-27
Measure data using LEGO EV3 blocks	9-28
Flexibility in Android support package installation	9-28
Accept data from Android keyboard	9-28
Measure data using Android sensor blocks	9-28
Communicate with Arduino during Normal mode simulation using Simulink IO	9-28
MATLAB Function Blocks	9-29
Redesigned MATLAB Function Report: View information in new user interface with improved navigation	9-29
Sparse Matrices: Enable more efficient computation in a MATLAB Function block	9-29
Cell Array Support: Import cell arrays into a MATLAB Function block by using coder.load and load	9-30

Delete Method: Call clean-up code automatically when handle classes are deleted	9-30
Statistics and Machine Learning Toolbox Code Generation: Generate code for distance calculation on vectors and matrices, and for prediction by using k-nearest neighbor with Kd-tree search and nontree ensemble models	9-30
Code generation for MATLAB sprintf function	9-31
Code generation for MATLAB sort function options	9-31
Code generation for more MATLAB functions	9-31
Code generation for more Audio Toolbox functions	9-31
Code generation for more Communications System Toolbox functions ...	9-32
Code generation for more DSP System Toolbox functions	9-32
S-Functions	9-33
S-functions support interleaved complex data	9-33
simstruc.h is restructured to include only documented SimStruct APIs ..	9-33
C-MEX S-functions built without using the mex command	9-33

R2017b

Simulation Analysis and Performance	10-2
Simulation Manager: Monitor, inspect, and visualize simulation progress and results	10-2
Enhanced Simulation Data Inspector UI and API	10-3
Logging Improvements for the Simulation Data Inspector	10-3
New Simulation Control and Visualization Blocks in the Dashboard Library	10-3
Solver Profiler supports fixed step solvers	10-4
Configure Solver Profiler from the command line	10-4
Reduce memory usage when linearizing large models	10-5
Simulink Cache: Generate Simulink cache files for top models in accelerator mode	10-5
slsfncgctrl function not supported to report diagnostics programmatically	10-5
Suppress immaterial diagnostic warnings and errors from specific blocks	10-5
Comprehensive run-time diagnostics for wrapping and saturating overflows from Stateflow and MATLAB Function blocks	10-5
Simulink Editor	10-6
Hidden Block Names: Improve model appearance by hiding default block names	10-6
Signal Tracing: Highlight and navigate a signal from its source to a destination	10-6
Autoroute Lines: Select blocks and multiple lines for improved line routing	10-6
Create Subsystem: Intuitive port placement and naming	10-6
Double-Click Action: Add blocks or annotations on the canvas	10-7
Searching Models: Cancel find and use search history across models ...	10-7

Simulink API: Column vector and matrix values are not supported in some functions	10-8
Component-Based Modeling	10-9
Schedulable Components: Explicitly schedule models for simulation and adaptation to your software environment	10-9
Tables in Masks: Present and sort your mask parameters in a searchable table	10-9
Simulink Variants: Create more customizable variant models by using improved Variant Subsystem and variant condition propagation capabilities	10-9
Variant Reducer Enhancements	10-11
Variant Systems: Convert Model blocks with model variants to Variant Subsystem blocks	10-11
Signal attribute display at load and edit time	10-12
Bus Signals: Learn about buses for simpler diagrams, with productivity tips	10-12
slreplace_mux function not supported	10-13
Tune Slider and Dial using variables or tune variables from Slider and Dial	10-13
Update library links programmatically	10-14
Project and File Management	10-15
Simulink Project Upgrade: Update all models and library blocks used in your project to the latest release	10-15
Automatic Project Creation: Easily turn a folder into a project and manage your files, data, and environment in one place	10-15
Model Compare and Merge: Identify differences between model elements, Stateflow charts, and MATLAB Function blocks	10-15
Automate Comparison Reports: Compare models and generate reports programmatically	10-15
Programmatic Project Setup: Manage startup and shutdown files	10-16
Missing Product Identification: Find and install required products for project templates	10-16
Project Shortcuts Toolstrip: Simplified workflow for setting up shortcuts for frequent tasks	10-16
Compare Git Branches: Show differences from parent files and save copies	10-16
Data Management	10-17
Model Data Editor: Easily view, filter, group, and edit more data used by a model including signals, states, and referenced variables	10-17
Signal Editor: Create and edit input signals that can be organized for multiple simulations	10-17
Defining Missing Variables: Identify and easily fix missing, deleted, or renamed variables	10-18
Lookup table class supports even spacing	10-18
Lookup table object structures	10-18
Root Inport Mapper update	10-18
Signal Loading: Load timetable data into Simulink models	10-18
Signal Loading: Incrementally stream DatasetRef data from MAT-file to root-level Inport blocks	10-19

Data Store Memory Logging: Stream to MAT-File or Simulation Data Inspector	10-20
Variable Editor Access: Launch the Variable Editor from more convenient locations	10-21
Data Navigation Enhancements: More easily create workspace variables from block parameters	10-21
Button for Lookup Table Objects: Create Simulink.LookupTable and Simulink.Breakpoint objects in the Model Explorer more quickly ...	10-22
Shared Local Data Store: Share data between instances of a reusable model	10-22
Simplified Configuration of Block States: Configure block states by using uniformly named programmatic parameters	10-22
Tunable Parameters: Tune parameters in model workspace	10-23
Configuration Parameters Dialog Box: View your model configuration parameters in unified dialog box with search capability	10-23
Legacy Code Tool: Convert N-D matrix column-major format to row-major format	10-24
Block Enhancements	10-25
Scoped Simulink Functions: Create Simulink Functions that can now cross model boundaries for reusable software components	10-25
Tunability of n-D Lookup Table and Prelookup blocks	10-25
Import Functional Mockup Unit (FMU) model for simulation	10-25
Signal Builder block updates	10-25
Assignment and Selector blocks support enumerated data type	10-26
Display, create, edit, and switch scenarios with Signal Editor block ...	10-26
Faster code generation and rapid accelerator mode startup for MATLAB System block	10-26
Specify discrete single-rate sample time for MATLAB System block ...	10-26
S-Function Builder: New Start and Terminate methods and option to add PWorks	10-27
From Spreadsheet block updates	10-27
Step back support in the Floating Scope block	10-27
Improvements to interactive legend in scope blocks	10-27
New model advisor check and command-line API to analyze S-functions for possible problems and improvements	10-27
MinMax block now supports boolean	10-28
Math Function block log and log10 changes	10-28
Sum block changes	10-28
MATLAB Function block change	10-29
Algebraic constraint block updates	10-29
Connection to Hardware	10-30
Simulink Support Package for Parrot Minidrones: Deploy flight control algorithms on PARROT minidrones	10-30
Support for Arduino MKR1000 Hardware: Run Simulink models on Arduino MKR1000 boards	10-30
Blocks added to Android support package	10-30
Support for DSP System Toolbox Array Plot block on Apple iOS and Android apps	10-30
MATLAB Function Blocks	10-31

FFT Library Calls for FFT Functions: Speed up simulation of FFT functions in a MATLAB Function block	10-31
Strings: Represent text as a string scalar in a MATLAB Function block	10-31
Cell Arrays and Classes in Structures: Use structures that contain cell arrays and classes in a MATLAB Function block	10-31
Class Folders for Classes in a MATLAB Function block: Use MATLAB classes defined by using multiple files	10-32
Property Validation: Use classes that restrict property values in a MATLAB Function block	10-32
Value Classes in a MATLAB Function Block: Pass objects of value classes to and from extrinsic functions	10-32
Statistics and Machine Learning Toolbox Code Generation: Generate C code for prediction by using discriminant analysis, k-nearest neighbor, SVM regression, regression tree ensemble, and Gaussian process regression models	10-32
Code generation for more MATLAB functions	10-33
Code generation for more Audio Toolbox System objects	10-35
Code generation for more DSP System Toolbox System objects	10-35
Code generation for more Phased Array System Toolbox System objects and functions	10-35
Code generation for more Robotics System Toolbox functions	10-35
S-Functions	10-36
New continuous time C S-Function examples	10-36
Variable Discrete Sample Time	10-36

R2017a

Simulation Analysis and Performance	11-2
Parallel Simulations: Directly run multiple parallel simulations from the parsim command	11-2
Simulink Cache: Get simulation results faster by using shared model artifacts	11-2
Inport File Streaming: Stream large input signals from MAT-files without loading the data into memory	11-2
Unified Streaming and Logging: Mark a signal once to stream it to the Simulation Data Inspector and log it to the MATLAB workspace	11-3
Simulation Data Inspector: Run simulation comparisons with a new UI, time tolerance support, and faster performance	11-4
Dashboard Block Connection Indicators: Easily determine which block in your model is associated with a given Dashboard block	11-4
Signal Tracing: Incrementally trace and highlight paths for debugging	11-5
Root-Level Inport Blocks: Create dataset for root-level Inport blocks	11-5
Simulation Logging Data and Metadata: Access simulation data and information more directly	11-5
Rapid Accelerator mode: Rapid Accelerator now supports S-functions without source code	11-6

Signal Editor: Create and edit input signals that can be organized for multiple simulations	11-6
Improved simulation performance when stepping back is enabled	11-6
Simulink Diagnostic Management: Control which simulation and fixed-point diagnostic warnings you receive from specific blocks, including model reference	11-7
Simulink Editor	11-9
Automatic Port Creation: Add inports and outports to blocks when routing signals	11-9
Model Block Masking: Customize the parameter dialog boxes for referenced models	11-9
Quick Find: Use a modifier to search for model properties in search box	11-9
Format Painter: Copy formatting between model elements	11-9
Refresh Library Browser: Update quick insert list with custom libraries using menu command	11-9
Functionality Being Removed or Changed	11-10
Optimize rendering during mask icon drawing	11-10
Component-Based Modeling	11-11
Reduced Bus Wiring: Quickly group signals as buses and automatically create bus element ports for fewer signal lines between and within subsystems	11-11
Bus and Vector Mixtures Not Supported	11-12
Inline Variants: Single-Input/Single-Output Inline Variant blocks support zero active variant control	11-12
Searchable, sortable tables for parameterizing reusable models with model arguments	11-13
Project and File Management	11-14
Simulink Project Upgrade: Easily update all the models in your Simulink Project to the latest release	11-14
Missing Product Identification: Fix models with unresolved library links and unknown block types by finding and installing missing products	11-14
Git Pull: Fetch and merge in one step	11-15
Project Creation API: Set up projects programmatically, including shortcuts and referenced projects	11-15
Referenced Project Change Management: Compare components with checkpoints	11-15
Source Control Toolstrip: Simplified workflow for working with source control	11-15
Custom Task Tool: Improved interface for managing custom tasks and creating reports from results	11-16
Git Remote Repositories: Connect existing project to a remote repository	11-16
Start Page Example Search: Find featured examples	11-16
Model Templates: Simplified workflow for exporting models to templates	11-16
Start Page Favorites: Easily get back to your favorite models and projects	11-16
Project Componentization: Include referenced projects in templates for sharing components	11-17

bdIsDirty Function: Programmatically check whether models contain unsaved changes	11-17
listRequiredFiles Function: Get project file dependencies programmatically	11-17
Data Management	11-18
Simulation Data: Easily access simulation output data in the MATLAB Variable Editor and MATLAB Command Window	11-18
Management of workspace variables and mask parameters from block parameters	11-18
Association of root-level Outport block with Simulink.Signal object	11-18
Initial State: Log and load initial states using Dataset format	11-18
Root Inport Mapper Tool Updates	11-19
Legacy Code Tool StartFcnSpec and InitializeConditionsFcnSpec accept outputs as arguments	11-19
Utility to generate Simulink representations of custom data types defined by external C code	11-19
Direct representation of fixed-point data types by Simulink.AliasType	11-19
Display of alias, base, or both data types in a model	11-20
More accurate comparison of nondouble data to specified minimum and maximum values	11-20
Deep copy of handle objects by Simulink.ModelWorkspace.assignin	11-21
Use of From Workspace block in a model that uses a data dictionary	11-22
Specify 64-bit integer data types without a Fixed-Point Designer license	11-22
Block Enhancements	11-23
Support for Scopes in For Each Subsystems	11-23
Scope Blocks: Support for nonvirtual bus and array of buses signals	11-23
Specify image file icons for MATLAB System block	11-23
Copy scope to clipboard	11-23
Interactive legend for scopes	11-23
Stem plot option for Scope block	11-23
Simulink Blocks: Simulink implements same workflow when adding a block through the user interface or the command line	11-24
Slider Gain block: Minimum and Maximum values must not be same	11-24
Default input signal attributes for MATLAB System block	11-24
Additional calls to Propagation Methods getOutputDataTypeImpl, getOutputSizeImpl and isOutputComplexImpl during the model pre-compile phase	11-24
Math Function block rem, mod, and pow function changes	11-24
Trigonometric Function block asin, asinh, acos, and acosh function changes	11-25
Dynamic memory allocation for unbounded arrays and large arrays	11-25
Better handling of promoted parameter	11-25
Connection to Hardware	11-26
Wireless Connectivity: Use UDP and TCP/IP blocks to let Simulink hardware targets communicate with each other	11-26
Support for print and println on Arduino Serial Transmit block	11-26
Hardware plugin detection for Arduino boards in MATLAB, Simulink	11-26
Blocks added to LEGO EV3 support package	11-26
Blocks added to Raspberry Pi support package	11-26

Support for all Android smartphones and tablets	11-27
Blocks added to Android support package	11-27
Blocks added to Apple iOS support package	11-27
Support for Scope block on Apple iOS and Android apps	11-28
MATLAB Function Blocks	11-29
Dynamic memory allocation for unbounded arrays and large arrays . . .	11-29
Nested functions	11-29
Handle classes in value classes	11-30
Constant folding of value classes	11-30
Class properties and structure fields passed by reference to external C functions	11-30
Function specialization prevention with coder.ignoreConst	11-31
New coder.unroll syntax for more readable code	11-31
Size argument for coder.opaque	11-32
Code generation for more MATLAB functions	11-32
Code generation for more Audio Toolbox System objects	11-32
Code generation for more Communications System Toolbox System objects	11-32
Code generation for more DSP System Toolbox System objects	11-33
Code generation for more Phased Array System Toolbox System objects	11-33
Code generation for more Robotics System Toolbox functions and classes	11-33
Code generation for more Signal Processing Toolbox functions	11-34
Statistics and Machine Learning Toolbox Code Generation: Generate C code for prediction by using linear models, generalized linear models, decision trees and ensembles of classification trees	11-34
Enhancement to synchronous subsystem support	11-34
Support for tunable structure array parameters	11-35
State behavior specification for function-call input events	11-35
S-Functions	11-36
Functionality being Removed or Changed	11-36

R2016b

Simulation Analysis and Performance	12-2
Just-in-Time Acceleration Builds: Quickly build the top-level model for improved performance when running simulations in Accelerator mode	12-2
Dataset Signal Plot: View and analyze dataset signals directly from the MATLAB command line	12-2
Multi-State Image Dashboard Block: Display different images based on the signal value	12-2
Simplified tasking mode setup	12-2
Diagnostic Suppressor: Suppress specific simulation warnings on particular blocks	12-3
Diagnostic Viewer: Improved build diagnostics display	12-3

Export functions allow periodic function calls	12-3
Simulink Editor	12-5
Property Inspector: Edit parameters and properties of model elements using a single interface	12-5
Edit-Time Checking: Detect and fix potential issues in your model at design time	12-5
Finder: Search for model elements using improved interface	12-5
Annotations in Libraries: Add annotations from libraries into models . . .	12-5
Library Browser: Expand or collapse libraries by default	12-6
Library Browser API: Programmatically refresh the Library Browser . . .	12-6
Annotations: Click once to select annotation	12-6
Default Model Font: Specify default font for model elements	12-6
Simulink Preferences: Simplified and reorganized interface	12-6
Simulink Editor Fonts: FreeType font engine replaces Windows GDI font engine	12-7
Component-Based Modeling	12-8
Initialize and Terminate Function Blocks: Respond to events to model dynamic startup and shutdown behavior	12-8
Variant Subsystem Condition Propagation: Automatically assign variant conditions to blocks outside the subsystem for improved performance	12-8
Simulink Units Updates	12-8
Additional SimStruct Functions to Specify Units for Input and Output Ports	12-9
Additional heterogeneous targets supported for concurrent execution . .	12-9
Simulink.BusElement: SamplingMode property removed to support having blocks specify whether to treat inputs as frame-based signals	12-9
Export functions allow periodic function calls	12-9
Variant Refresh: Improved performance with removal of live refresh . .	12-10
Variant Subsystem: Convert Subsystems with physical ports to Variant	12-10
Variant Reducer: Additional model reduction modes in Variant Reducer (requires SLDV product license)	12-10
Enhanced find_mdrefs function: Keep models loaded that the function loads	12-11
Subsystem conversion to referenced models: Automatic subsystem wrapper and improved Goto and From block handling	12-11
Disallow multiple iterations of root Inport function-call with discrete sample time	12-11
Project and File Management	12-12
Default Model Template: Use your own customized settings when creating new models	12-12
Upgrade Advisor API: Automate the process of upgrading large model hierarchies	12-13
Project-Wide Search: Search inside all models and supporting files	12-13
Refactoring Tools: Rename folders and automatically replace all references	12-13
Project Toolbox Analysis: Find products and toolboxes used by a project	12-13

Project Derived File Analysis: Find out-of-date .p, .slxp. and .mex files in a project	12-13
Project Export Profiles: Share specified files to zip archive	12-14
Project Batch Job Report: Archive results in a document	12-14
Git Submodules: Include submodules in your project	12-14
C/C++ file dependency analysis: View dependencies between C/C++ source and header files in the Impact graph	12-14
Updated source control SDK: Write a source control integration providing file-based actions and annotations	12-14
Diff Tools: Customize external source control tools to use MATLAB to compare and merge	12-14
SVN Cleanup: Fix problems with working copy locks	12-15
Command-line Impact Analysis: Update and analyze the dependencies graph programmatically	12-15
Data Management	12-16
Model Data Editor: Configure model data properties using a table within the Simulink Editor	12-16
Output Logging: Log data incrementally, with support for rapid accelerator mode and variant conditions	12-16
Logging Inside For Each Subsystem: Log signals inside a For Each subsystem by marking lines with antennas	12-17
Logged Dataset Data Analysis: Call same function for all timeseries objects in logged Dataset data	12-17
Scalar expansion of initial value for data store	12-17
Technique to determine whether signal has variable size	12-17
View your model configuration parameters as a group on the All Parameters tab	12-17
Enhanced error reporting and extended syntax for specifying argument dimensions for function specifications in Legacy Code Tool	12-18
Class to package and share breakpoint and table data for lookup tables	12-18
Root Inport Mapping Tool Updates	12-18
Option to disable resolution of signals and states to Simulink.Signal objects	12-19
Help fixing configuration errors from Diagnostic Viewer	12-19
Metadata for Logging to Persistent Storage: Simulation metadata contains persistent storage logging settings to facilitate analysis of data from multiple simulations	12-19
Improved display of large arrays by Model Explorer and Simulink.Parameter property dialog boxes	12-20
Configuration set in base workspace resolves variables in base workspace	12-20
Connection to Hardware	12-21
Raspberry Pi 3 Support: Run Simulink models on Raspberry Pi 3 hardware	12-21
Arduino: Improved External mode over serial communication	12-21
Simulink Support Package for Samsung GALAXY Android Devices renamed to Simulink Support Package for Android Devices	12-21
Google Nexus Support: Run Simulink models on Google Nexus Android devices	12-21
Block Enhancements	12-22

State Reader and Writer Blocks: Reset and record states during model execution	12-22
MATLAB System Block Support for Global Data: Access Simulink data stores from System objects using global variables	12-22
MATLAB System block now supports enumerated data types	12-22
MATLAB System Block Support for LAPACK: Generate faster standalone code for linear algebra in a MATLAB System block	12-22
Simpler way to call System objects	12-23
System objects support for additional inputs, global variables, and enumeration data types	12-23
Prelookup and Interpolation Using Prelookup Block Bus Support: Simplify and extend use of index and fraction signals	12-23
From Spreadsheet block updates	12-24
Property inspector available for Simulink blocks	12-24
Manual Variant Source and Sink: Switch manually between different variants without using conditions	12-24
Block Mask: Improved performance while evaluating mask parameter in fast restart mode	12-24
Slider Range Parameter: Dynamically change the range of slider and dial parameter	12-24
Some types of unit delay blocks obsoleted	12-24
Enhanced discrete block behavior	12-25
MATLAB Function Blocks	12-26
MATLAB Language Support: Use recursive functions and anonymous functions in a MATLAB Function block	12-26
Variable-Size Cell Array Support: Use cell to create a variable-size cell array in a MATLAB Function block	12-26
Error for testing equality between enumeration and character array in a MATLAB Function block	12-27
Incremental build for relocation of MATLAB program files on the MATLAB path	12-27
Additional I/O Support: Generate code for fseek, ftell, fwrite	12-28
Code generation for additional MATLAB functions	12-28
Code generation for additional Audio Toolbox functions	12-28
Code generation for additional Computer Vision System Toolbox functions	12-28
Statistics and Machine Learning Toolbox Code Generation: Generate code for prediction by using SVM and logistic regression models	12-29
Communications and DSP Code Generation: Generate code for additional functions	12-29
Conditional breakpoints for run-time debugging	12-30
Compiler optimization parameter support for faster simulation	12-30
Run-Time error stack in Diagnostic Viewer	12-30
Modeling Guidelines	12-31
Modeling guidelines for high-integrity systems	12-31

Simulation Analysis and Performance	13-2
Automatic Solver Option: Set up and simulate your model more quickly with automatically selected solver settings	13-2
One-Click Display: Click a signal line when the simulation is running to view the current value	13-2
Simulation Metadata Diagnostics: Understand why a simulation has stopped in batch or individual runs	13-2
Multi-Input Root Inport Mapping: Connect multiple sets of input signals to your Simulink model for interactive or batch simulation	13-2
Simulation for Mixed Targets: Simulate system-level designs that integrate referenced models targeting an assembly of heterogeneous embedded devices	13-2
Time Out feature for Performance Advisor run time	13-3
Solver Profiler to speed up simulation performance	13-3
Diagnostic Viewer performance improvement	13-3
 Component-Based Modeling	 13-4
Variant Source and Sink Blocks with Condition Propagation: Design variant choices and automatically remove unneeded functionality based on block connectivity	13-4
Scoping Simulink Functions: Call Simulink Function blocks within a subsystem hierarchy	13-4
Simulink Units: Specify, visualize, and check consistency of units on interfaces	13-4
Mask Dialogs: Create masks with flexible layout options and new control parameters	13-6
Mask Images: Quickly add images to masks and while keeping the port names visible	13-6
Tracing Simulink Functions: Display connections between all Function Callers and a Simulink Function	13-6
Signal Label Propagation for Referenced Models: Propagate signal labels out of referenced models by default	13-7
Simulink.SubSystem.convertToModelReference function for multiple subsystem conversion: Convert multiple subsystems with one command	13-7
Subsystem to Model Reference Conversion: Insert subsystem wrapper to preserve model layout	13-7
Model Reference Conversion Automatic Fix for Goto Blocks: Convert subsystems with Goto blocks more easily	13-7
Virtual Bus Signals Across Model Reference Boundaries: Use virtual bus signals as inputs or outputs of a referenced model	13-8
Bus Selector and Bus Assignment Block Signals: Display full signal path while editing a model	13-8
Multi-Input Bus-Capable Block Ports: Simulate unconnected multi-input bus- capable block ports without error	13-8
Output Blocks with Bus Output: Simulate Outport blocks with a bus output without error	13-9
Function-Call Split block with multiple outputs	13-9
Function-Call Split block with no input signal	13-9
Trigger port with inherited periodic function-call signal	13-9

Standalone code generation for models with asynchronous function-call inputs	13-9
Additional component parameters saved with Simulink.ConfigSet.saveAs	13-10
Project and File Management	13-11
Start Page: Get started or resume work faster by accessing templates, recent models, and featured examples	13-11
Automatic Renaming: Update all references in a project when you rename models, libraries, or MATLAB files	13-13
Three-Way Model Merge: Resolve conflicts between revisions and ancestor models using Simulink projects	13-13
Template API: Programmatically create models and projects from custom templates	13-13
Export function: Export to previous version using Simulink.exportToVersion	13-13
Dirty Model Management: Identify, save, or discard unsaved changes in project models	13-14
Source Control API: Programmatically get modified files and revision information	13-14
Source Control Notifications: List changed files on update (SVN); find out if your branch is behind the origin (Git)	13-14
SVN Externals: Include files in projects from other repositories or repository locations	13-14
Custom Shortcut Icons: Personalize frequent task buttons on the toolbar	13-14
Simplified Configuration Parameters: Configure model more easily using streamlined category panes	13-15
Simulink Editor	13-18
Single-Selection Actions: Access commonly used editing actions when clicking a block or signal line	13-18
Multiple-Selection Cue: Selecting multiple blocks in the Simulink Editor shows new cue	13-18
Single Click for Quick Insert: Click block name once to insert block from list	13-18
Interactive Library Unlocking: Click lock symbol in custom libraries to unlock	13-19
Improved block search usability	13-19
Data Management	13-20
Signal and State Logging to File: Log data directly to a MAT-file for long simulations	13-20
Preserve symbolic constants in propagated signal dimensions	13-20
Dataset Format for Signal Logging: Log signals in format used for other logging	13-21
Unlimited Number of Data Points for Logging by Default: Log all data points by default	13-21
Root Inport Mapping Tool Updates	13-21
Function to convert MAT-file contents to Simulink.SimulationData.Dataset object	13-22
Functions to identify and close data dictionaries	13-22
Navigation to variables from additional block dialog boxes	13-22

Functionality Being Removed or Changed	13-23
Connection to Hardware	13-24
Hardware implementation parameters enabled by default	13-24
Mac Support for LEGO EV3: Run Simulink models on LEGO EV3 hardware from a Mac	13-24
Block Enhancements	13-25
From Spreadsheet Block Updates	13-25
System object enhancements to MATLAB System block	13-25
Unit Delay block does not accept rate transitions	13-25
Matrix Interpolation Block for Multidimensional Lookup Table Data . . .	13-25
Enhanced System Object Development with MATLAB Editor	13-25
Scope Block and Signal Viewer Enhancements	13-26
MATLAB Function Blocks	13-27
Cell Array Support: Use additional cell array features in a MATLAB Function block	13-27
Non-Power-of-Two FFT Support: Generate code for fast Fourier transforms for non-power-of-two transform lengths	13-27
Faster Standalone Code for Linear Algebra: Generate code that takes advantage of your own target-specific LAPACK library	13-27
Concatenation of variable-size, empty arrays	13-28
xcorr Code Generation: Generate faster code for xcorr with long input vectors	13-29
More keyboard shortcuts for the MATLAB Function report	13-30
Code generation for Audio Toolbox functions and System objects	13-30
Code generation for additional Computer Vision System Toolbox functions and objects	13-31
Image Processing Toolbox Code Generation: Generate code for additional functions	13-31
Code generation for additional MATLAB functions	13-31
Changes to code generation support for MATLAB functions	13-31
Code generation for additional Communications System Toolbox functions	13-31
Code generation for additional DSP System Toolbox	13-32
Code generation for additional Phased Array System Toolbox functions	13-32
Code generation for WLAN System Toolbox functions and System objects	13-32
Units for MATLAB Function blocks	13-32
In/Out Arguments: Specify same variable name for in/out arguments . .	13-33
UserData parameter available for storing values	13-33
Modeling Guidelines	13-34
High-Integrity Systems: Model object, file, and folder names	13-34
Model Advisor	13-35
Additional functionality for Model Advisor check that checks for usage of partial structure	13-35

S-Functions	13-36
ssSetSolverNeedsReset updates	13-36
ssSetSkipContStatesConsistencyCheck	13-36

R2015aSP1

Bug Fixes

R2015b

Simulink Editor	15-2
Signal Line Healing: Click once to repair broken signal lines after deleting blocks	15-2
Multilingual Names and Comments: Use any language to write block names, signal names, and MATLAB Function comments	15-2
Programmatic removal of mask dialog box controls and mask parameters	15-2
Alternative view of library contents in Library Browser	15-2
Prompt to set key parameter when dragging a block from the Library Browser	15-3
Printing to Postscript and EPS file formats	15-3
Programmatic addition of areas and images in models	15-3
Redesigned interface for Model Dependency Viewer	15-3
Visual cue for undo and redo of block parameter value changes	15-3
Simulation Analysis and Performance	15-5
New Interface for Scopes: View and debug signals with cursors and measurements	15-5
Fast Restart API: Programmatically run consecutive simulations more quickly	15-7
Auto solver that chooses solver for a model	15-7
Tunability of struct parameters in rapid accelerator mode	15-7
Port value labels for nonvirtual buses and bus signals	15-7
Visualization of inserted rate transition blocks	15-7
Common format for saving states, output, and final states data and other logging and loading techniques	15-8
Extended support for root Inport loading using Dataset format in rapid accelerator	15-8
Free MinGW-w64 compiler for running simulations on 64-bit Windows®	15-8
Component-Based Modeling	15-9
More flexible configuration of Application lifespan (days) parameter in a model reference hierarchy for simulation	15-9

Model Advisor checks for simplified initialization mode	15-9
Changes to export-function models	15-9
Saving of list view parameters with Simulink.ConfigSet.saveAs	15-9
Project and File Management	15-10
Referenced Projects: Create reusable components for large modeling projects	15-10
Configuration Parameters List View: List, edit, and search all configuration parameters within your model	15-10
Project Creation from a Model: Quickly organize your model and all dependent files	15-11
Faster, Improved Dependency Analysis: Analyze projects several times faster, identify referenced project files, and view library blocks	15-11
Management of shadowed and dirty files	15-12
Comparison of any pair of file revisions	15-12
Updated power window example	15-12
Case-sensitive model and library names	15-13
Warning for Model Info Configuration Manager	15-13
Data Management	15-14
Interval Logging: Specify start and stop time intervals to log only the data you need	15-14
Always-On Tunability: Tune all block parameters and workspace variables during a simulation	15-14
Arrays of structures as parameters	15-17
Improved methods to create custom data objects	15-18
No creation of parameter objects for mask initialization code	15-18
Sample time for signal logging	15-18
Same format for logging states, output, and final states as used for other logging and loading techniques	15-19
Root Inport loading in rapid accelerator mode using Dataset format	15-19
Logged signals with propagated names	15-19
Tolerance for data type mismatch between bus elements and structure fields	15-19
Summary of changes made to data dictionary	15-20
Rename All in Goto blocks	15-20
Change to visibility of SamplingMode property of signal objects	15-20
Continued availability of Simulink.saveVars	15-20
Simulink.SimulationData.Dataset updates	15-21
Edit Input button is now Connect Input	15-21
Legacy Code Tool support for conditional outputs	15-21
Connection to Hardware	15-22
Raspberry Pi 2 Support: Run Simulink models on Raspberry Pi 2 Model B hardware	15-22
Arduino Yun: Design and run Simulink models on Arduino Yun hardware	15-22
Hardware Implementation Selection: Quickly generate code for popular embedded processors	15-22
Signal Management	15-24

Virtual bus signal inputs to blocks that require nonbus or nonvirtual bus input	15-24
Entire nested bus assignment for Bus Assignment block	15-25
Block Enhancements	15-26
Waveform Generator Block: Define and output arbitrary waveform signals	15-26
From Spreadsheet Block: Read signal data into Simulink from a spreadsheet	15-26
MATLAB System block support for nonvirtual buses	15-26
Inport block update	15-26
From File updates for file name and signal preview	15-26
Inheriting of continuous sample time for discrete blocks	15-26
Evenly spaced breakpoints in Lookup Tables	15-26
Integrator block: Wrapped states for modeling rotary and cyclic state trajectories	15-26
Variant Subsystem block: Enhanced option for generating preprocessor conditionals	15-27
Constant sample time in S-function blocks	15-27
MATLAB Function Blocks	15-28
Calling of Simulink Functions	15-28
Nondirect feedthrough in MATLAB Function blocks	15-28
Overflow and data range detection settings unified with Simulink	15-28
No frame-based sampling mode for outputs	15-29
Code generation for cell arrays	15-29
LAPACK calls during simulation for algorithms that call linear algebra functions	15-30
Code generation for additional Image Processing Toolbox and Computer Vision System Toolbox functions	15-30
Code generation for additional Statistics and Machine Learning Toolbox functions	15-30
Code generation for additional MATLAB functions	15-31
Code generation for additional Communications System Toolbox, DSP System Toolbox, and Phased Array System Toolbox functions and System objects	15-31

R2015a

Simulink Editor	16-2
Bus Smart Editing Cue: Automatically create a bus from a set of signals	16-2
Area Annotations: Call out and separate regions of interest in model	16-2
Perspectives Controls: Access alternative views of your model, such as harness and interface views	16-3
Saving of viewmarks in Simulink models	16-3
Highlighting of the subsystem you navigated from	16-3
Annotation connector colors and width	16-4
Undo and redo of block parameter value changes	16-4

Display of product name in model title bar	16-4
Simulation Analysis and Performance	16-5
Dashboard Block Library: Tune and test simulations with graphical controls and displays	16-5
Algebraic Loop Highlighting: Find and remove algebraic loops in the model to boost simulation speed	16-5
Faster Simulations with Accelerated Referenced Models: Run faster consecutive simulations and step back and forth through simulations	16-5
Use SimulationMetadata to retrieve simulation metadata information	16-6
Simplified conversion of logged data to Dataset format for a common logged data format	16-6
Default setting for Automatic solver parameter selection	16-6
Improved heuristic for step size calculation	16-6
Step size details in solver information tooltip	16-7
Solver information in model after simulation	16-7
Component-Based Modeling	16-8
Consistent Data Support for Testing Components: Load input and log data of a component from buses and all data types	16-8
Model Reference Conversion Advisor enhancements	16-8
Reduced algebraic loops during model reference simulation	16-9
Multi-instance support for nonreusable functions in referenced models	16-9
Model referencing checks in Model Advisor to reduce warning messages	16-9
Model configuration parameter changes	16-10
Property name change in Simulink.ConfigSetRef	16-11
Flexible structure assignment of buses	16-11
Support for empty subsystems as variant choices	16-11
Conversion of MATLAB variables used in variant control expressions into Simulink.Parameter objects	16-11
Project and File Management	16-12
Simulink Project Sharing: Share a project using GitHub, email, or a MATLAB toolbox	16-12
Interactively manage the MATLAB search path for your project	16-12
Easy viewing and editing of project labels	16-12
Changed file lists and branch deletion in Git Manage Branches dialog box	16-13
New preferences to control loading and saving models	16-14
Improved error reporting from get_param, set_param and save_system	16-15
Model dependency analysis option to find enumeration definition files	16-15
Export to previous version supports seven years	16-15
Data Management	16-16
Data Dictionary API: Automate the creation and editing of data dictionaries with MATLAB scripts	16-16
Rename All: Change the name of a parameter and all its references	16-16
MATLAB Editor features for editing model workspace code	16-16

Management of variables from block dialog box fields	16-16
Other Data section added to data dictionary	16-16
Model-wide renaming of data stores	16-17
Reporting of enumerated types used by model	16-17
Root Inport Mapping tool updates	16-17
Connection to Educational Hardware	16-18
Simulink Support Package for Apple iOS Devices: Create an App that runs Simulink models and algorithms on your Apple iOS device	16-18
MathWorks response to the Shellshock vulnerability	16-18
Removed support for Gumstix Overo and PandaBoard hardware	16-18
Signal Management	16-19
Array of buses with Unit Delay block	16-19
Block Enhancements	16-20
Resettable Subsystem block to reset the subsystem states	16-20
Conditional display of the Sample Time parameter	16-20
Inheritance of frame-based input returns error	16-20
Scope block to Time Scope Block conversion	16-23
Option to provide PID gains as external inputs to PID Controller and PID controller (2DOF) blocks	16-23
Improvements for creating System objects	16-24
MATLAB System block support for model coverage analysis	16-25
Enable port on the Delay block	16-25
MATLAB Function Blocks	16-26
More efficient generated code for logical indexing	16-26
Faster compile time for large functions and models due to decreased constant folding limit	16-26
JIT compilation technology to reduce model update time	16-26
Code generation for casts to and from types of variables declared using coder.opaque	16-27
Improved recognition of compile-time constants	16-28
Code generation for additional Image Processing Toolbox and Computer Vision System Toolbox functions	16-29
Code generation for additional Communications System Toolbox, DSP System Toolbox, and Phased Array System Toolbox System objects ..	16-29
Code generation for additional Statistics and Machine Learning Toolbox functions	16-30
Code generation for additional MATLAB functions	16-30
Code generation for additional MATLAB function options	16-30
Model Advisor	16-31
Multiple instances of advisors	16-31
Improved advisor startup performance	16-31
Model Advisor check input parameters retained for each instance of check	16-32
Model referencing checks in Model Advisor to reduce warning messages	16-32

Simulink Editor	17-2
Smart Editing Cues: Accelerate model building with just-in-time contextual prompts	17-2
Viewmarks: Save graphical views of a model for quick access to areas of interest	17-3
Annotation Connectors: Associate annotations with blocks in models . . .	17-4
Edit bar for quick annotation formatting	17-4
Annotation table column and row resizing	17-4
Reuse of annotation text formatting	17-5
Annotation layering	17-5
Access Diagnostic Viewer from status bar	17-5
Printing to file	17-5
Simulation Analysis and Performance	17-6
Fast Restart: Run consecutive simulations more quickly	17-6
New Simulation Data Inspector: View live signal data and access visualization options such as data cursors	17-6
Fixed Point Support for Conditional Breakpoints	17-6
Quick Scan simulation in Performance Advisor for faster diagnosis	17-6
Removal of warning when variable-step solver is selected for discrete models	17-6
Block callbacks not evaluated in Rapid Accelerator mode with up-to-date check off	17-7
Functionality Being Removed or Changed	17-7
Improvements to Scope blocks and Scope viewers	17-7
Component-Based Modeling	17-9
Model Templates: Build models using design patterns that serve as starting points to solve common problems	17-9
Simulink Functions: Create and call functions across Simulink and Stateflow	17-9
Interface Display: View and trace the input and output signals of a model or subsystem	17-9
Model reference conversion enhancements	17-9
Include Simulink Models as Variant Choices	17-10
Arithmetic and Bit-Wise Operators in Variant Condition Expressions . . .	17-10
Export of chart-level functions in export-function models	17-11
Project and File Management	17-12
Block Dependencies in Impact Graph: Highlight the blocks affected by changes made to project files	17-12
Identify modified or conflicted folder contents using source control summary status	17-12
Simplified file views in Simulink Project	17-12
Simplified browsing and sharing of project templates	17-12
SVN and Git example Simulink Projects	17-13
Data Management	17-14

Root Import Mapping tool	17-14
Minimize and maximize buttons for the Configuration Parameters dialog box	17-14
Overflow diagnostics to distinguish between wrap and saturation	17-14
Change in behavior of <code>isequaln</code>	17-14
Change in Simulink check for types derived from <code>Simulink.IntEnumType</code>	17-15
Methods no longer inherited by Simulink enumerations	17-15
Connection to Educational Hardware	17-17
More Arduino Support: Run your model on Arduino Leonardo, Mega ADK, Mini, Fio, Pro, Micro and Esplora boards	17-17
Documentation installation with hardware support package	17-17
Signal Management	17-18
Signal name inheritance from bus object elements	17-18
Faster and more flexible <code>Simulink.Bus.createMATLABStruct</code> function ..	17-18
Block Enhancements	17-19
Nearest interpolation method available for n-D Lookup Table Block ...	17-19
MATLAB System block updates	17-19
Level-1 MATLAB S-Functions	17-19
Unfiltered-derivative option in discrete-time PID Controller blocks ...	17-19
MATLAB Function Blocks	17-20
Code generation for additional Image Processing Toolbox and Computer Vision System Toolbox functions	17-20
Code generation for additional Communications System Toolbox and DSP System Toolbox functions and System objects	17-20
Code generation for <code>ode23</code> and <code>ode45</code> ordinary differential equation solvers in MATLAB	17-21
Code generation for additional MATLAB functions	17-21
Code generation for additional MATLAB function options	17-21
Code generation for enumerated types based on built-in MATLAB integer types	17-22
Code generation for function handles in structures	17-22
Collapsed list for inherited properties in code generation report	17-23
Model Advisor	17-24
New check for Unit Delay and Zero-Order Hold blocks that perform rate transition	17-24
Highlighted configuration parameters from Model Advisor reports ...	17-24

R2014a

Simulink Editor	18-2
------------------------------	-------------

Annotations with rich text, graphics, and hyperlinks	18-2
Diagnostic Viewer to collect information, warnings, and error messages	18-2
Option to bring contents of a hierarchical subsystem into the parent subsystem with one click	18-2
Support for native OS touch gestures, such as pinch-to-zoom and panning	18-3
Operating system print options for models	18-3
Preference for line crossing style	18-3
Scalable graphics output to clipboard for Macintosh	18-3
Sliders, dials, and spinboxes available as parameter controls in masks	18-3
Component-Based Modeling	18-4
Option to choose default variants	18-4
Option to choose variants that differ in number of input and output ports	18-4
Advisor-based workflow for converting subsystems to Model blocks	18-4
Single-model workflow for algorithm partitioning and targeting of multicore processors and FPGAs	18-4
Easier MATLAB System block creation via autocompletion and browsing for System object names	18-4
Improved algebraic loop handling and reduced data copies with the Bus Selector block	18-5
Faster response time when opening bus routing block dialog boxes and propagating signal labels	18-5
Usability enhancements to configure a model for concurrent execution on a target	18-5
Default setting of Underspecified initialization detection diagnostic is Simplified	18-6
Discrete-Time Integrator block has dialog box changes for initialization	18-6
System objects Propagates mixin methods	18-6
Simulation Analysis and Performance	18-8
Reduced setup and build time for Model blocks when using Rapid Accelerator mode	18-8
Performance Advisor checks that validate overall performance improvement for all suggested changes and set code generation option for MATLAB System block	18-8
Improved navigation of the Performance Advisor report	18-8
Block behavior for asynchronous initiator with constant sample time . . .	18-9
Global setting for validation of checks in Performance Advisor	18-9
Guided setup in Performance Advisor	18-9
Project and File Management	18-10
Branching support through Git source control	18-10
Comparison of project dependency analysis results	18-10
Impact graph layout algorithm improved for easier identification of top models and their dependencies	18-10
Impact analysis example for finding and running impacted tests	18-10
Performance improvements for common scripting operations such as adding and removing files and labels	18-11

Conflict resolution tools to extract conflict markers	18-11
Updated Power Window Example	18-11
Data Management	18-12
Data dictionary for defining and managing design data associated with models	18-12
Rapid Accelerator mode signal logging enhanced to avoid rebuilds and to support buses and referenced models	18-12
Simplified tuning of all parameters in referenced models	18-12
Simulink.findVars supported in referenced models	18-13
Saving workspace variables and their values to a MATLAB script	18-13
Frame-based signals in the To Workspace block	18-13
Simulation mode consistency for Data Import/Export pane output options parameter	18-14
Dimension mismatch handling for root Inport blocks improved	18-14
Simulink.Bus.createObject support for structures of timeseries objects	18-15
Signal logging override for model reference variants	18-15
Improved To Workspace block default for fixed-point data	18-15
Legacy Code Tool support for 2-D row-major matrix	18-15
Model Explorer property name filtering refined	18-15
Connection to Educational Hardware	18-17
Support for Arduino Due hardware	18-17
Support for Arduino WiFi Shield hardware	18-17
Support for LEGO MINDSTORMS EV3 hardware	18-17
Updates to support for LEGO MINDSTORMS NXT hardware	18-18
Support for Samsung GALAXY Android devices	18-18
Block Enhancements	18-20
Enumerated data types in the Direct Lookup Table (n-D) block	18-20
Improved performance and code readability in linear search algorithm for Prelookup and n-D Lookup Table blocks	18-20
System object file templates	18-20
Relay block output of fixed-in-minor-step continuous signal for continuous input	18-20
MATLAB Function Blocks	18-21
Generating Simulation Target typedefs for imported bus and enumerated data types	18-21
Complex data types in data stores	18-21
Unicode character support for MATLAB Function block names	18-21
Support for int64 and uint64 data types in MATLAB Function blocks	18-21
Streamlined MEX compiler setup and improved troubleshooting	18-21
Code generation for additional Image Processing Toolbox functions	18-21
Code generation for additional Signal Processing Toolbox, Communications System Toolbox, and DSP System Toolbox functions and System objects	18-22
Code generation for MATLAB fminsearch optimization function, additional interpolation functions, and additional interp1 and interp2 interpolation methods	18-22
Code generation for fread function	18-23

Enhanced code generation for switch statements	18-23
Code generation for value classes with set.prop methods	18-23
Code generation error for properties that use AbortSet attribute	18-23
Toolbox functions for code generation	18-23
Modeling Guidelines	18-26
Modeling guidelines for high-integrity systems	18-26
Model Advisor	18-27
Improved navigation of the Model Advisor report, including a navigation pane, collapsible content, and filters based on check status	18-27
Option to run Model Advisor checks in the background	18-27
Upgrade Advisor check for get_param calls for block CompiledSampleTime	18-27
Upgrade Advisor check for signal logging in Rapid Accelerator mode	18-27

R2013b

New Simulink Editor	19-2
Ability to add rich controls, links, and images to customized block interfaces using the Mask Editor	19-2
Content preview for subsystems and Stateflow charts	19-2
Comment-through capability to temporarily delete blocks and connect input signals to output signals	19-2
New options added to find_system command	19-2
Visual cues for signal lines that cross	19-2
UTF-16 character support for block names, signal labels, and annotations in local languages	19-3
Unified Print Model dialog box for printing	19-3
Block Parameters dialog box access from Block Properties dialog box	19-4
Notification bar icon indicator for multiple notifications	19-4
Component-Based Modeling	19-5
MATLAB System block for including System objects in Simulink models	19-5
Variant Manager that manages all the variants in a model in one place	19-5
Improved componentization capabilities for modeling scheduling diagrams with root-level function-call inports	19-5
Array of buses signal logging in model reference accelerator mode	19-5
Ability to add, delete, and move input signals within Bus Creator block	19-5
Streamlined approach to migrating from Classic to Simplified initialization mode	19-5
Simplified display of sorted execution order	19-5
Enhanced model reference rebuild algorithm for MATLAB Function blocks	19-5

Simulation Analysis and Performance	19-7
LCC compiler included on Windows 64-bit platform for running simulations	19-7
Signal logging in Rapid Accelerator mode	19-7
Performance Advisor checks for Rapid Accelerator mode and data store memory diagnostics	19-7
Long long integers in simulation targets for faster simulation on Win64 machines	19-7
Auto-insertion of rate transition block	19-8
Compiled sample time for multi-rate blocks returns cell array of all sample times	19-8
Improvement to model reference parallel build check in Performance Advisor	19-10
Improved readability in Performance Advisor reports	19-10
Simulation Data Inspector launch using simplot command	19-10
Project and File Management	19-11
Impact analysis by exploring modified or selected files to find dependencies	19-11
Option to export impact analysis results to the workspace, batch processing, or image files	19-11
Identification of requirements documents during project dependency analysis	19-11
Simplified label creation by dragging a label onto files in any view	19-11
Shortcut renaming, grouping, and execution from any view using the Toolstrip	19-12
Data Management	19-13
Streamlined selection of one or more signals for signal logging	19-13
Simplified modeling of single-precision designs	19-13
Connection status visualization and connection method customization for root inport mapping	19-14
Conversion of numeric variables into Simulink.Parameter objects	19-14
Model Explorer search options summary hidden by default	19-15
Simulink.DualScaledParameter class	19-15
Legacy data type specification functions return numeric objects	19-15
Root Inport Mapping Error Messages	19-17
Root inport mapping example	19-17
Connection to Educational Hardware	19-18
Ability to run models on target hardware from the Simulink toolbar ...	19-18
Support for Arduino hardware available on Mac OS X	19-18
Support for Arduino Ethernet Shield and Arduino Nano 3.0 hardware .	19-19
Signal Management	19-20
Port number display to help resolve error messages	19-20
Enforced bus diagnostic behavior	19-20
Block Enhancements	19-22
Improved performance of LUT block intermediate calculations	19-22

Name changes that unify storage class parameters	19-22
Warnings when using old parameter names with spaces	19-22
Strictly monotonically increasing time values on Repeating Sequence block	19-22
pow function in Math function block that supports Code Replacement Library (CRL)	19-23
Continuous Linear Block improvements, such as diagnostics, readability, and stricter checks	19-23
MATLAB Function Blocks	19-24
Code generation for Statistics Toolbox and Phased Array System Toolbox	19-24
Toolbox functions for code generation	19-24
External C library integration using coder.ExternalDependency	19-24
Updating build information using coder.updateBuildInfo	19-25
Conversion of MATLAB expressions into C constants using coder.const	19-25
Highlighting of constant function arguments in the compilation report	19-25
coder.target syntax change	19-25
LCC compiler included on Windows 64-bit platform for running simulations	19-25
Modeling Guidelines	19-26
Modeling guidelines for high-integrity systems	19-26
Model Advisor	19-27
Collapsible content within Model Advisor reports	19-27
Reorganization of Model Advisor checks	19-27
Check for strict single precision usage	19-27

R2013a

New Simulink Editor	20-2
Reordering of tabs in tabbed windows	20-2
Scalable vector graphics for mask icons	20-2
Simulation Stepper Default Value Change	20-2
Component-Based Modeling	20-3
Direct active variant control via logical expressions	20-3
Live update for variant systems and commented-out blocks	20-3
Masking of linked library blocks	20-3
Target profiling for concurrent execution to visualize task execution times and task-to-core assignment	20-3
Incremental block-to-task mapping workflow support enabled by automatic block-to-task assignment for multicore execution on embedded targets	20-3

PIL and SIL modes for concurrent execution	20-3
Parameterized task periods for concurrent execution	20-4
Relaxed configuration parameter setting requirements	20-4
Connection to Educational Hardware	20-5
Support for Gumstix Overo hardware	20-5
Support for Raspberry Pi hardware	20-5
Blocks for GPIO, LED, and eSpeak Text to Speech on BeagleBoard	20-6
Blocks for GPIO, LED, and eSpeak Text to Speech on PandaBoard	20-6
Blocks for Compass and IR Receiver sensors on LEGO MINDSTORMS NXT	20-6
Project and File Management	20-8
Simplified scripting interface for automating Simulink Project tasks	20-8
Option to use elements from multiple templates when creating a new project	20-8
Saving and reloading of dependency analysis results	20-8
Robust loading of projects with conflicted metadata project definition files	20-8
New project preferences to control logging and warnings	20-8
Data Management	20-9
Fixed-Point Advisor support for model reference	20-9
Arrays of buses loading and logging	20-9
Root Inport Mapping tool changes	20-9
New Root Inport Mapping Examples	20-9
Level-1 data classes not supported	20-10
Simulink data type classes do not support inexact enumerated property value matching	20-10
Simulation Analysis and Performance	20-12
Simulation Performance Advisor report that shows both check results and actions taken	20-12
Improved simulation performance when stepping back is enabled	20-12
Simulation Data Inspector run-configuration options for names and placement in run list	20-12
Arrays of buses displayed in Simulation Data Inspector	20-12
Simulation Data Inspector overwrite run specification	20-12
Signal Management	20-13
Referenced models sample times	20-13
Triggered subsystem sample times	20-13
Simulation of variable-size scalar signals	20-13
Block Enhancements	20-14
CORDIC approximation method for atan2 function of Trigonometric Function block	20-14
Product and Gain blocks support Basic Linear Algebra Subprogram (BLAS) library	20-14
Performance Advisor check for Delay block circular buffer setting	20-14

MATLAB Function Blocks	20-15
Masking of MATLAB Function blocks to customize appearance, parameters, and documentation	20-15
File I/O function support	20-15
Support for nonpersistent handle objects	20-15
Include custom C header files from MATLAB code	20-15
Load from MAT-files	20-16
coder.opaque function enhancements	20-16
Complex trigonometric functions	20-16
Support for integers in number theory functions	20-16
Enhanced support for class property initial values	20-17
Default use of Basic Linear Algebra Subprograms (BLAS) Libraries . . .	20-18
New toolbox functions supported for code generation	20-18
Function being removed	20-19
 Modeling Guidelines	 20-20
Modeling Guidelines for High-Integrity Systems	20-20
MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow	20-20
 Model Advisor	 20-21
Model Advisor checks reorganized in a future release	20-21
Model Advisor navigation between Upgrade Advisor, Performance Advisor, and Code Generation Advisor	20-21
Report	20-21
Preferences dialog box	20-21
By Product folder not displayed	20-22

R2012b

New Simulink Editor	21-2
Tabbed windows and automatic window reuse to minimize window clutter	21-2
Smart signal routing that determines the simplest signal line path without overlapping blocks	21-3
Explorer bar to help with navigating through a model	21-3
Simulation stepper to simulate and rewind a model one step at a time . .	21-3
Ability to comment out blocks	21-3
Subsystem badges to identify and look under masked subsystems	21-4
Reorganized menu to fit common Model-Based Design workflow	21-4
Palette for commonly used actions	21-4
Panning and zooming	21-5
Display of overlapping blocks	21-5
Unification of Simulink and Stateflow Editors	21-5
Simulink Editor preferences	21-5
Toolbar and status bar control	21-5
Visual editing based on model objects	21-6
Improved callback error handling	21-6

Simulink Editor Changes	21-6
Connection to Educational Hardware	21-17
Support for Arduino and PandaBoard hardware	21-17
Bluetooth download to LEGO MINDSTORMS NXT hardware	21-18
Performance	21-19
Simulation Performance Advisor that analyzes your model and provides advice on how to increase simulation performance	21-19
Project and File Management	21-20
Simulink default file format SLX that uses the OPC standard	21-20
Simulink Upgrade Advisor to help migrate files to the current release .	21-21
Built-in SVN adapter for Simulink Projects that provides connectivity to SVN and support for server-based repositories	21-21
Simulink Project Tool dependency graph that provides highlights by file type, dependency type, and label	21-21
Redesigned graphical tool for efficient Simulink Projects workflow	21-22
Batch operation support for files in a Simulink Project	21-22
Create and open recent Simulink Projects from MATLAB	21-22
Block Enhancements	21-24
Menu item to convert configurable and normal subsystems to variant subsystems	21-24
Masking improvements, including the ability to reuse masks, delete existing masks on blocks, and use the shortcut operator in mask callback code	21-24
Default output data type of Logic blocks changed to boolean	21-24
Signal Attributes tab of dialog box for Operator blocks renamed to Data Type	21-25
Parameter name changes for Unit Delay block	21-25
New variants of Delay block in Discrete library	21-25
Some Probe block parameters no longer support boolean data type ...	21-26
Internationalization of block dialog box titles and buttons and block tooltips	21-27
Enabled and triggered subsystems	21-27
Data Management	21-28
Variable Editor access from within Model Explorer	21-28
Logged simulation data from Simulation Data Inspector accessible from Simulink toolbar	21-28
Specify verifySignalAndModelPaths action	21-29
Import and map data to root-level input ports	21-29
Dataset signal logging format for increased flexibility and ease of use .	21-29
Data type field displays user-defined data types	21-30
Simulink.VariableUsage to get variable information	21-30
Customizable line specification in Simulation Data Inspector	21-30
Simulation Data Inspector report includes harness model information .	21-30
Component-Based Modeling	21-31

Model configuration for targets with multicore processors	21-31
New Simulink.GlobalDataTransfer class	21-31
Reduced memory usage in models with many library links	21-31
Configuration Reference dialog box to propagate and undo configuration settings to all referenced models	21-32
Context-dependent function-call subsystem input handling improved . .	21-32
Simulink.Variant object and the model InitFcn	21-33
Signal Management	21-34
Sample time propagation changes	21-34
Signal Builder	21-34
User Interface Enhancements	21-35
Model Advisor Dashboard	21-35
Show partial or whole model hierarchy contents	21-35
Improved icons for model objects	21-36
Simulink Debugger	21-36
Multiple modifiers for custom accelerators	21-36
Model Advisor Checks	21-37
Verify Syntax of Library Models	21-37
MATLAB Function Blocks	21-38
New toolbox functions supported for code generation	21-38
New System objects supported for code generation	21-38

R2012a

Component-Based Modeling	22-2
Interactive Library Forwarding Tables for Updating Links	22-2
Automatic Refresh of Links and Model Blocks	22-2
Model Configuration for Targets with Multicore Processors	22-2
MATLAB Function Blocks	22-4
Integration of MATLAB Function Block Editor into MATLAB Editor	22-4
Code Generation for MATLAB Objects	22-4
Specification of Custom Header Files Required for Enumerated Types	22-4
Data Management	22-5
New Infrastructure for Extending Simulink Data Classes Using MATLAB Class Syntax	22-5
Change in Behavior of isequal	22-6
isContentEqual Will Be Removed in a Future Release	22-6
Change in Behavior of int32 Property Type	22-7

RTWInfo Property Renamed	22-7
deepCopy Method Will Be Removed in a Future Release	22-7
New Methods for Querying Workspace Variables	22-7
Default Package Specification for Data Objects	22-8
Simulink.Parameter Enhancements	22-8
Custom Storage Class Specification for Discrete States on Block Dialog Box	22-8
Enhancement to set_param	22-8
Simulink.findVars Support for Active Configuration Sets	22-9
Bus Support for To File and From File Blocks	22-9
Bus Support for To Workspace and From Workspace Blocks	22-9
Logging Fixed-Point Data to the To Workspace Block	22-10
Improved Algorithm for Best Precision Scaling	22-10
Enhancement of Mask Parameter Promotion	22-10
File Management	22-11
SLX Format for Model Files	22-11
Simulink Project Enhancements	22-12
Signal Management	22-14
Signal Hierarchy Viewer	22-14
Signal Label Propagation Improvements	22-14
Frame-Based Processing: Inherited Option of the Input Processing Parameter Now Provides a Warning	22-15
Logging Frame-Based Signals	22-16
Frame-Based Processing: Model Reference	22-16
Removing Mixed Frameness Support for Bus Signals on Unit Delay and Delay	22-17
Block Enhancements	22-18
Delay Block Accepts Buses and Variable-Size Signals at the Data Input Port	22-18
n-D Lookup Table Block Has New Default Settings	22-18
Blocks with Discrete States Can Specify Custom Storage Classes in the Dialog Box	22-18
Inherited Option of the Input Processing Parameter Now Provides a Warning	22-19
User Interface Enhancements	22-21
Model Advisor: Highlighting	22-21
Model Explorer: Grouping Enhancements	22-21
Model Explorer: Row Filter Button	22-21
Simulation Data Inspector Enhancements	22-22
Port Value Displays	22-23
Modeling Guidelines	22-24
Modeling Guidelines for High-Integrity Systems	22-24
MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow	22-24
Execution on Target Hardware	22-25

New Feature for Running Models Directly from Simulink on Target Hardware	22-25
--	-------

R2011b

Simulation Performance	23-2
Accelerator Mode Now Supports Algebraic Loops	23-2
Component-Based Modeling	23-3
For Each Subsystem Support for Continuous Dynamics	23-3
Enable Port as an Input to a Root-Level Model	23-3
Finder Option for Looking Inside Referenced Models	23-3
Improved Detection for Rebuilding Model Reference Targets	23-3
Model Reference Target Generation Closes Unneeded Libraries	23-4
Concurrent Execution Support	23-4
Finer Control of Library Links	23-4
Mask Built-In Blocks with the Mask Editor	23-4
Parameter Checking in Masked Blocks	23-5
Menu Options to Control Variants	23-5
MATLAB Function Blocks	23-6
Simulation Supported When the Current Folder Is a UNC Path	23-6
Simulink Data Management	23-7
Default Design Minimum and Maximum are []/[], Not -inf/inf	23-7
Bus Elements Now Have Design Minimum and Maximum Properties ...	23-7
Compiled Design Minimum and Maximum Values Exposed on Block Inport and Outport	23-7
Back-Propagated Minimum and Maximum of Portion of Wide Signal Are Now Ignored	23-8
Easier Importing of Signal Logging Data	23-8
Partial Specification of External Input Data	23-9
Command-Line Interface for Signal Logging	23-9
Access to the Data Import/Export Pane from the Signal Logging Selector	23-9
Inexact Property Names for User-Defined Data Objects Will Not Be Supported in a Future Release	23-9
Alias Types No Longer Supported with the slDataTypeAndScale Function	23-10
Simulink.StructType Objects Will Not Be Supported in a Future Release	23-10
Old Block-specific Data Type Parameters No Longer Supported	23-10
Simulink.Signal and Simulink.Parameter Will Not Accept Input Arguments	23-10
Data Import/Export Pane Changes	23-10
Simulation Data Inspector Tool Replaces Time Series Tool	23-11
Simulink File Management	23-12

Project Management	23-12
Simulink Signal Management	23-13
Signal Conversion Block Enhancements	23-13
Environment Controller Block Support for Non-Bus Signals	23-14
Sample Time Propagation Changes	23-14
Frame-Based Processing	23-14
Block Enhancements	23-16
New Delay Block That Upgrades the Integer Delay Block	23-16
Sqrt and Reciprocal Sqrt Blocks Support Explicit Specification of Intermediate Data Type	23-17
Discrete Zero-Pole Block Supports Single-Precision Inputs and Outputs	23-18
n-D Lookup Table Block Supports Tunable Table Size	23-18
Boolean Output Data Type Support for Logic Blocks	23-19
Derivative Block Parameter Change	23-20
User Interface Enhancements	23-21
Model Explorer: First Two Columns in Contents Pane Remain Visible ..	23-21
Model Explorer: Subsystem Code View Added	23-21
Model Explorer: New Context Menu Options for Model Configurations	23-21
Simulation Data Inspector Enhancements	23-22
Conversion of Error and Warning Message Identifiers	23-22
New Modeling Guidelines	23-24
Modeling Guidelines for High-Integrity Systems	23-24
Modeling Guidelines for Code Generation	23-24

R2011a

Simulation Performance	24-2
Restore SimState in Models Created in Earlier Simulink Versions	24-2
Improved Absolute Tolerance Implementation	24-2
Component-Based Modeling	24-3
Refreshing Linked Blocks and Model Blocks	24-3
Enhanced Model Block Displays Variant Model Choices	24-3
Creating a Protected Model Using the Simulink Editor	24-3
MATLAB Function Blocks	24-4
Embedded MATLAB Function Block Renamed as MATLAB Function Block	24-4
Support for Buses in Data Store Memory	24-4

Simulink Data Management	24-5
Signal Logging Selector	24-5
Dataset Format Option for Signal Logging Data	24-5
From File Block Supports Zero-Crossing Detection	24-6
Signal Builder Block Now Supports Virtual Bus Output	24-6
Signal Builder Block Now Shows the Currently Active Group	24-6
signalbuilder Function Change	24-6
Range-Checking Logic for Fixed-Point Data During Simulation Improved	24-6
Data Object Wizard Now Supports Boolean, Enumerated, and Structured Data Types for Parameters	24-7
Error Now Generated When Initialized Signal Objects Back Propagate to Output Port of Ground Block	24-7
No Longer Able to Set RTWInfo or CustomAttributes Property of Simulink Data Objects	24-8
Global Data Stores Now Treat Vector Signals as One or Two Dimensional	24-8
No Longer Able to Use Trigger Signals Defined as Enumerations	24-9
Conversions of Simulink.Parameter Object Structure Field Data to Corresponding Bus Element Type Supported for double Only	24-9
Simulink.CustomParameter and Simulink.CustomSignal Data Classes To Be Deprecated in a Future Release	24-10
Parts of Data Class Infrastructure No Longer Available	24-10
Simulink Signal Management	24-11
Data Store Support for Bus Signals	24-11
Accessing Bus and Matrix Elements in Data Stores	24-11
Array of Buses Support for Permute Dimensions, Probe, and Reshape Blocks	24-11
Using the Bus Editor to Create Simulink.Parameter Objects and MATLAB Structures	24-12
Block Enhancements	24-13
Lookup Table, Lookup Table (2-D), and Lookup Table (n-D) Blocks Replaced with Newer Versions in the Simulink Library	24-13
Magnitude-Angle to Complex Block Supports CORDIC Algorithm and Fixed-Point Data Types	24-16
Trigonometric Function Block Supports Complex Exponential Output ..	24-17
Shift Arithmetic Block Supports Specification of Bit Shift Values as Input Signal	24-18
Multiple Lookup Table Blocks Enable Removal of Range-Checking Code	24-18
Enhanced Dialog Layout for the Prelookup and Interpolation Using Prelookup Blocks	24-21
Product of Elements Block Uses a Single Algorithm for Element-Wise Complex Division	24-22
Sign Block Supports Complex Floating-Point Inputs	24-22
MATLAB Fcn Block Renamed to Interpreted MATLAB Function Block ..	24-22
Environment Controller Block Port Renamed from RTW to Coder	24-22
Block Parameters on the State Attributes Tab Renamed	24-22
Block Parameters and Values Renamed for Lookup Table Blocks	24-23
Performance Improvement for Single-Precision Computations of Elementary Math Operations	24-23

Dead Zone Block Expands the Region of Zero Output	24-23
Enhanced PID Controller Blocks Display Compensator Formula in Block Dialog Box	24-24
Ground Block Always Has Constant Sample Time	24-24
New Function-Call Feedback Latch Block	24-24
Outport Driving Merge Block Does Not Require IC in Simplified Initialization Mode	24-25
Discrete Filter, Discrete FIR Filter, and Discrete Transfer Fcn Blocks Now Have Input Processing Parameter	24-26
Model Blocks Can Now Use the GetSet Custom Storage Class	24-26
User Interface Enhancements	24-27
Model Explorer: Hiding the Group Column	24-27
Simulation Data Inspector Enhancements	24-27
Model Advisor	24-29
Configuration Parameters Dialog Box Changes	24-29
S-Functions	24-30
S-Functions Generated with legacy_code function and singleCPPMexFile S- Function Option Must Be Regenerated	24-30

R2010bSP2

Bug Fixes

R2010bSP1

Bug Fixes

R2010b

Simulation Performance	27-2
Elimination of Regenerating Code for Rebuilds	27-2
Component-Based Modeling	27-3
Model Workspace Is Read-Only During Compilation	27-3
Support for Multiple Normal Mode Instances of a Referenced Model ...	27-3
New Variant Subsystem Block for Managing Subsystem Design Alternatives	27-3
Support for Bus and Enumerated Data Types on Masks	27-4

sl_convert_to_model_reference Function Removed	27-4
Verbose Accelerator Builds Parameter Applies to Model Reference SIM Target Builds in All Cases	27-4
Embedded MATLAB Function Blocks	27-5
Specialization of Embedded MATLAB Function Blocks in Simulink Libraries	27-5
Support for Creation and Processing of Arrays of Buses	27-5
Ability to Include MATLAB Code as Comments in Generated C Code	27-5
Data Properties Dialog Box Enhancements	27-5
Simulink Data Management	27-7
Enhanced Support for Bus Objects as Data Types	27-7
Enhancements to Simulink.NumericType Class	27-8
Importing Signal Data Sets into the Signal Builder Block	27-8
signalbuilder Function Changes	27-8
From File Block Enhancements	27-9
Finding Variables Used by a Model or Block	27-9
enumeration Function Replaced With MATLAB Equivalent	27-9
Programmatic Creation of Enumerations	27-10
Simulink.Signal and Simulink.Parameter Objects Now Obey Model Data Type Override Settings	27-10
Simulink File Management	27-11
Autosave Upgrade Backup	27-11
Model Dependencies Tools	27-11
Simulink Signal Management	27-12
Arrays of Buses	27-12
Loading Bus Data to Root Input Ports	27-13
Block Enhancements	27-14
Prelookup Block Supports Dynamic Breakpoint Data	27-14
Interpolation Using Prelookup Block Supports Dynamic Table Data ...	27-14
Multiport Switch Block Supports Specification of Default Case for Out-of- Range Control Input	27-14
Switch Block Icon Shows Criteria and Threshold Values	27-14
Trigonometric Function Block Supports Expanded Input Range for CORDIC Algorithm	27-15
Repeating Sequence Stair Block Supports Enumerated Data Types	27-15
Abs Block Supports Specification of Minimum Output Value	27-15
Saturation Block Supports Logging of Minimum and Maximum Values for the Fixed-Point Tool	27-15
Vector Concatenate Block Now Appears in the Commonly Used and Signal Routing Libraries	27-15
Model Discretizer Support for Second-Order Integrator Block	27-15
Integer Delay and Unit Delay Blocks Now Have Input Processing Parameter	27-16
Data Store Read Block Sample Time Default Changed to -1	27-16
Support of Frame-Based Signals Being Removed From the Bias Block .	27-16
Relaxation of Limitations for Function-Call Split Block	27-17

User Interface Enhancements	27-18
Model Explorer and Command-Line Support for Saving and Loading Configuration Sets	27-18
Model Explorer: Grouping by a Property	27-18
Model Explorer: Filtering Contents	27-18
Model Explorer: Finding Variables That Are Used by a Model or Block	27-19
Model Explorer: Finding Blocks That Use a Variable	27-19
Model Explorer: Exporting and Importing Workspace Variables	27-20
Model Explorer: Link to System	27-20
Lookup Table Editor Can Now Propagate Changes in Table Data to Workspace Variables with Nonstandard Data Format	27-20
Enhanced Designation of Hybrid Sample Time	27-20
Inspect Solver Jacobian Pattern	27-20
Inspection of Values of Elements in Checksum	27-21
Conversion of Error and Warning Messages Identifiers	27-21
View and Compare Logged Signal Data from Multiple Simulations Using New Simulation Data Inspector Tool	27-21
Viewing Requirements Linked to Model Objects	27-21
S-Functions	27-22
Legacy Code Tool Support for Arrays of Simulink.Bus	27-22
S-Functions Generated with legacy_code function and singleCPPMexFile S- Function Option Must Be Regenerated	27-22
Level-2 M-File S-Function Block Name Changed to Level-2 MATLAB S- Function	27-22
Functions Removed	27-23
Function Being Removed in a Future Release	27-23

R2010a

Simulation Performance	28-2
Computation of Sparse and Analytical Jacobian for Implicit Simulink Solvers	28-2
Sparse Perturbation Support for RSim and Rapid Accelerator Mode	28-2
Increased Accuracy in Detecting Zero-Crossing Events	28-2
Saving Code Generated by Accelerating Models to slprj Folder	28-2
Component-Based Modeling	28-3
Defining Mask Icon Variables	28-3
For Each Subsystem Block	28-3
New Function-Call Split Block	28-3
Trigger Port Enhancements	28-4
Embedded MATLAB Function Blocks	28-5

New Ability to Use Global Data	28-5
Support for Logical Indexing	28-5
Support for Variable-Size Matrices in Buses	28-5
Support for Tunable Structure Parameters	28-5
Check Box for 'Treat as atomic unit' Now Always Selected	28-5
Simulink Data Management	28-6
New Function Finds Variables Used by Models and Blocks	28-6
MATLAB Structures as Tunable Structure Parameters	28-6
Simulink.saveVars Documentation Added	28-6
Custom Floating-Point Types No Longer Supported	28-6
Data Store Logging	28-7
Models with No States Now Return Empty Variables	28-7
To File Block Enhancements	28-8
From File Block Enhancements	28-8
Root Inport Support for Fixed-Point Data Contained in a Structure	28-9
Simulink Signal Management	28-10
Enhanced Support for Proper Use of Bus Signals	28-10
Bus Initialization	28-10
S-Functions for Working with Buses	28-11
Command Line API for Accessing Information About Bus Signals	28-11
Signal Name Propagation for Bus Selector Block	28-12
Block Enhancements	28-13
New Square Root Block	28-13
New Second-Order Integrator Block	28-13
New Find Nonzero Elements Block	28-13
PauseFcn and ContinueFcn Callback Support for Blocks and Block Diagrams	28-14
Gain Block Can Inherit Parameter Data Type from Gain Value	28-14
Direct Lookup Table (n-D) Block Enhancements	28-14
Multiport Switch Block Allows Explicit Specification of Data Port Indices	28-14
Trigonometric Function Block Supports CORDIC Algorithm and Fixed-Point Data Types	28-16
Enhanced Block Support for Enumerated Data Types	28-16
Lookup Table Dynamic Block Supports Direct Selection of Built-In Data Types for Outputs	28-17
Compare To Zero and Wrap To Zero Blocks Now Support Parameter Overflow Diagnostic	28-17
Data Type Duplicate Block Enhancement	28-18
Lookup Table and Lookup Table (2-D) Blocks To Be Deprecated in a Future Release	28-18
Elementary Math Block Now Obsolete	28-20
DocBlock Block RTF File Compression	28-21
Simulink Extras PID Controller Blocks Deprecated	28-21
User Interface Enhancements	28-22
Model Explorer Column Views	28-22
Model Explorer Display of Masked Subsystems and Linked Library Subsystems	28-22

Model Explorer Object Count	28-23
Model Explorer Search Option for Variable Usage	28-23
Model Explorer Display of Signal Logging and Storage Class Properties	28-23
Model Explorer Column Insertion Options	28-23
Diagnostics for Data Store Memory Blocks	28-23
New Command-Line Option for RSim Targets	28-24
Simulink.SimulationOutput.get Method for Obtaining Simulation Results	28-24
Simulink.SimState.ModelSimState Class has New snapshotTime Property	28-24
Simulink.ConfigSet.saveAs to Save Configuration Sets	28-24
S-Functions	28-25
Building C MEX-Files from Ada and an Example Ada Wrapper	28-25
New S-Function API Checks for Branched Function-Calls	28-25
New C MEX S-Function API and M-File S-Function Flag for Compliance with For Each Subsystem	28-25
Legacy Code Tool Enhanced to Support Enumerated Data Types and Structured Tunable Parameters	28-25
Documentation Improvements	28-27
Modeling Guidelines for High-Integrity Systems	28-27
MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow Included in Help	28-27

R2009bSP1

Bug Fixes

R2009b

Simulation Performance	30-2
Single-Output sim Syntax	30-2
Expanded Support by Rapid Accelerator	30-2
SimState Support in Accelerator Mode	30-2
Integer Arithmetic Applied to Sample Hit Computations	30-2
Improved Accuracy of Variable-Step Discrete Solver	30-2
Component-Based Modeling	30-4
Enhanced Library Link Management	30-4
Enhanced Mask Editor Provides Tabs and Signal Attributes	30-4
Model Reference Variants	30-4

Protected Referenced Models	30-4
Simulink Manifest Tools	30-5
S-Function Builder	30-5
Embedded MATLAB Function Blocks	30-6
Support for Variable-Size Arrays and Matrices	30-6
Change in Text and Visibility of Parameter Prompt for Easier Use with Fixed-Point Advisor and Fixed-Point Tool	30-6
New Compilation Report for Embedded MATLAB Function Blocks	30-6
New Options for Controlling Run-time Checks for Faster Performance ..	30-6
Embedded MATLAB Function Blocks Improve Size Propagation Behavior	30-7
Simulink Data Management	30-8
New Function Exports Workspace Variables and Values	30-8
New Enumerated Constant Block Outputs Enumerated Data	30-8
Enhanced Switch Case Block Supports Enumerated Data	30-8
Code for Multiport Switch Block Shows Enumerated Values	30-8
Data Class Infrastructure Partially Deprecated	30-8
Saving Simulation Results to a Single Object	30-8
Simulation Restart in R2009b	30-9
Removing Support for Custom Floating-Point Types in Future Release ..	30-9
Simulink File Management	30-10
Removal of Functions	30-10
Deprecation of SaveAs to R12 and R13	30-10
Improved Behavior of Save_System	30-10
Simulink Signal Management	30-11
Variable-Size Signals	30-11
Block Enhancements	30-12
New Turnkey PID Controller Blocks for Convenient Controller Simulation and Tuning	30-12
New Enumerated Constant Block Outputs Enumerated Data	30-12
Enhanced Switch Case Block Supports Enumerated Data	30-12
Code for Multiport Switch Block Shows Enumerated Values	30-12
Discrete Transfer Fcn Block Has Performance, Data Type, Dimension, and Complexity Enhancements	30-13
Lookup Table (n-D) Block Supports Parameter Data Types Different from Signal Data Types	30-13
Reduced Memory Use and More Efficient Code for Evenly Spaced Breakpoints in Prelookup and Lookup Table (n-D) Blocks	30-14
Math Function Block Computes Reciprocal of Square Root	30-14
Math Function Block Enhancements for Real-Time Workshop Code Generation	30-14
Relational Operator Block Detects Signals That Are Infinite, NaN, or Finite	30-15
Changes in Text and Visibility of Dialog Box Prompts for Easier Use with Fixed-Point Advisor and Fixed-Point Tool	30-15
Direct Lookup Table (n-D) Block Enhancements	30-16

Unary Minus Block Enhancements	30-17
Weighted Sample Time Block Enhancements	30-17
Switch Case Block Parameter Change	30-17
Signal Conversion Block Parameter Change	30-18
Compare To Constant and Compare To Zero Blocks Use New Default Setting for Zero-Crossing Detection	30-18
Signal Builder Block Change	30-18
User Interface Enhancements	30-19
Context-Sensitive Help for Simulink Blocks in the Continuous Library .	30-19
Adding Blocks from a Most Frequently Used Blocks List	30-19
Highlighting for Duplicate Inport Blocks	30-19
Using the Model Explorer to Add a Simulink.NumericType Object	30-19
Block Output Display Dialog Has OK and Cancel Buttons	30-20
Improved Definition of Hybrid Sample Time	30-20
Find Option in the Model Advisor	30-20

R2009a

Simulation Performance	31-2
Saving and Restoring the Complete SimState	31-2
Save Simulink Profiler Results	31-2
Component-Based Modeling	31-3
Port Value Displays in Referenced Models	31-3
Parallel Builds Enable Faster Diagram Updates for Large Model Reference Hierarchies In Accelerator Mode	31-3
Embedded MATLAB Function Blocks	31-5
Support for Enumerated Types	31-5
Use of Basic Linear Algebra Subprograms (BLAS) Libraries for Speed ..	31-5
Data Management	31-6
Signal Can Resolve to at Most One Signal Object	31-6
“Signed” Renamed to “Signedness” in the Simulink.NumericType class	31-6
“Sign” Renamed to “Signedness” in the Data Type Assistant	31-6
Tab Completion for Enumerated Data Types	31-7
Simulink File Management	31-8
Model Dependencies Tools	31-8
Block Enhancements	31-9
Prelookup and Interpolation Using Prelookup Blocks Support Parameter Data Types Different from Signal Data Types	31-9

Lookup Table (n-D) and Interpolation Using Prelookup Blocks Perform Efficient Fixed-Point Interpolations	31-9
Expanded Support for Simplest Rounding Mode to Maximize Block Efficiency	31-10
New Rounding Modes Added to Multiple Blocks	31-10
Lookup Table (n-D) Block Performs Faster Calculation of Index and Fraction for Power of 2 Evenly-Spaced Breakpoint Data	31-12
Discrete FIR Filter Block Supports More Filter Structures	31-12
Discrete Filter Block Performance, Data Type, Dimension, and Complexity Enhancements	31-12
MinMax Block Performs More Efficient and Accurate Comparison Operations	31-13
Logical Operator Block Supports NXOR Boolean Operator	31-13
Discrete-Time Integrator Block Uses Efficient Integration-Limiting Algorithm for Forward Euler Method	31-13
Dot Product Block Converted from S-Function to Core Block	31-13
Pulse Generator Block Uses New Default Values for Period and Pulse Width	31-14
Random Number, Uniform Random Number, and Unit Delay Blocks Use New Default Values for Sample Time	31-14
Trigonometric Function Block Provides Better Support of Accelerator Mode	31-14
Reshape Block Enhanced with New Input Port	31-14
Multidimensional Signals in Simulink Blocks	31-14
Subsystem Blocks Enhanced with Read-Only Property That Indicates Virtual Status	31-15
User Interface Enhancements	31-16
Port Value Displays in Referenced Models	31-16
Print Sample Time Legend	31-16
M-API for Access to Compiled Sample Time Information	31-16
Model Advisor Report Enhancements	31-16
Counterclockwise Block Rotation	31-16
Physical Port Rotation for Masked Blocks	31-16
Smart Guides	31-17
Customizing the Library Browser's User Interface	31-17
Subsystem Creation Command	31-17
Removal of Lookup Table Designer from the Lookup Table Editor	31-17
S-Functions	31-18
Level-1 Fortran S-Functions	31-18

R2008b

Simulation Performance	32-2
Parallel Simulations in Rapid Accelerator Mode	32-2
Improved Rebuild Mechanism in Rapid Accelerator Mode	32-2
Data Type Size Limit on Accelerated Simulation Removed	32-2

New Initialization Behavior in Conditional, Action, and Iterator Subsystems	32-2
Component-Based Modeling	32-4
Processor-in-the-Loop Mode in Model Block	32-4
Conditionally Executed Subsystem Initial Conditions	32-4
Model Block Input Enhancement	32-5
One Parameter Controls Accelerator Mode Build Verbosity	32-5
Embedded MATLAB Function Blocks	32-7
Support for Fixed-Point Word Lengths Up to 128 Bits	32-7
Enhanced Simulation and Code Generation Options for Embedded MATLAB Function Blocks	32-7
Data Type Override Now Works Consistently on Outputs	32-7
Improperly-Scaled Fixed-Point Relational Operators Now Match MATLAB Results	32-7
Data Management	32-9
Support for Enumerated Data Types	32-9
Simulink Bus Editor Enhancements	32-9
New Model Advisor Check for Data Store Memory Usage	32-9
Simulink File Management	32-10
Model Dependencies Tools	32-10
Block Enhancements	32-11
Trigonometric Function Block	32-11
Math Function Block	32-11
Merge Block	32-11
Discrete-Time Integrator Block	32-11
Modifying a Link to a Library Block in a Callback Function Can Cause Illegal Modification Errors	32-11
Random Number Block	32-11
Signal Generator Block	32-12
Sum Block	32-12
Switch Block	32-12
Uniform Random Number Block	32-12
User Interface Enhancements	32-13
Sample Time	32-13
Model Advisor	32-13
“What’s This?” Context-Sensitive Help for Commonly Used Blocks	32-13
Compact Icon Option Displays More Blocks in Library Browser	32-14
Signal Logging and Test Points Are Controlled Independently	32-14
Signal Logging Consistently Retains Duplicate Signal Regions	32-15
Simulink Configuration Parameters	32-15
Model Help Menu Update	32-16
Unified Simulation and Embeddable Code Generation Options	32-17
Mapping of Target Object Properties to Parameters in the Configuration Parameters Dialog Box	32-27

New Parameters in the Configuration Parameters Dialog Box for Simulation and Embeddable Code Generation	32-32
S-Functions	32-34
Ada S-Functions	32-34
Legacy Code Tool Enhancement	32-34
MATLAB Changes Affecting Simulink	32-35
Changes to MATLAB Startup Options	32-35
MATLAB Graphics Tools Not Supported Under -nojvm Startup Option .	32-35

R2008a

Simulation Performance	33-2
Rapid Accelerator	33-2
Additional Zero Crossing Algorithm	33-2
Component-Based Modeling	33-3
Efficient Parent Model Rebuilds	33-3
Scalar Root Inputs Passed Only by Reference	33-3
Unlimited Referenced Models	33-3
Embedded MATLAB Function Blocks	33-4
Nontunable Structure Parameters	33-4
Bidirectional Traceability	33-4
Specify Scaling Explicitly for Fixed-Point Data	33-4
Data Management	33-5
Array Format Cannot Be Used to Export Multiple Matrix Signals	33-5
Bus Editor Upgraded	33-5
Changing Nontunable Values Does Not Affect the Current Simulation . . .	33-5
Detection of Illegal Rate Transitions	33-5
Explicit Scaling Required for Fixed-Point Data	33-6
Fixed-Point Details Display Available	33-6
More than 2GB of Simulation Data Can be Logged on 64-Bit Platforms	33-6
Order of Simulink and MPT Parameter and Signal Fields Changed	33-6
Range Checking for Complex Numbers	33-7
Rate Transition Blocks Needed on Virtual Buses	33-7
Sample Times for Virtual Blocks	33-8
Signals Needing Resolution Are Graphically Indicated	33-8
Simulink File Management	33-9
Autosave	33-9
Old Version Notification	33-9

Model Dependencies Tools	33-9
Block Enhancements	33-10
New Discrete FIR Filter Block Replaces Weighted Moving Average Block	33-10
Rate Transition Block Enhancements	33-10
Enhanced Lookup Table (n-D) Block	33-10
New Accumulator Parameter on Sum Block	33-10
User Interface Enhancements	33-11
Simulink Library Browser	33-11
Simulink Preferences Window	33-11
Model Advisor	33-11
Solver Controls	33-11
“What’s This?” Context-Sensitive Help Available for Simulink Configuration Parameters Dialog	33-12
S-Functions	33-13

R2007b

Simulation Performance	34-2
Simulink Accelerator™	34-2
Simulink Profiler	34-2
Compiler Optimization Level	34-2
Variable-Step Discrete Solver	34-2
Referenced Models Can Execute in Normal or Accelerator Mode	34-2
Accelerator and Model Reference Targets Now Use Standard Internal Functions	34-3
Component-Based Modeling	34-4
New Instance View Option for the Model Dependency Viewer	34-4
Mask Editor Now Requires Java	34-4
Embedded MATLAB Function Blocks	34-5
Complex and Fixed-Point Parameters	34-5
Support for Algorithms That Span Multiple M-Files	34-5
Loading R2007b Embedded MATLAB Function Blocks in Earlier Versions of Simulink Software	34-5
Data Management	34-6
New Diagnostic for Continuous Sample Time on Non-Floating-Point Signals	34-6
New Standardized User Interface for Specifying Data Types	34-6
New Block Parameters for Specifying Minimum and Maximum Values ..	34-7
New Range Checking of Block Parameters	34-8
New Diagnostic for Checking Signal Ranges During Simulation	34-8

Configuration Management	34-9
Disabled Library Link Management	34-9
Model Dependencies Tools	34-9
Embedded Software Design	34-10
Legacy Code Tool Enhancement	34-10
Block Enhancements	34-11
Product Block Reorders Inputs Internally	34-11
Block Data Tips Now Work on All Platforms	34-11
Enhanced Data Type Support for Blocks	34-11
New Simulink Data Class Block Object Properties	34-11
New Break Link Options for save_system Command	34-12
Simulink Software Checks Data Type of the Initial Condition Signal of the Integrator Block	34-12
Usability Enhancements	34-13
Model Advisor	34-13
Alignment Commands	34-13
S-Functions	34-14
New S-Function APIs to Support Singleton Dimension Handling	34-14
New Level-2 M-File S-Function Example	34-14

R2007a+

Bug Fixes

R2007a

Multidimensional Signals	36-2
Multidimensional Signals in Simulink Blocks	36-2
Multidimensional Signals in S-Functions	36-4
Multidimensional Signals in Level-2 M-File S-Functions	36-4
New Block Parameters	36-4
GNU Compiler Upgrade	36-4
Changes to Concatenate Block	36-4
Changes to Assignment Block	36-5

Changes to Selector Block	36-5
Improved Model Advisor Navigation and Display	36-6
Change to Simulink.ModelAdvisor.getModelAdvisor Method	36-6
New Simulink Blocks	36-7
Change to Level-2 MATLAB S-Function Block	36-7
Model Dependency Analysis	36-7
Model File Monitoring	36-7
Legacy Code Tool Enhancements	36-7
Continuous State Names	36-8
Changes to Embedded MATLAB Function Block	36-9
New Function Checks M-Code for Compliance with Embedded MATLAB Subset	36-9
Support for Multidimensional Arrays	36-9
Support for Function Handles	36-9
Enhanced Support for Frames	36-9
New Embedded MATLAB Runtime Library Functions	36-9
Using & and Operators in Embedded MATLAB Function Blocks	36-11
Calling get Function from Embedded MATLAB Function Blocks	36-11
Documentation on Embedded MATLAB Subset has Moved	36-11
Referenced Models Support Non-Zero Start Time	36-12
New Functions Copy a Model to a Subsystem or Subsystem to Model	36-12
New Functions Empty a Model or Subsystem	36-12
Default for Signal Resolution Parameter Has Changed	36-12
Referencing Configuration Sets	36-13
New Block, Model Advisor Check, and Utility Function for Bus to Vector Conversion	36-13
Enhanced Support for Tunable Parameters in Expressions	36-14
New Loss of Tunability Diagnostic	36-14
Port Parameter Evaluation Has Changed	36-14
Data Type Objects Can Be Passed Via Mask Parameters	36-15
Expanded Options for Displaying Subsystem Port Labels	36-15

Model Explorer Customization Option Displays Properties of Selected Object	36-15
Change to PaperPositionMode Parameter	36-16
New Simulink.Bus.objectToCell Function	36-16
Simulink.Bus.save Function Enhanced To Allow Suppression of Bus Object Creation	36-16
Change in Version 6.5 (R2006b) Introduced Incompatibility	36-16
Nonverbose Output During Code Generation	36-16
SimulationMode Removed From Configuration Set	36-16

R2006b

Model Dependency Viewer	37-2
Enhanced Lookup Table Blocks	37-2
Legacy Code Tool	37-2
Simulink Software Now Uses Internal MATLAB Functions for Math Operations	37-2
Enhanced Integer Support in Math Function Block	37-3
Configuration Set Updates	37-3
Command to Initiate Data Logging During Simulation	37-3
Commands for Obtaining Model and Subsystem Checksums	37-3
Sample Hit Time Adjusting Diagnostic	37-4
Function-Call Models Can Now Run Without Being Referenced	37-4
Signal Builder Supports Printing of Signal Groups	37-4
Method for Comparing Simulink Data Objects	37-4
Unified Font Preferences Dialog Box	37-4
Limitation on Number of Referenced Models Eliminated for Single References	37-4
Parameter Objects Can Now Be Used to Specify Model Configuration Parameters	37-5

Parameter Pooling Is Now Always Enabled	37-5
Attempting to Reference a Symbol in an Uninitialized Mask Workspace Generates an Error	37-5
Changes to Integrator Block's Level Reset Options	37-6
Embedded MATLAB Function Block Features and Changes	37-6
Support for Structures	37-6
Embedded MATLAB Editor Analyzes Code with M-Lint	37-6
New Embedded MATLAB Runtime Library Functions	37-6
New Requirement for Calling MATLAB Functions from Embedded MATLAB Function Blocks	37-8
Type and Size Mismatch of Values Returned from MATLAB Functions Generates Error	37-8
Embedded MATLAB Function Blocks Cannot Output Character Data ...	37-9

R2006a+

No New Features or Changes

R2006a

Signal Object Initialization	39-2
Icon Shape Property for Logical Operator Block	39-2
Data Type Property of Parameter Objects Now Settable	39-2
Range-Checking for Parameter and Signal Object Values	39-2
Expanded Menu Customization	39-2
Bringing the MATLAB Desktop Forward	39-2
Converting Atomic Subsystems to Model References	39-3
Concatenate Block	39-3
Model Advisor Changes	39-3
Model Advisor Tasks Introduced	39-3
Model Advisor API	39-3
Built-in Block's Initial Appearance Reflects Parameter Settings	39-3
Double-Click Model Block to Open Referenced Model	39-4

Signal Logs Reflect Bus Hierarchy	39-4
Tiled Printing	39-4
Solver Diagnostic Controls	39-4
Diagnostic Added for Multitasking Conditionally Executed Subsystems	39-4
Embedded MATLAB Function Block Features and Changes	39-4
Option to Disable Saturation on Integer Overflow	39-4
Nontunable Option Allows Use of Parameters in Constant Expressions	39-5
Enhanced Support for Fixed-Point Arithmetic	39-5
Support for Integer Division	39-5
New Embedded MATLAB Runtime Library Functions	39-5
Setting FIMATH Cast Before Sum to False No Longer Supported in Embedded MATLAB Function Blocks	39-7
Type Mismatch of Scalar Output Data in Embedded MATLAB Function Blocks Generates Error	39-8
Implicit Parameter Type Conversions No Longer Supported in Embedded MATLAB Function Blocks	39-8
Fixed-Point Parameters Not Supported	39-9
Embedded MATLAB Function Blocks Require C Compiler for Windows 64	39-9

R14SP3

Model Referencing	40-2
Function-Call Models	40-2
Using Noninlined S-Functions in Referenced Models	40-2
Referenced Models Without Root I/O Can Inherit Sample Times	40-2
Referenced Models Can Use Variable Step Solvers	40-2
Model Dependency Graphs Accessible from the Tools Menu	40-2
Command That Converts Atomic Subsystems to Model References	40-2
Model Reference Demos	40-2
Block Enhancements	40-4
Variable Transport Delay, Variable Time Delay Blocks	40-4
Additional Reset Trigger for Discrete-Time Integrator Block	40-4
Input Port Latching Enhancements	40-4
Improved Function-Call Inputs Warning Label	40-4
Parameter Object Expressions No Longer Supported in Dialog Boxes ...	40-5
Modeling Enhancements	40-6
Annotations	40-6
Custom Signal Viewers and Generators	40-6
Model Explorer Search Option	40-6
Using Signal Objects to Assign Signal Properties	40-6

Bus Utility Functions	40-6
Fixed-Point Support in Embedded MATLAB Function Blocks	40-6
Embedded MATLAB Function Editor	40-7
Input Trigger and Function-Call Output Support in Embedded MATLAB Function Blocks	40-7
Find Options Added to the Data Object Wizard	40-7
Fixed-Point Functions No Longer Supported for Use in Signal Objects ..	40-7
Simulation Enhancements	40-8
Viewing Logged Signal Data	40-8
Importing Time-Series Data	40-8
Using a Variable-Step Solver with Rate Transition Blocks	40-8
Additional Diagnostics	40-8
Data Integrity Diagnostics Pane Renamed, Reorganized	40-8
Improved Sample-Time Independence Error Messages	40-8
User Interface Enhancements	40-10
Model Viewing	40-10
Customizing the Simulink User Interface	40-10
MEX-Files	40-11
MEX-Files on Windows Systems	40-11
MEX-File Extension Changed	40-11

R14SP2

Multiple Signals on Single Set of Axes	41-2
Logging Signals to the MATLAB Workspace	41-2
Legends that Identify Signal Traces	41-2
Displaying Tic Labels	41-2
Opening Parameters Dialog Box	41-2
Rootlevel Input Ports	41-2

R2022a

Version: 10.5

New Features

Bug Fixes

Version History

Simulink Editor

Simulink Fundamentals course

You can now access the self-paced, interactive Simulink® Fundamentals course that teaches you to:

- Use the Simulink environment.
- Model continuous and discrete dynamic systems.
- Organize growing models in subsystems that create visual and functional hierarchy.
- Componentize models by referencing other models and creating libraries.
- Optimize simulation performance.

Simulink Fundamentals uses tasks to teach concepts incrementally, such as through a real-life example with a 3D printer. You receive automated assessments and feedback after submitting tasks. Your progress is saved when you exit the course, so you can complete the course in multiple sessions.

The screenshot displays the Simulink Editor interface for a training task. On the left, the 'Training - Tasks' panel shows 'Task 1' with instructions: 'As you work through this module, you will build different representations of the nozzle logic system. The blocks at right represent most of an algorithm that compares both the current (simulated) X and Y position of the nozzle with the desired position. In this task, you will add the last piece of the logic: a check that an external control signal is true at the specified time steps.' Below this, it says 'In this training environment, clicking Submit will simulate and submit your model for grading. If you wish to simulate your model before grading, you can use the Run button in the Simulink toolbar.' A 'TASK' box contains the instruction: 'Complete the algorithm by adding a Logical Operator block from the Simulink library to your model. Leave the default Operator, AND, selected. Connect the block to the unconnected'.

The central workspace shows a Simulink model titled 'SimulinkFundamentals'. It features a 'Nozzle Algorithm' subsystem. Inputs include 'Reference signals' (desired_x, desired_y) and 'Simulated sensor data' (meas_x, meas_y). The algorithm calculates 'des_pos' and 'meas_pos', then uses an 'abs' block and a 'pos_err' block to determine 'pos_err'. A 'tol' block is also present. The output is 'nozzle_cmd', which is connected to a 'Select X Position' block. Red dashed arrows indicate unconnected outputs from the 'abs' and 'pos_err' blocks.

On the right, the 'Training - Assessment' panel lists requirements:

- ? Does the model contain the following block? Logical Operator
- ? Are the following blocks connected correctly? Logical Operator Scope
- ? Are the following blocks connected correctly? From Workspace Logical Operator
- ? Are the following blocks connected correctly? Compare To Constant Logical Operator


For more information, see [Simulink Fundamentals](#).

Learn tab provides access to in-product trainings


The Simulink Start Page now has a **Learn** tab that provides access to in-product trainings, such as **Simulink Onramp** and **Simulink Fundamentals**.

New
Examples
Learn


▼ Getting Started




Simulink Onramp
14 modules 2 hours




Stateflow Onramp
12 modules 2 hours



Control Design Onramp with Simu...
7 modules 1 hours



Simscape Onramp
9 modules 1.5 hours




Circuit Simulation Onramp
7 modules 2 hours


▼ Simulink Fundamentals Course [Table of Contents](#)

Comprehensive training on how to use Simulink, a graphical simulation tool for modeling dynamic physical systems. The course is self-paced and interactive, with hands-on exercises in Simulink.


Suggested prerequisite: Simulink Onramp or equivalent knowledge of Simulink.




Simulink Environment Overview
2 modules 2 hours




Modeling Dynamic Systems
2 modules 2 hours



Organizing Growing Models
2 modules 2.5 hours



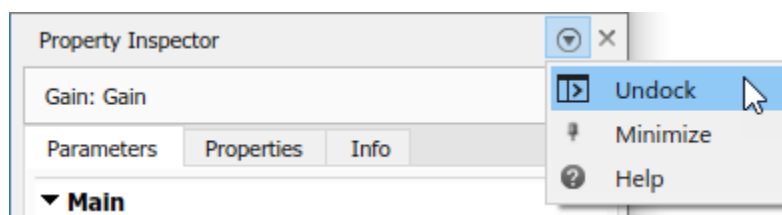
Componentizing Models
2 modules 2.5 hours



Optimizing Simulation Performance
1 modules 1 hours

Drop-down menu for Simulink Editor pane interactions

In Simulink Editor panes, a new button to the right of the pane title provides options to undock, minimize, and open help for the pane.



When a pane is not docked to the Simulink Editor, the button provides the option to dock and open help for the pane.

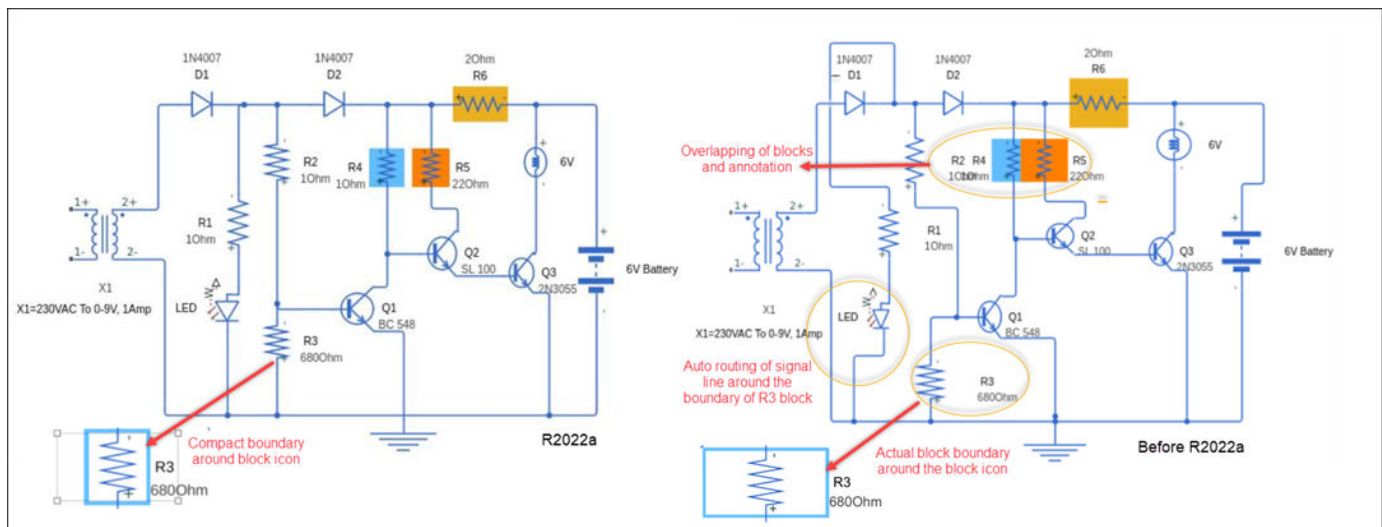
When a pane is minimized, the button provides the option to undock, restore, and open help for the pane.

Compact Boundaries for Simscape Blocks Improve Model Layout and Readability

Simscape™ blocks that have unused space around the block icon and do not have a solid boundary line now get a more compact boundary in block diagrams. This enhancement affects only the block display and enables these improvements:

- The signal lines pass near the new compact block boundary, which allows better autorouting of lines.
- Icon, badge, annotation, and action bar for the block appears near the compact boundary of the block. The block background color applies only within the compact boundary, which improves readability.
- Adjacent blocks align close to each other without overlap, which avoids truncation of content.

For example, see Resistor, Variable Resistor, and Capacitor blocks. The compact boundary improvements do not apply to blocks that have a solid boundary line, for example, PS Ramp and Gear Box blocks.



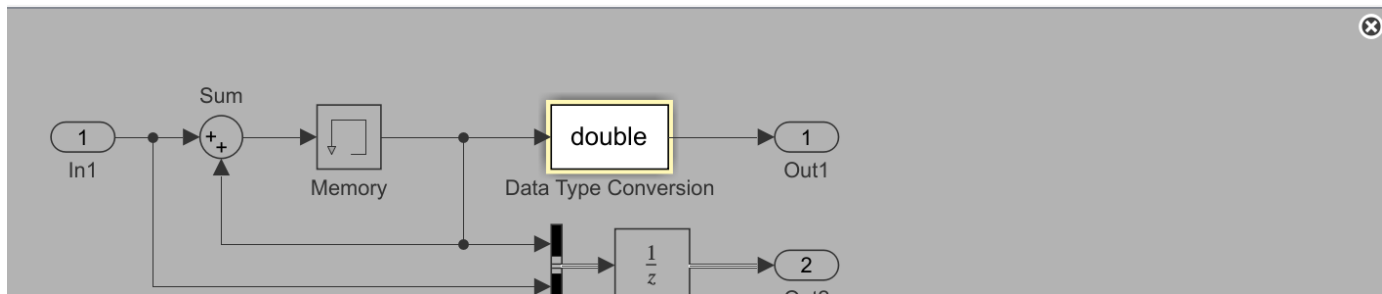
Version History

If you open a model created using a previous version of Simulink in R2022a and the model has compact blocks, then background color for such blocks is applied only within the compact boundary.

If you save a model created in R2022a to a previous version, the compact boundaries will be removed. Adjacent blocks can overlap, background color is applied to the complete block boundary, and signal lines can appear as passing through the block.

Highlight reduced nonvirtual blocks

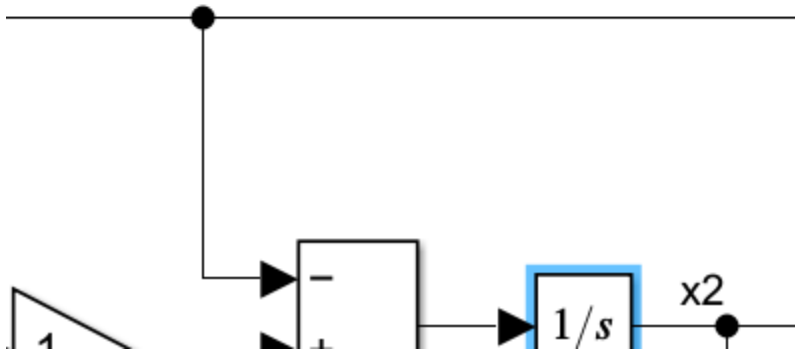
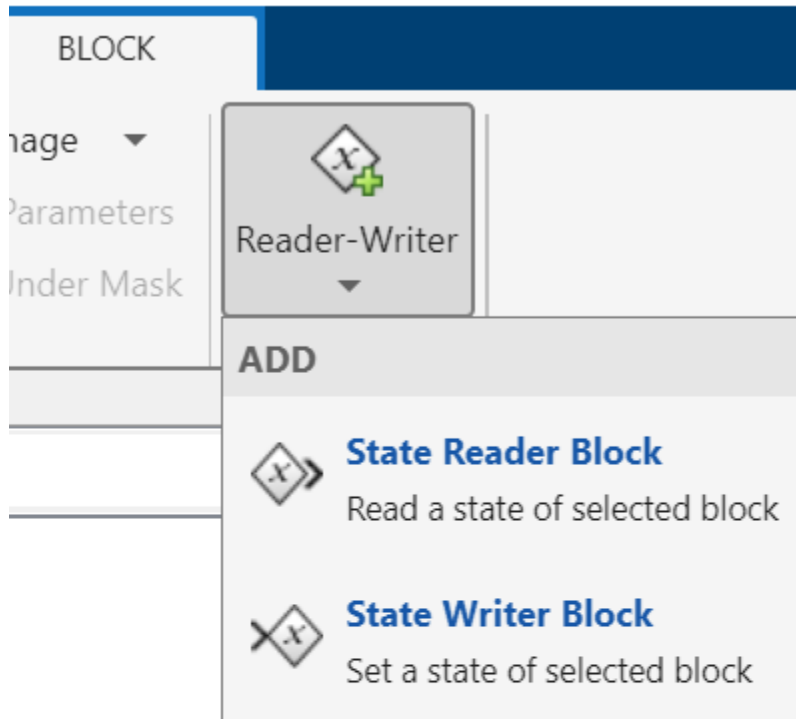
You can now highlight nonvirtual blocks that are removed to reduce execution time during model simulation and code generation. To enable this functionality, select the model configuration parameter **Block reduction**. For more information, see “Block reduction”.



Add State Reader, State Writer, and Parameter Writer blocks from Simulink toolstrip

You can now add State Reader, State Writer, and Parameter Writer blocks to your model directly from the Simulink toolstrip.

To add a State Reader or State Writer block, highlight by clicking on a block that has a state that the State Reader or State Writer block can read or write to. On the current block tab of the toolstrip, click **Reader-Writer** and select **State Reader Block** or **State Writer Block** from the list. A State Reader or State Writer block appears on the canvas with the **State owner block** parameter set to the block that was highlighted. The method of adding a State Reader or State Writer block by right-clicking and dragging a state owner block is no longer supported starting in R2022a.

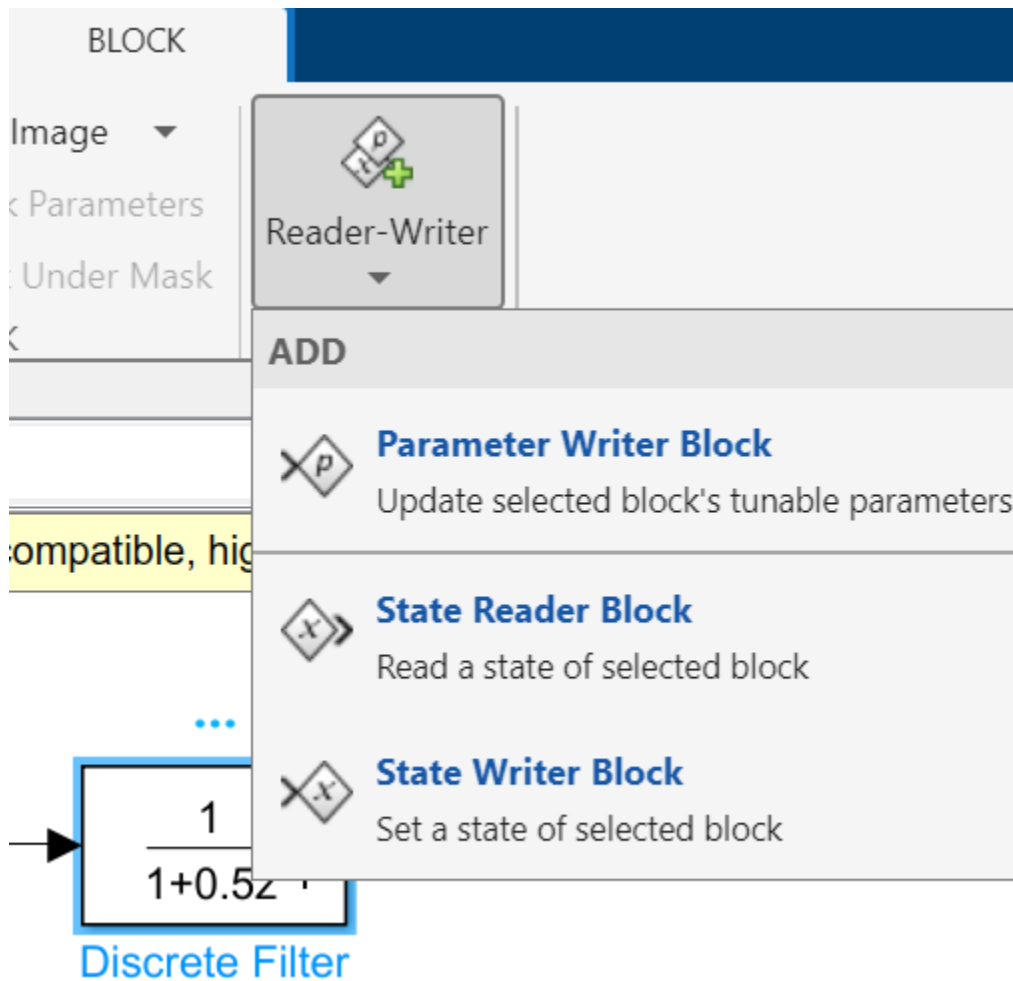


Note Depending on the type of block you highlight, the current block tab of the toolstrip might be labeled with the name of the highlighted block, for example, **Constant**, or it might be labeled **Block**.

Similarly, to add a Parameter Writer block, highlight by clicking on a block with a parameter that the Parameter Writer block can write to. Then, on the current block tab of the toolstrip, click **Parameter Writer**. A Parameter Writer block appears on the canvas with the **Parameter owner block** parameter set to the block that was highlighted.



If you highlight a block that has both a state that the State Reader or State Writer block can read or write to and a parameter that the Parameter Writer block can write to, on the current block tab of the toolstrip, click **Reader-Writer**. Select Parameter Writer Block, State Reader Block, or State Writer Block from the list.

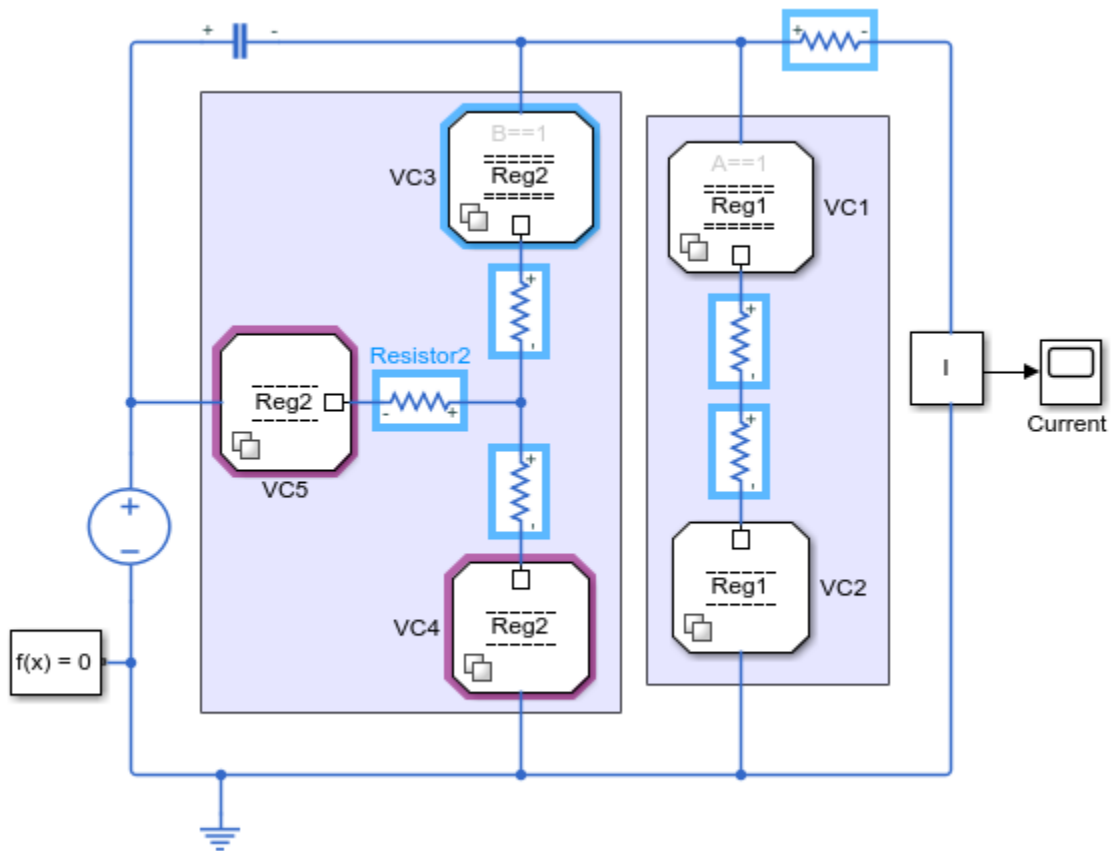


After you add a State Reader, State Writer, or Parameter Writer block, cut and paste it to move it to the desired location within the model hierarchy.

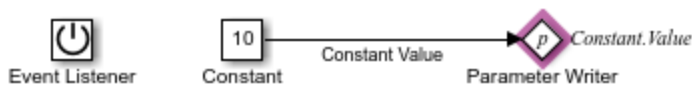
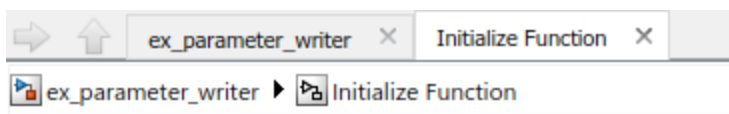
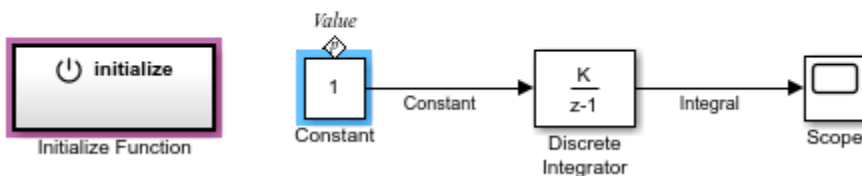
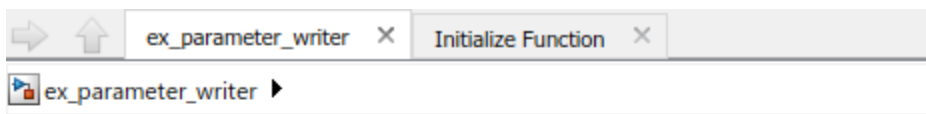
Related block highlighting for Variant Connector and Parameter Writer blocks

Starting in R2022a, selecting a primary Variant Connector block, a Parameter Writer block, or a block related to either of them, highlights all blocks that are related to the selected block.

A primary Variant Connector block and all nonprimary Variant Connector blocks with the same **Connector Tag** as that primary block constitute a set of related blocks.



A Parameter Writer block and the parameter owner block of the value that it writes constitute a set of related blocks.

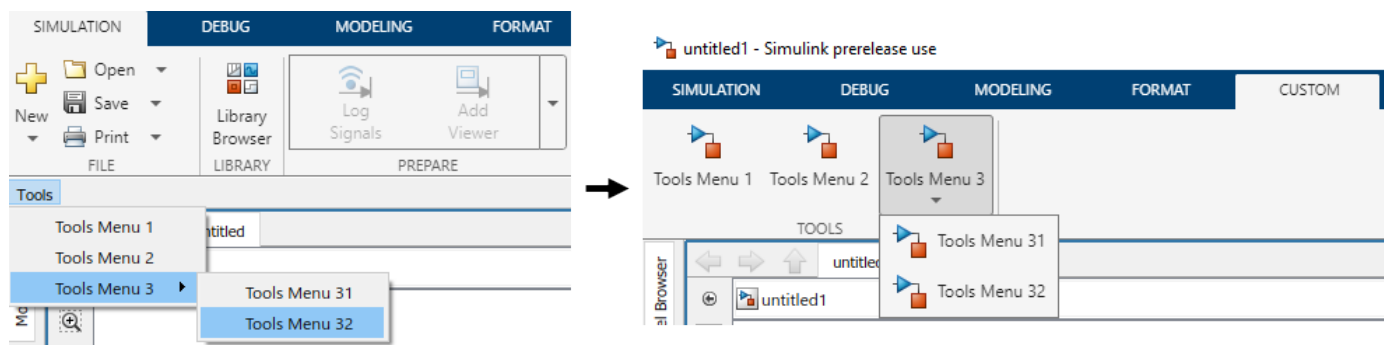


When a related block is highlighted, blocks in the current model that contain the related block are also highlighted. For example, an Initialize Function block is highlighted when it contains a Parameter Writer block that is related to the selected block.

Port toolbar menu customizations to custom tab in toolbar

A custom toolbar menu in the Simulink Editor can now be converted into a toolstrip tab using the `slConvertCustomMenus` command.

Converting a custom toolbar menu creates a toolstrip tab with a single section, which is populated by the converted menu items from the toolbar menu. List items from the menu are converted into push buttons, and list items with pop-up lists are converted into drop-down buttons.



The toolbar menu is not automatically deleted during the conversion. You can remove it by commenting out or deleting the corresponding `addCustomMenuFcn` call in the `sl_customization` file.

Any functionality that custom tabs do not support is ignored during the conversion. However, you can edit the generated tab. For more information, see “Create Custom Simulink Toolstrip Tabs”.

View and trace callbacks

You can view and trace all the callbacks executed in a model. Enable the option **Log Callbacks** to record all callbacks executed in the model. To view the callback report, go to **Diagnostics > Callback Tracer**.

For more information, see “View and Trace Callbacks in Model”

Use modelfinder to find Simulink models

Using `modelfinder` you can:

- Find models that match the specified string in model name, blocks, annotations, or description.

You can also find models that matches a specific search string and contains specified blocks. For example, you can search models that matches the search string `dc motor` in it and contains `pulse generator` and `pi controller` blocks.

- Index models in a specific path. Register or unregister a folder to include all the models in that folder so that they appear in the `modelfinder` search results.

For more information, see `modelfinder|modelfinder.registerFolder | modelfinder.unregisterFolder`

Unified breakpoint list in the Simulink Editor for debugging


In R2022a, you can use the new breakpoint list in the Simulink Editor to manage breakpoints set in a model. The unified breakpoint list includes useful metadata and tools for navigating to, modifying, organizing, sorting, and searching the breakpoints in the list. The breakpoint list includes different types of breakpoints that you can set, for example, Stateflow® and MATLAB® Function blocks.

Toolstrip-based UI for Model Advisor

The Model Advisor user interface now includes simplified toolstrip with new features.

- **Filter Checks** - Filters the checks based on their respective result statuses, such as Failed, Passed, Justified.
- **Justify** - Justify the violations. Justifications allow you to add a rationale for violations observed during Model Advisor analysis. For more information, see “Justify Violated Blocks from the Model Advisor Check Analysis” (Simulink Check).
- **Fix** - Fixes the violations by setting the violated parameter values to recommended values.
- **Reports** - Export Model Advisor analysis reports in HTML, PDF, and DOCX formats.
- **Manage Configurations** - Create, load, restore, and associate configurations in simple workflows.

This table describes the changes to the menu items in the Model Advisor:

Previous UI Element	New UI Element	Description of the Change
Run Selected Checks	Run Checks	No change in functionality. Use this button to run selected checks.
Settings > (options)	Open > (options)	Some older options are no longer available. Use this button to open or customize Model Advisor Configuration Editor .
Generate Report...	Report > (options)	No change in functionality. Use this button to generate the Model Advisor analysis report in HTML format. Use the drop-down option to select PDF or WORD .
Configure Hidden input parameters	Configure input parameters in Model Advisor Configuration Editor	Configure hidden input parameters was a hyperlink in the check window. In R2022a, you can now click the Configure input parameters in Model Advisor Configuration Editor icon  .

Previous UI Element	New UI Element	Description of the Change
Edit > Send Check IDs to Workspace	Right-click on any check, and select Send Check IDs to Workspace .	Use this option to send check IDs to workspace.
Edit > Send Instance IDs to Workspace	Right-click on any check, and select Send Instance IDs to Workspace .	Use this option to send instance IDs to workspace.

This table describes the navigation options removed from Model Advisor:

Navigation Options Removed
Settings > Treat as referenced model
Settings > Preferences
Edit > Reset
Switch to Model Advisor Dashboard
Run Checks in Background
Highlighting check results
Highlight exclusion results
Highlighting

For more information, see “Run Model Advisor Checks and Review Results” (Simulink Check).

Functionality being removed or changed

Toolbar menu customization will be removed

Still runs

Toolbar menu customization will be removed in a future release. Use custom toolstrip tabs instead.

Custom tabs better integrate into the Simulink Toolstrip. Menu items that were previously hidden within the toolbar menus are directly available on custom tabs. For information on custom tabs, see “Create Custom Simulink Toolstrip Tabs”.

To migrate your toolbar menu customization, see “Port toolbar menu customizations to custom tab in toolstrip” on page 1-10.

New simulation status during initialization of fast restart simulation

Behavior change

Starting in R2022a, fast restart simulations clearly indicate when the simulation is restarting. While fast restart is in progress, the simulation status becomes `Restarting`, and the Simulink Editor becomes frozen like it is during model compilation and initialization.

In previous releases, the Simulink Editor and simulation status did not give an indication that fast restart was in progress. The table summarizes updates to messages in the Simulink Editor and to the `SimulationStatus` model configuration parameter value.

Simulation Status Information while Fast Restart is in Progress	Before R2022a	Starting in R2022a
Simulation status message in the bottom left of the Simulink Editor	Fast Restart (Ready)	Restarting
Model status in the bottom right of the Simulink Editor before simulation starts	Initialized	Compiled
SimulationStatus configuration parameter value	compiled	restarting

Models or scripts that conditionalize behavior based on the value of the `SimulationStatus` parameter might behave differently or give an error during fast restart simulations.

Simulation Analysis and Performance

Local solver

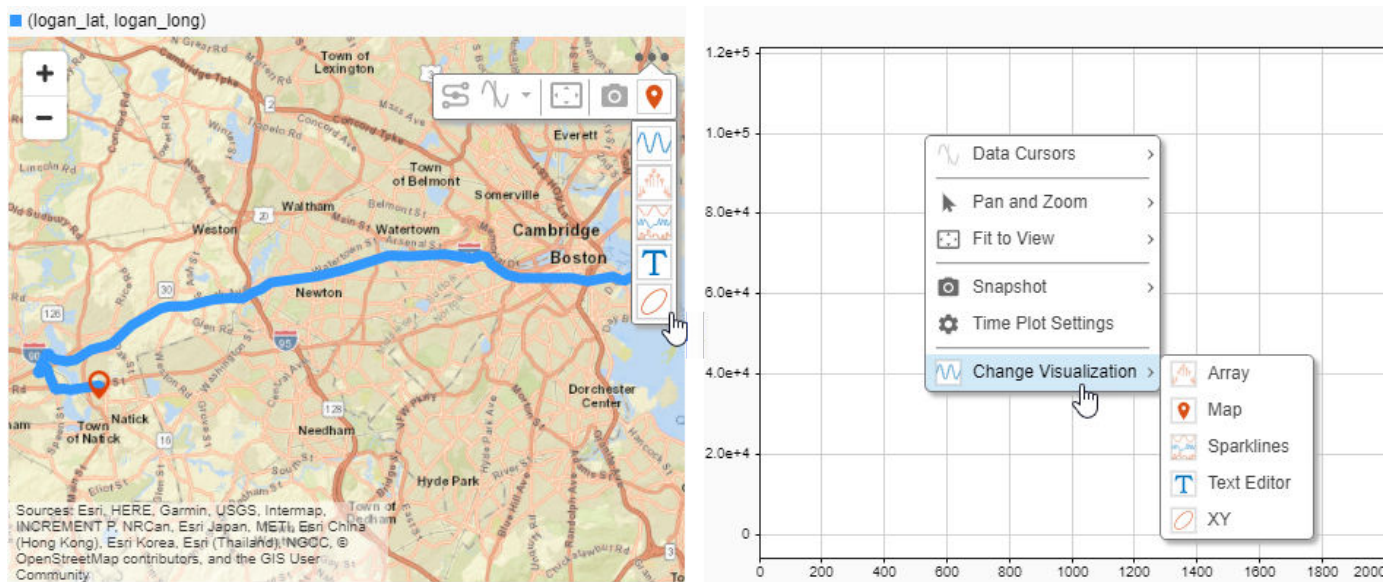
Starting in R2022a, you can speed up simulation in model references by using the local solver to choose different tradeoffs for different sections of a model, depending on your needs. This new feature lets you choose less expensive solvers for slower models.

Zero-Crossing detection using Fixed-Step Solver

Starting in R2022a, you can use fixed-step solvers to detect zero-crossings and locate events based on continuous zero-crossing signals that register events.

Change visualization for a subplot more easily in the Simulation Data Inspector

Starting in R2022a, the subplot menu and the context menu for subplots in the Simulation Data Inspector include an option to change the visualization for that subplot. Previously, to change the visualization for a subplot, you could only use the Visualization Gallery in the Simulation Data Inspector.

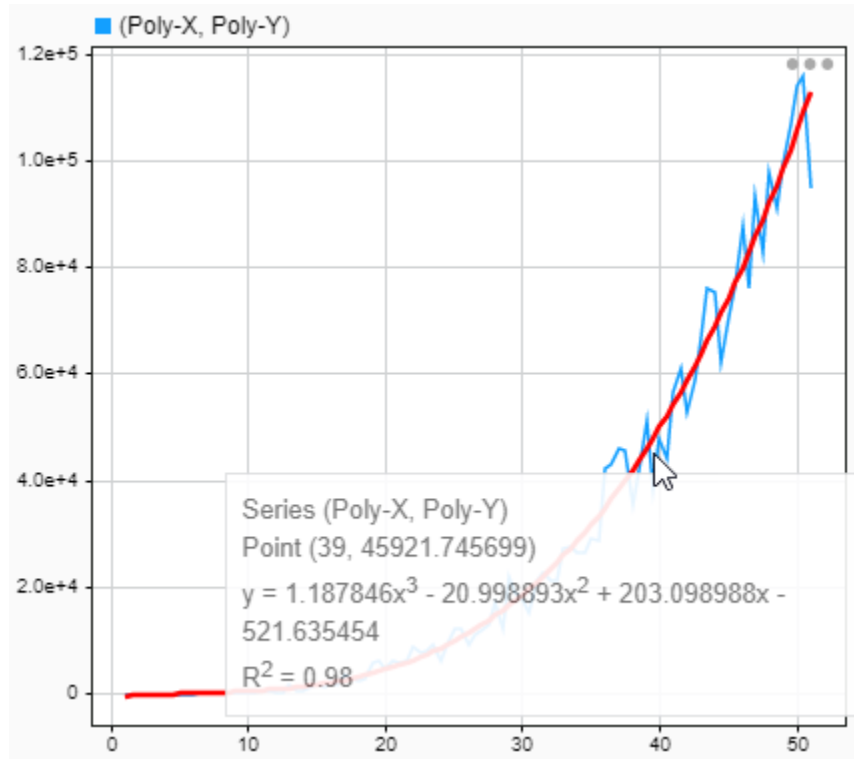


Changing the visualization for several subplots in the Simulation Data Inspector is also easier. The Visualization Gallery now remains open until you close it, and you can freely position the window within the plot area.

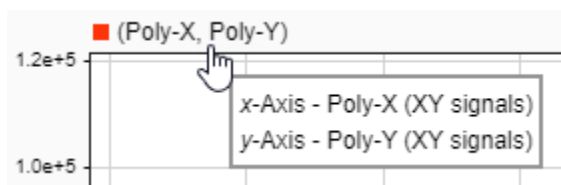
XY, sparklines, and map visualization enhancements

Several visualizations you can use in the Record block and Simulation Data Inspector have enhancements in R2022a:

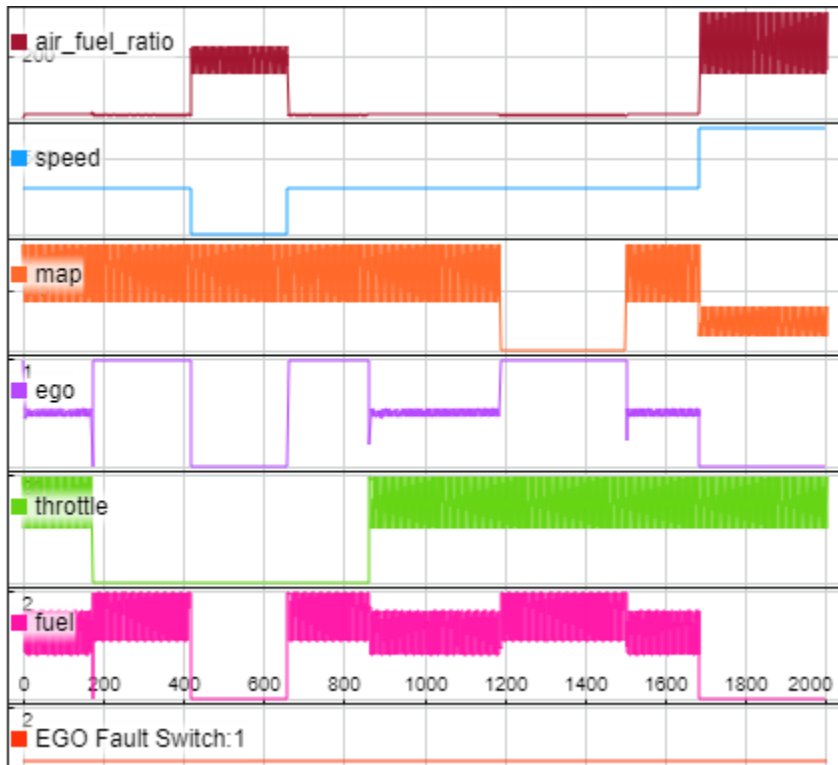
- When you add a trend line to an XY plot, the trend line now has a tooltip that shows:
 - The series and signals to which the trend line was fitted.
 - The x- and y-coordinates for the point nearest the pointer.
 - The trend line equation and R-squared value.



- The legend for the XY plot has a tooltip that indicates which run contains the plotted signal data.



- Sparklines plots now show tick labels for the time axis only on the last sparkline with a visible time axis. You can control whether tick labels are shown for all sparklines or only the last sparkline using the **Visualization Settings**.



- Maps now show a legend that indicates which signals provide the latitude and longitude data for the plotted route. The legend has a tooltip that indicates which run contains the plotted data.



Interactive comparison report enhancements

Interactive comparison reports created using the Simulation Data Inspector have these enhancements:

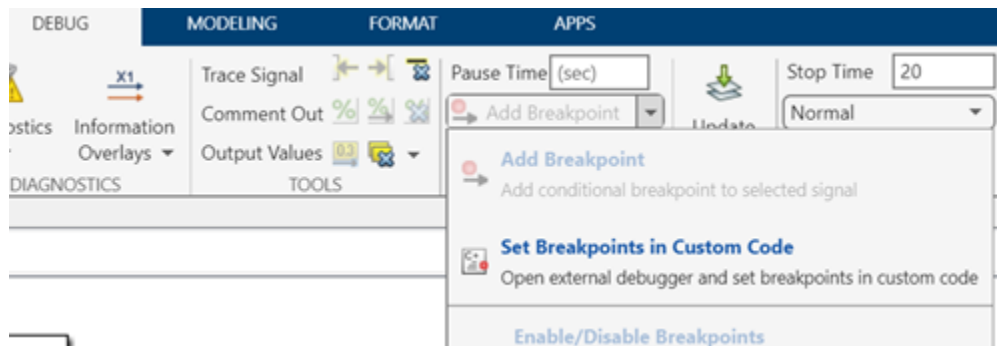
- You can sort comparison results in the report by name, absolute tolerance, relative tolerance, maximum difference, and result.
- The generated comparison report reflects the grouping in the Simulation Data Inspector at the time the report was generated. For example, when you group the results by data hierarchy, the results are grouped by data hierarchy in the generated report.

External debugger integration for debugging custom C/C++ code

You can now debug your custom C/C++ code from within Simulink by launching an external debugger and setting breakpoints in your custom code. The following debuggers are supported:

- On Windows®, Microsoft® Visual Studio® debugger
- On Linux®, GNU® Data Display Debugger (DDD)
- On Mac OS, LLDB Debugger (LLDB)

To launch the external debugger, on the **Debug** tab of the Simulink toolstrip, in the **Breakpoints** section, click the arrow to the right of the **Add Breakpoint** button. From the menu, select **Set Breakpoints in Custom Code**.



For more information, see “Debug Custom C/C++ Code”.

Parameter Combinations in Multiple Simulations panel

In R2022a, you can use Parameter Combinations in the Multiple Simulations panel of the Simulink Editor for workflows with multiple simulations, such as Monte-Carlo simulations and parameter sweeps. Parameter Combinations allows you to create sequential and exhaustive combinations of parameters, specify value ranges, and run simulations with these combinations

Functionality being removed or changed

Simulink.sdi.setRunOverwrite function has been removed

Errors

In R2022a, the `Simulink.sdi.setRunOverwrite` function has been removed. Using the function in a script causes an error.

In R2018b, the `Simulink.sdi.setRunOverwrite` function was replaced. Since R2018b, the `Simulink.sdi.setRunOverwrite` function has had no effect on the Simulation Data Inspector, and scripts that use the function have generated a warning.

To retain data for only a single run in the Simulation Data Inspector, use the `Simulink.sdi.setAutoArchiveMode` function and the `Simulink.sdi.setArchiveRunLimit` function. Configure the Simulation Data Inspector to automatically move runs into the archive, and set the archive run limit to 0.

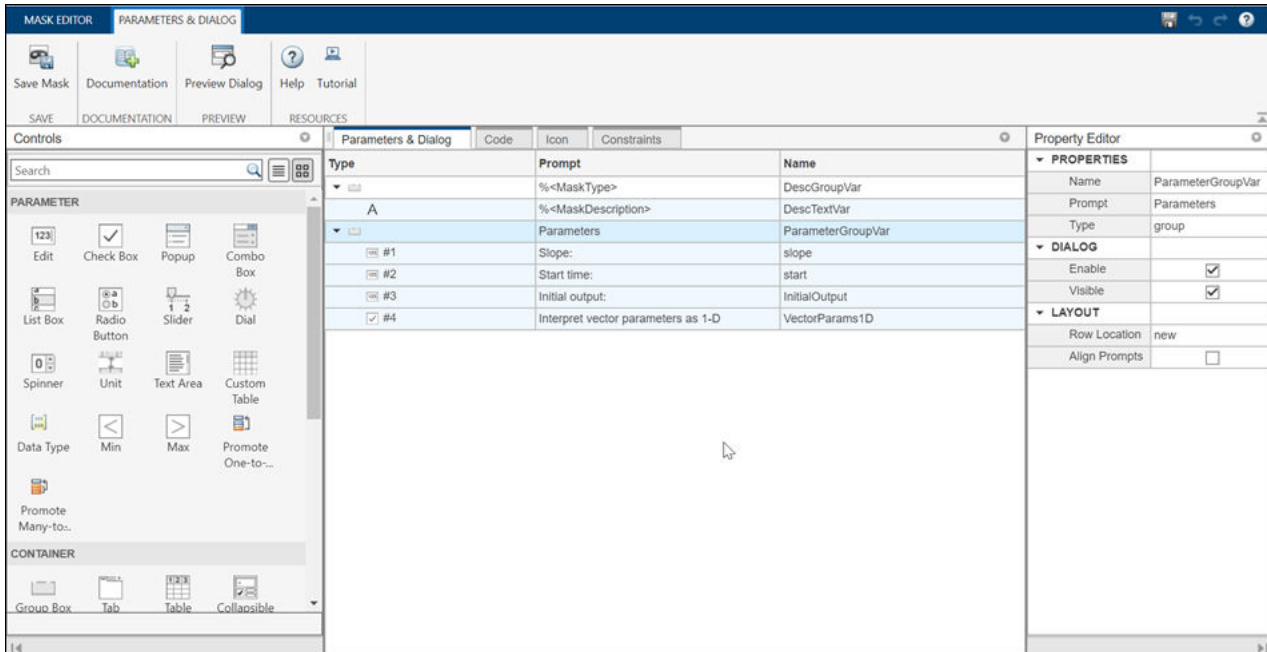
```
Simulink.sdi.setAutoArchiveMode(true);
Simulink.sdi.setArchiveRunLimit(0);
```

For more information about configuring the Simulation Data Inspector archive, see “Archive Behavior and Run Limit”.

Component-Based Modeling

Mask Editor streamlines masking workflow for Parameter Constraint, Port Constraint, and Parameter Promotion

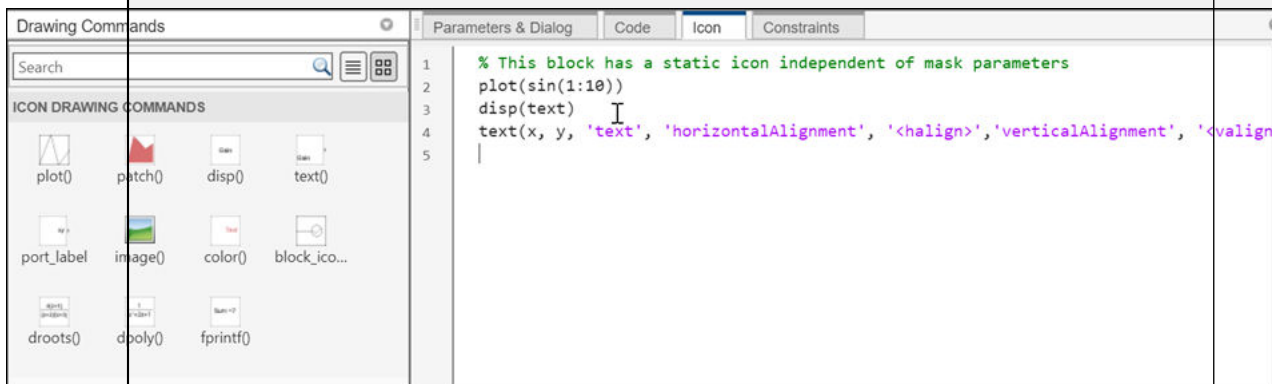
In R2022a, the Mask Editor has new capabilities that enable you to perform masking operations with fewer clicks.



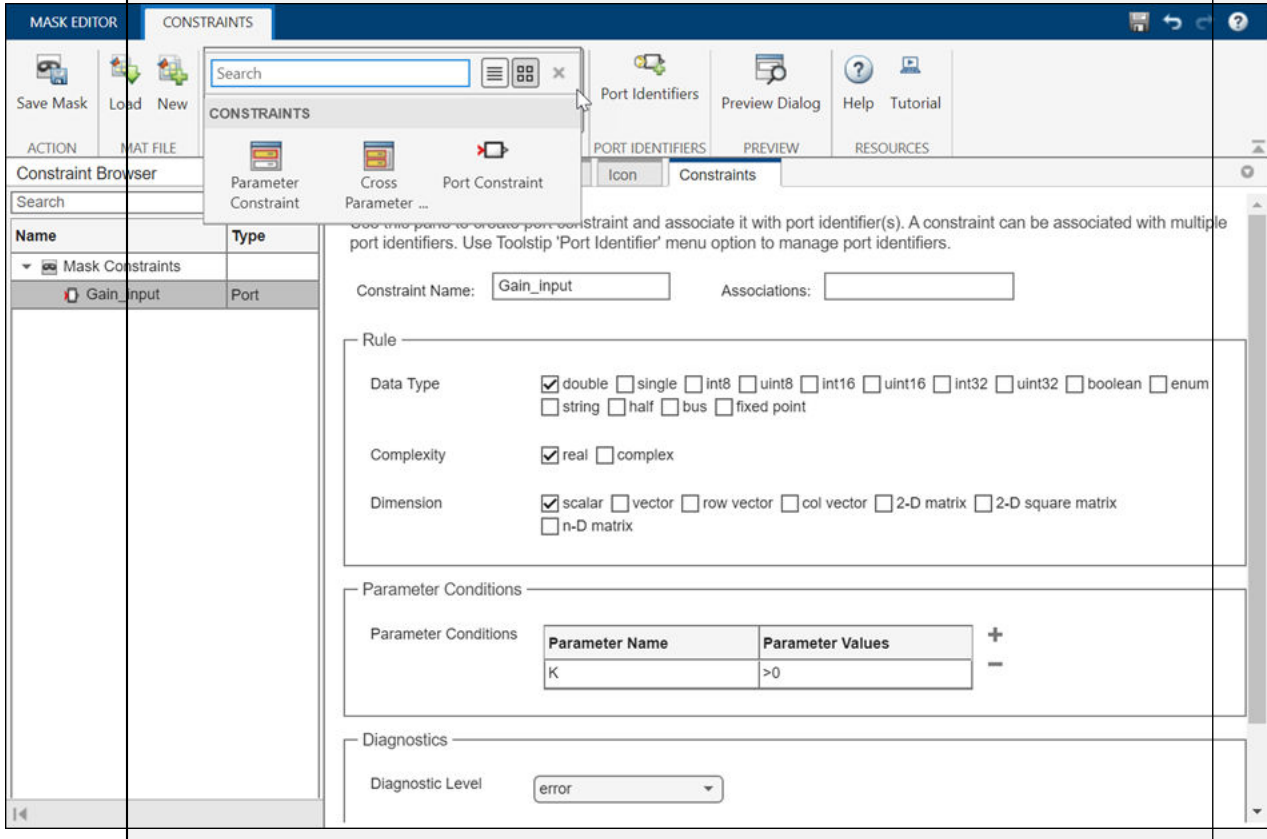
This table lists the new and improved capabilities in Mask Editor.

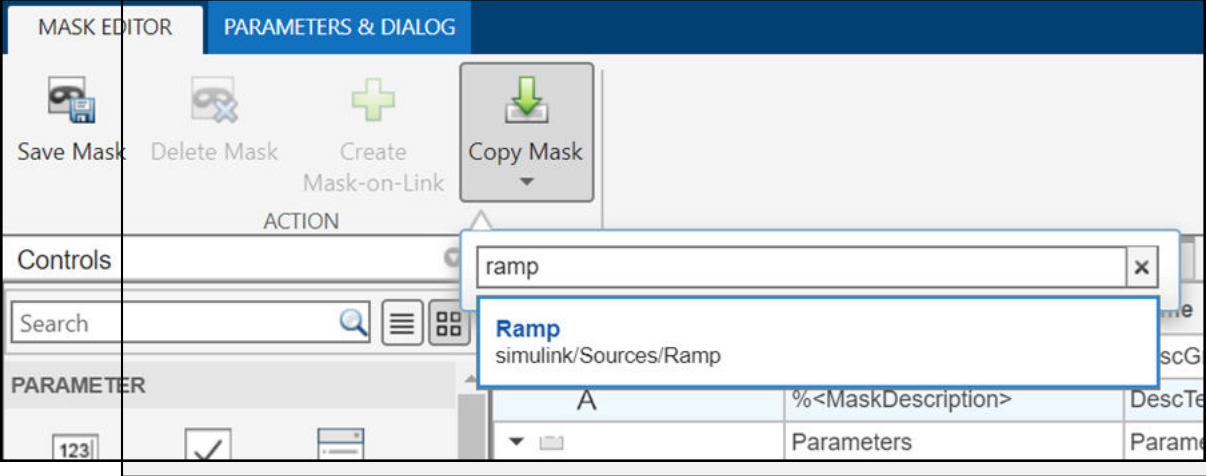
New and Improved Mask Editor Features

Feature	Improved or New?
Context-based tool strip for easy navigation - The menu options appear based on context. For example, the options for Parameter Constraint , Cross Parameter Constraint , and Port Constraint appear only when you select the Constraint tab.	New
Integrated view of initialization code and callback code - You can view the mask initialization code and callback code in a single integrated MATLAB Editor. Previously, the initialization code and callback code were in different locations. The Mask Editor code functionalities are like those in the MATLAB Editor, with some limitations. For example, the autocomplete functionality is supported, but you cannot set a breakpoint in your code.	New
Upload image within Mask Editor - You can set an image for the masked block icon within the Mask Editor.	New
Icon drawing commands - The skeleton for drawing commands appears when you click the commands.	New

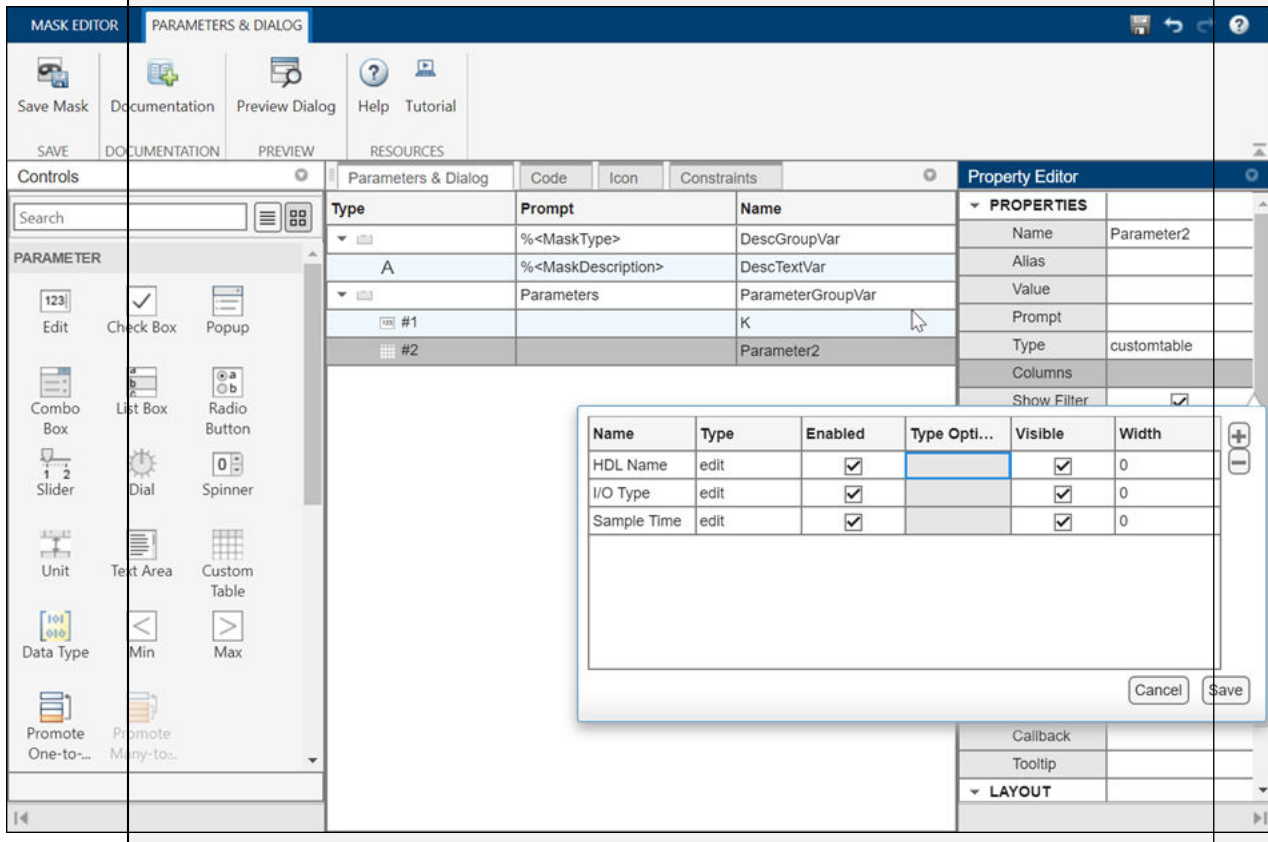


Feature	Improved or New?
<p>New Constraint Manager with support for port constraints - Constraint Manager is integrated inside the Mask Editor.</p> <ul style="list-style-type: none"> • Manage shared constraints with the Constraint Browser. • Specify constraints on the input and output ports of a masked block. The port attributes get checked against the constraints when you compile the model. For more information, see “Validate Input and Output Port Signals Using Port Constraints” 	<p>New</p>

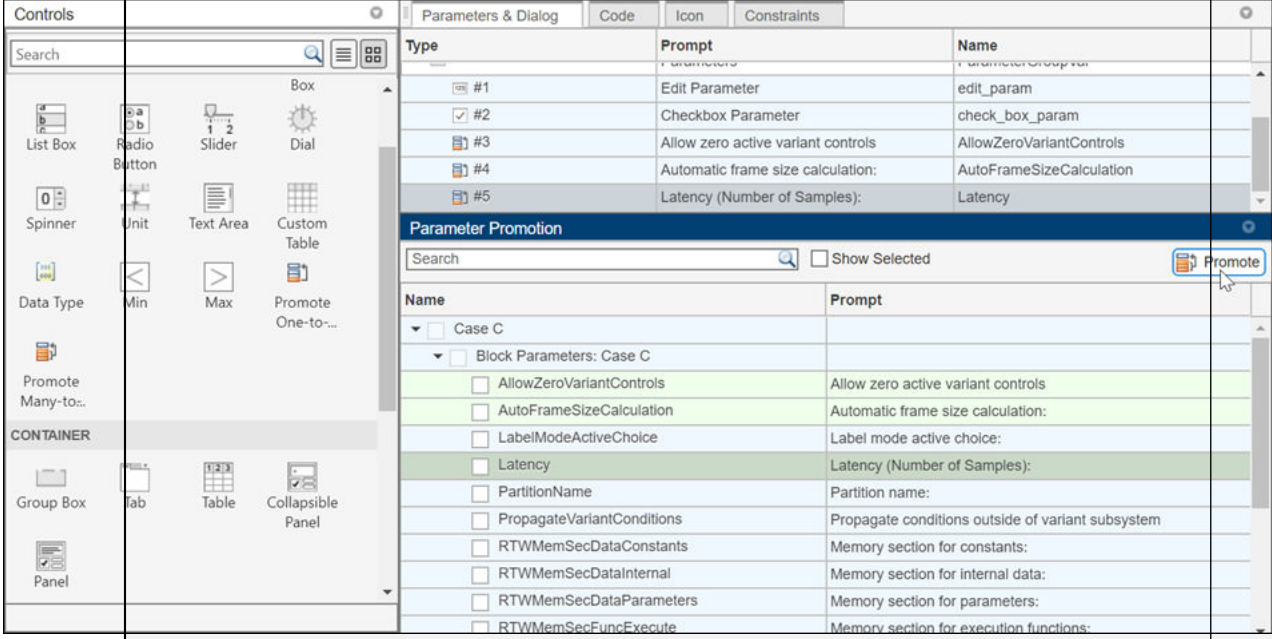


Feature	Improved or New?						
Copy mask definitions from Simulink library blocks - Search for the desired block and click Copy Mask to import the mask definition from an existing block.	New						
 <p>The screenshot displays the 'MASK EDITOR' window with the 'PARAMETERS & DIALOG' tab selected. The 'ACTION' bar contains icons for 'Save Mask', 'Delete Mask', 'Create Mask-on-Link', and 'Copy Mask'. A search dropdown is open, showing 'ramp' in the input field and a result for 'Ramp' with the path 'simulink/Sources/Ramp'. Below the search, a table is partially visible with columns for parameter names and descriptions.</p> <table border="1" data-bbox="592 808 1317 894"><thead><tr><th data-bbox="592 808 893 840">A</th><th data-bbox="893 808 1218 840">%<MaskDescription></th><th data-bbox="1218 808 1317 840">DescTe</th></tr></thead><tbody><tr><td data-bbox="592 840 893 894">Parameters</td><td data-bbox="893 840 1218 894">Parameters</td><td data-bbox="1218 840 1317 894">Parame</td></tr></tbody></table>		A	%<MaskDescription>	DescTe	Parameters	Parameters	Parame
A	%<MaskDescription>	DescTe					
Parameters	Parameters	Parame					

Feature	Improved or New?
<p>Enhanced Property Editor for parameters - The new Property Editor allows you to set values for parameters such as Tree, LUT Widget, Tunable Popup, Custom Table, and Text Area. For example, you can use the Property Editor to specify the column properties of a custom table. Previously, you had to set values in text form using delimiters.</p>	Improved



Feature	Improved or New?
<p>Enhanced Parameter Promotion workflow - You can now promote parameters in fewer steps. An integrated view allows you to see all promotable parameters of a block. You can select multiple parameters to promote in the case of many-to-one parameter promotion. For more information, see “Promote Block Parameters on a Mask”</p>	Improved



The screenshot shows a software interface with a 'Controls' panel on the left and a 'Parameters & Dialog' panel on the right. The 'Parameters & Dialog' panel has tabs for 'Parameters', 'Code', 'Icon', and 'Constraints'. The 'Parameters' tab is active, showing a table with columns 'Type', 'Prompt', and 'Name'. Below this table is a 'Parameter Promotion' dialog with a search bar, a 'Show Selected' checkbox, and a 'Promote' button. The dialog also contains a table with columns 'Name' and 'Prompt', listing various parameters like 'AllowZeroVariantControls', 'AutoFrameSizeCalculation', etc., each with a checkbox.

Specify port constraints on masked block

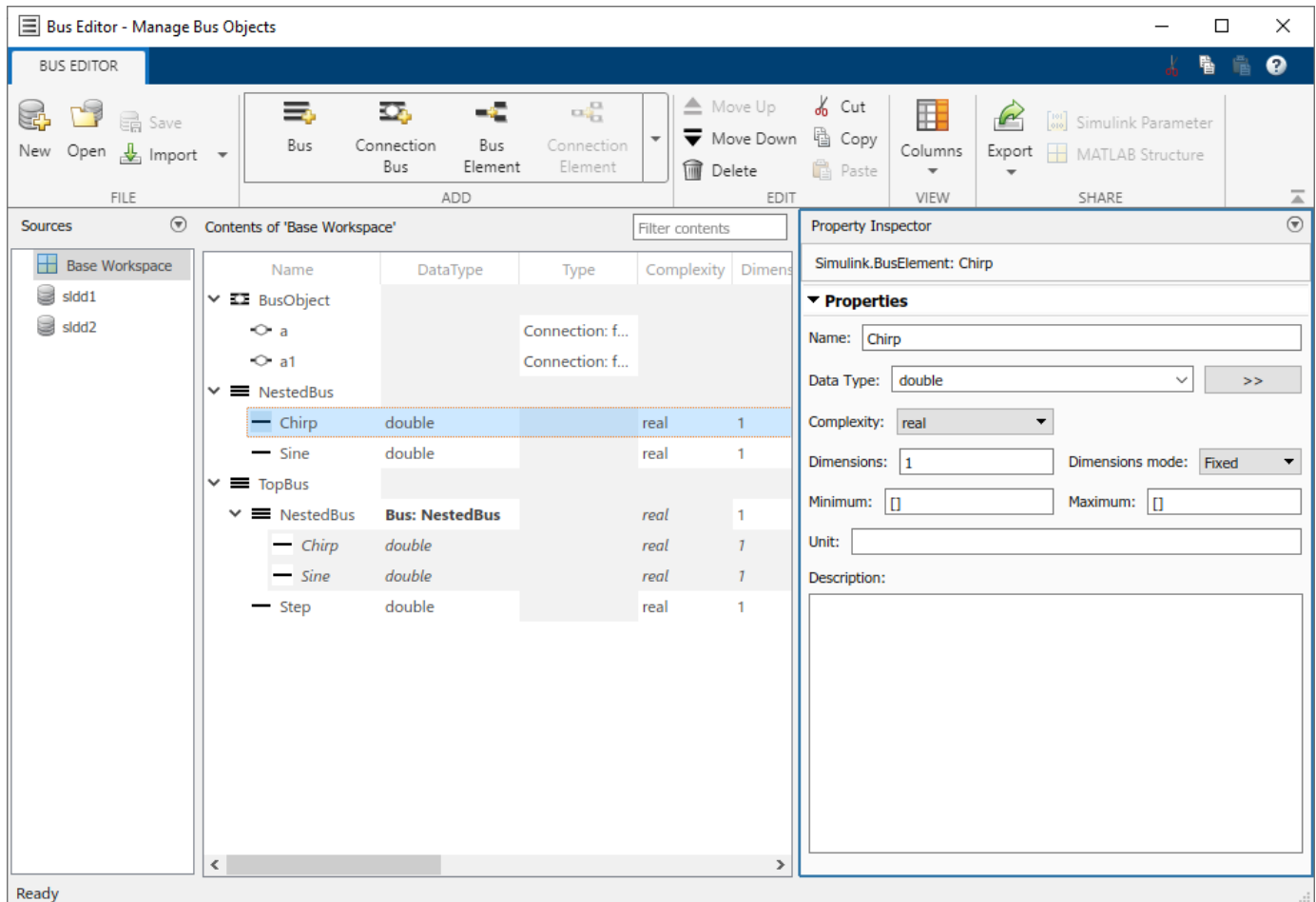
Starting in 2022a onward, you can specify constraints on the input and output ports of a masked block. The port attributes are checked against the constraints when you compile the model.

For more information, see “Validate Input and Output Port Signals Using Port Constraints”.

Bus Editor enhancements

The **Bus Editor** loads faster and has an updated interface for creating, modifying, and managing bus objects.

- A toolbar provides easy access to actions.
- A **Sources** pane provides the available object sources.
- An interactive table displays editable information about the objects, such as hierarchy and properties.
- A **Property Inspector** pane lets you focus on one object at a time and edit its properties.

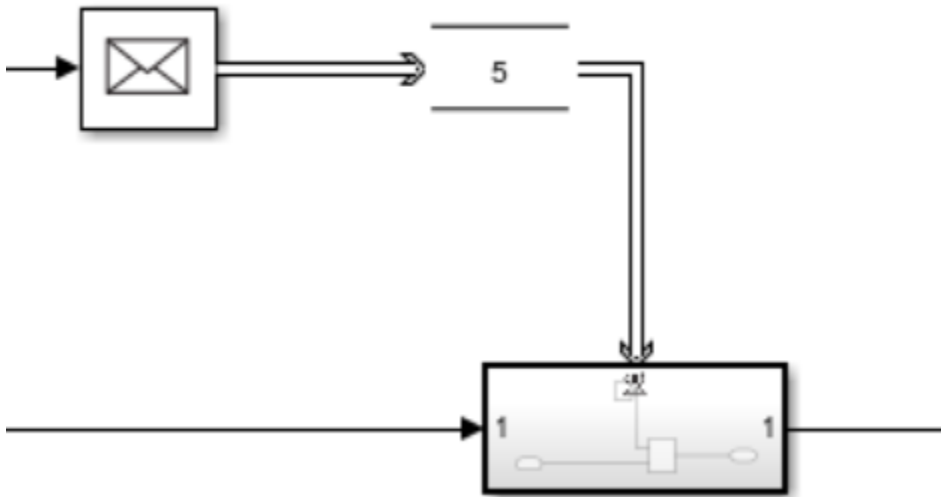


For more information, see [Bus Editor](#).

Message Triggered Subsystem and Message Polling Subsystem blocks: Process messages by executing subsystem when message is available

The new Message Triggered Subsystem and Message Polling Subsystem blocks are each a type of conditionally executed subsystem that uses messages as the control signal. The information contained in the messages is accessible inside the subsystem.

- A Message Triggered Subsystem block executes whenever a message is available at the control port, independent of block sample time.
- A Message Polling Subsystem block pulls messages from a queue periodically based on its sample time and executes only when a message is available.

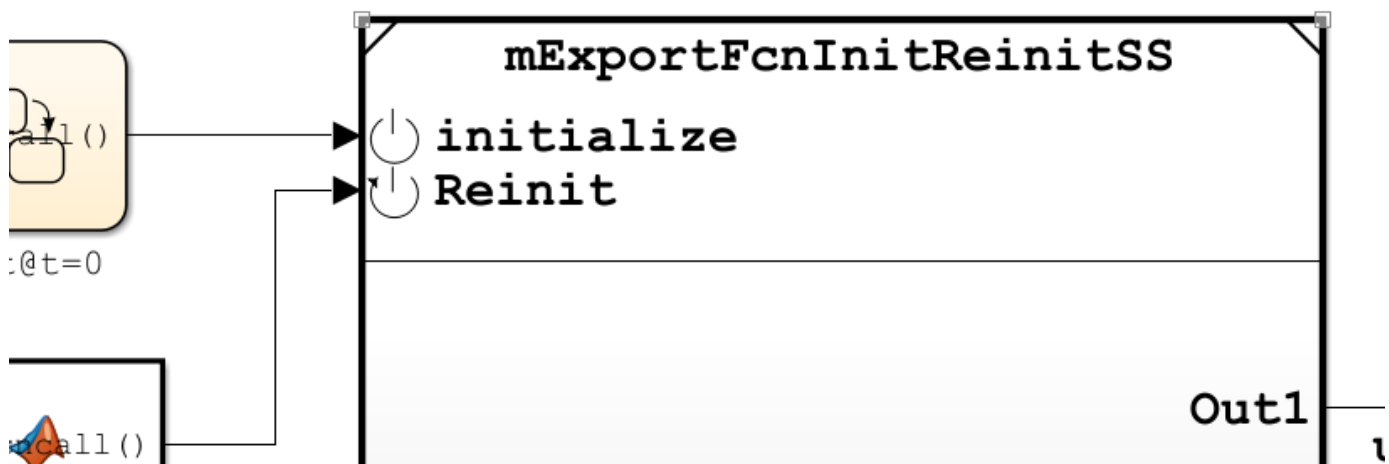


A Message Triggered Subsystem block can be used to define an independent function at the root level of an export-function model. See “Export-Function Models Overview”.


For more information about the Message Triggered Subsystem and Message Polling Subsystem blocks, see Message Triggered Subsystem and “Using Message Polling and Message Triggered Subsystems”.

Reinitialize ports and Reinitialize Function block: Reinitialize referenced model or subsystem states during simulation

Starting in R2022a, you can reinitialize the states of the blocks inside a referenced model or subsystem at any time during model simulation. Sending a function-call signal to a reinitialize port of the Model block or the Subsystem block creates a reinitialize event that implicitly initializes the states of all blocks within the Model or Subsystem block.



To add a reinitialize port to a referenced model or a subsystem, place a Reinitialize Function block inside the referenced model or subsystem. Then, select the **Show model reinitialize ports** check box in the Model block parameters dialog or the **Show subsystem reinitialize ports** check box in

the Subsystem block parameters dialog. The Model or Subsystem block displays a reinitialize port, indicated by the  icon.

By customizing the contents of a Reinitialize Function block, for instance by using State Writer and Parameter Writer blocks, you can configure a reinitialize event to explicitly override specified states in a referenced model or subsystem in addition to implicitly initializing all states in the referenced model or subsystem. See “Reinitialize States of Blocks in Subsystem”.

A reinitialize event can be used to initialize states of blocks inside nested subsystems or referenced models and to initialize certain types of blocks that are not affected by an Enabled Subsystem block reset.

You can place multiple Reinitialize Function blocks inside a referenced model or subsystem. In this case, Simulink creates a reinitialize port for each Reinitialize Function block, with each port name corresponding to the **Event name** of the reinitialize event as specified in the Event Listener block inside the Reinitialize Function block.

Function ports and port-scoped Simulink Function blocks: Call and define functions across peer models

Starting in R2022a, Simulink models can have function ports through which one referenced model calls a function defined in a second referenced model. A model can issue a function call through an invoking function port created by a Function Element Call block to invoke a function defined in another model using a port-scoped Simulink Function block and exported through an exporting function port created by a Function Element block. You can use function ports to model client and server components in a distributed service architecture as well as to facilitate data sharing using a functional interface between component models. See “Call Simulink Functions in Other Models Using Function Ports”.

Code reuse support in code generation

In previous releases, code generated from Simscape models did not support code reuse. This limitation has now been removed.

Use the new match filter with `find_system` to find both active and inactive blocks in a model

You can use the built-in match filter, `Simulink.match.allVariants()`, with the `find_system` function to find all blocks in a variant model regardless of whether the block is active or inactive due to variants.

For more information, see “MatchFilter”.

Find variant control variables used by variant parameters

Starting in R2022a, the import control variables operation in Variant Manager and the `Simulink.VariantManager.findVariantControlVars` method find variant control variables used by variant parameters (`Simulink.VariantVariable`) that are accessible to the model. See, `findVariantControlVars`.

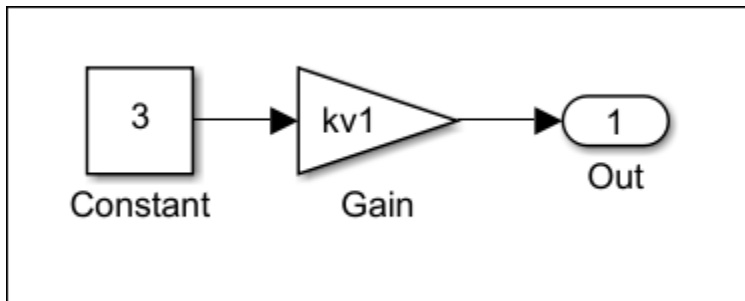
Compile code conditionally for all variant parameter values with different dimensions

You can generate code for active and inactive values of variant parameters that have different dimensions. All other attributes, such as data type and complexity, must be the same across all choices of a variant parameter.

When you generate code for variant parameters with different dimensions, with the activation time of the associated variant control variable object to `code compile`, Embedded Coder® generates code for active and inactive values of variant parameters. The code contains variant parameter definitions and an autogenerated dimension identifier in C preprocessor conditionals `#if` and `#elif`. The C preprocessor conditionals enable you to conditionally compile the code for a given active value of a variant parameter. You do not need to regenerate the code every time you change the value of the variant control variables that are associated with variant parameters. For more information on conditionally compiling code, see “Compile Code Conditionally for all Values of Variant Parameters with Same and Different Dimensions”.

Previously, if a variant parameter had values with different dimensions, you could generate code only for the active values. You had to regenerate the code every time you changed the value of the variant control variable before compiling the code.

Consider this model.



In this model, the **Gain** parameters of the Gain blocks are set to the variant parameter `kv1`.

- 1 In the MATLAB Command Window, define `kv1` as follows. `kv1` has values with different dimensions, and the variant activation time of the associated variant control variable object `V` is set to `code compile`. Set the storage class of `kv1` with exported data scope to generate a code that reuses the variant parameter values that you specified in the model. You can choose to generate a code that imports variant parameter values and its dimensions from external code prior to code compilation by setting the storage class with imported data scope. For more information, see described in “Choose Storage Class for Controlling Data Representation in Generated Code” (Embedded Coder). In this example, the storage class of the variant parameters is set to “ExportedGlobal” (Embedded Coder).

```

% exported parameters
spec = Simulink.Parameter();
spec.CoderInfo.StorageClass = 'ExportedGlobal';

% variant parameters
choice1 = 'V == 1';
choice2 = 'V == 2';
choice3 = 'V == 3';
  
```

```
kv1 = Simulink.VariantVariable('Choices', {choice1, [1, 2, 3, 4], choice2,
    [1, 2, 3, 4, 5], choice3, [1, 2, 3, 4, 5, 6]}, 'Specification', 'spec');
```

```
% variant control variable
```

```
V = Simulink.VariantControl('Value', 1, 'ActivationTime', 'code compile');
```

- 2 Generate the code using Embedded Coder. The code that you generate contains the variant parameter definition section that has active and inactive values of kv1 that you specified in the model. The code also contains the dimension identifier kv1_dim0 to store the dimensions of variant parameter values across the model.

```
//mVPrmVariableDim.c
#if V == 1
real_T kv1[kv1_dim0] = { 1.0, 2.0, 3.0, 4.0 };

#elif V == 2
real_T kv1[kv1_dim0] = { 1.0, 2.0, 3.0, 4.0, 5.0 };

#elif V == 3
real_T kv1[kv1_dim0] = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 };
#endif

//rtw_variant_dims.h
/* Definition for custom storage class: Define */
#if V == 1
#define kv1_dim0          4
#elif V == 2
#define kv1_dim0          5
#elif V == 3
#define kv1_dim0          6
#endif
```

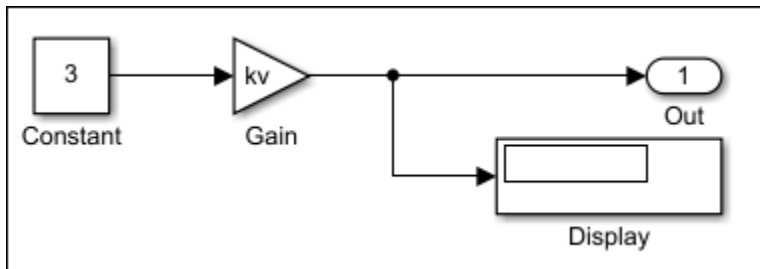
When you compile this code, the compiler evaluates the `#if` and `#elif` conditions to determine the active values and dimension of variant parameters based on the value of the variant control variable `V` that you provide as an input to the compiler. If you specify the value of `V` as 1, the condition `V == 1` evaluates to `true`, and the values enclosed in `V == 1` becomes active during code compilation. You can then specify a different value for `V` and recompile the same code for any other active value of kv1.

Reuse, control representation, and import variant parameter values in generated code using Simulink.Parameter variables

You can specify `Simulink.Parameter` variables as values in variant parameters. Using `Simulink.Parameter` variables allows you to share values among multiple variant parameters, separate values from their data types and other properties, and control the appearance, placement, definition, and declaration of variant parameters values in the generated code. `Simulink.Parameter` variables help you to import variant parameter values from external code at the beginning of code compile, simulation-loop, and model startup phases of simulation and code generation workflows based on the activation time you specify. For more information, see “Stages to Set Active Choices in Variant Blocks and Variant Parameters”. When you use `Simulink.Parameter` variables as values in a variant parameter, corresponding properties of all the `Simulink.Parameter` variables must be the same. Only the values of `Simulink.Parameter` variables can vary.

Previously, you could specify only numeric values in variant parameters.

Consider this model.



In this model, the **Gain** parameter of the Gain block is set to the variant parameter kv.

- 1 In the MATLAB Command Window, define kv as follows. kv has the two values kv_hatchback and kv_sedan of Simulink.Parameter type, and the storage class is set to ImportedExtern.

```
% imported parameters
kv_hatchback = Simulink.Parameter(3);
kv_sedan = Simulink.Parameter(6);
kv_hatchback.CoderInfo.StorageClass = 'ImportedExtern';
kv_sedan.CoderInfo.StorageClass = 'ImportedExtern';
```

```
% variant parameters
kv = Simulink.VariantVariable('Choices', {'V == 1', 'kv_hatchback',
    'V == 2', 'kv_sedan'});
```

- 2 Define a variant control variable object V with the value set to 1 and the activation time set startup.

```
V = Simulink.VariantControl('Value', 1, 'ActivationTime', 'startup');
```

- 3 Generate the code using Embedded Coder or Simulink Coder™. The code is generated as follows.

```
//vPrmImported_private.h
extern real_T kv_hatchback;
extern real_T kv_sedan;

//vPrmImported.c
void vPrmImported_initialize(void) {
if (V == 1) {
    mVPrmImported_P.kv = kv_hatchback;
} else if (V == 2) {
    mVPrmImported_P.kv = kv_sedan;
}
}
```

The initialize function of the generated code is executed at the model startup phase to determine the active value of kv based on the variant condition that evaluates to true. If the condition `V == 1` evaluates to true, kv_hatchback is active. If `V == 2` evaluates to true, kv_sedan is active. The generated code imports active values from external code at model startup.

For more information, see “Reuse Variant Parameter Values from Handwritten Code Using Simulink.Parameter Variables”.

Use enumerated values to improve code readability of variant blocks and variant parameters with startup activation time

You can associate enumerated type of values for the variant control variables with “startup” activation time. Enumerated types improve code readability and simplify the generated code by

giving descriptive names to integers used as variant control values in variant control variable objects. For more information, see “Enumerated Types To Improve Code Readability of Variant Control Variables of Variant Blocks” and “Improve Code Readability of Variant Parameters Using Enumerated Types”.

Only enumerations that defined using these techniques are supported:

- Using the function `Simulink.defineIntEnumType`
- By subclassing one of these built-in classes:
 - Built-in integer data types `int8`, `int16`, `int32`, `uint8`, or `uint16`
 - `Simulink.IntEnumType`

These enumerations are also supported when permanently stored in a Simulink data dictionary. See “Enumerations in Data Dictionary”.

Mask parameter enhancements

Starting in R2022a, you can specify the `Image` and `Value Type` for the `Data Type` parameter. You can also have an embedded browser in the mask dialog by specifying `HTML Text` as `Text Type` for the `Text Area` parameter.

Retrieve list of options and evaluated value of a parameter

Starting R2022a, you can retrieve the list of options for a parameter, Simulink object, block diagram, and annotations using the `get_param` keyword `options`. You can also access the evaluated value of a masked parameter using `get_param` with the keyword `value`.

Use this syntax to get the list of options:

```
get_param(<block_path>, 'options@<parameter_name>');
```

Use this syntax to get the evaluated value is:

```
get_param(<block_path>, 'value@<parameter_name>');
```

Here, `block_path` is the block, `parameter_name` is the name of the parameter on the block whose list of options or evaluated value is to be retrieved.

For more information, see `get_param`

Constraint Manager integrated with Mask Editor

The Constraint Manager is now integrated inside the Mask Editor. The workflow for sharing constraints is simplified, and you can specify constraints for the input and output ports of blocks.

- In the current shared constraint workflow, the Constraint Manager is opened in a separate window, and you must create a constraint, save it in a MAT file, and associate the file name with the parameter. Now, you can create or load a MAT file, define constraints or modify existing ones, and associate constraints with parameters within the Mask Editor. The **Constraint Browser** allows you to manage shared constraints.

MASK EDITOR **CONSTRAINTS**

Save Mask Load New Parameter Constraint Cross Parameter ... Port Constraint Port Identifiers Preview Dialog Help Tutorial

ACTION MAT FILE CONSTRAINTS PORT IDENTIFIERS PREVIEW RESOURCES

Constraint Browser

Name	Type
Mask Constraints	
Gain_input	Port
Delta	Parameter

Use this pane to create parameter constraints and associate them with edit parameters. If multiple attributes are selected from a rule category, the parameter value must satisfy at least one attribute from each category.

Constraint Name:

Rule

Data Type:

Complexity: real complex

Dimension: scalar row vector column vector 2-D matrix n-D matrix

Sign: positive negative zero

Finiteness: finite inf -inf nan

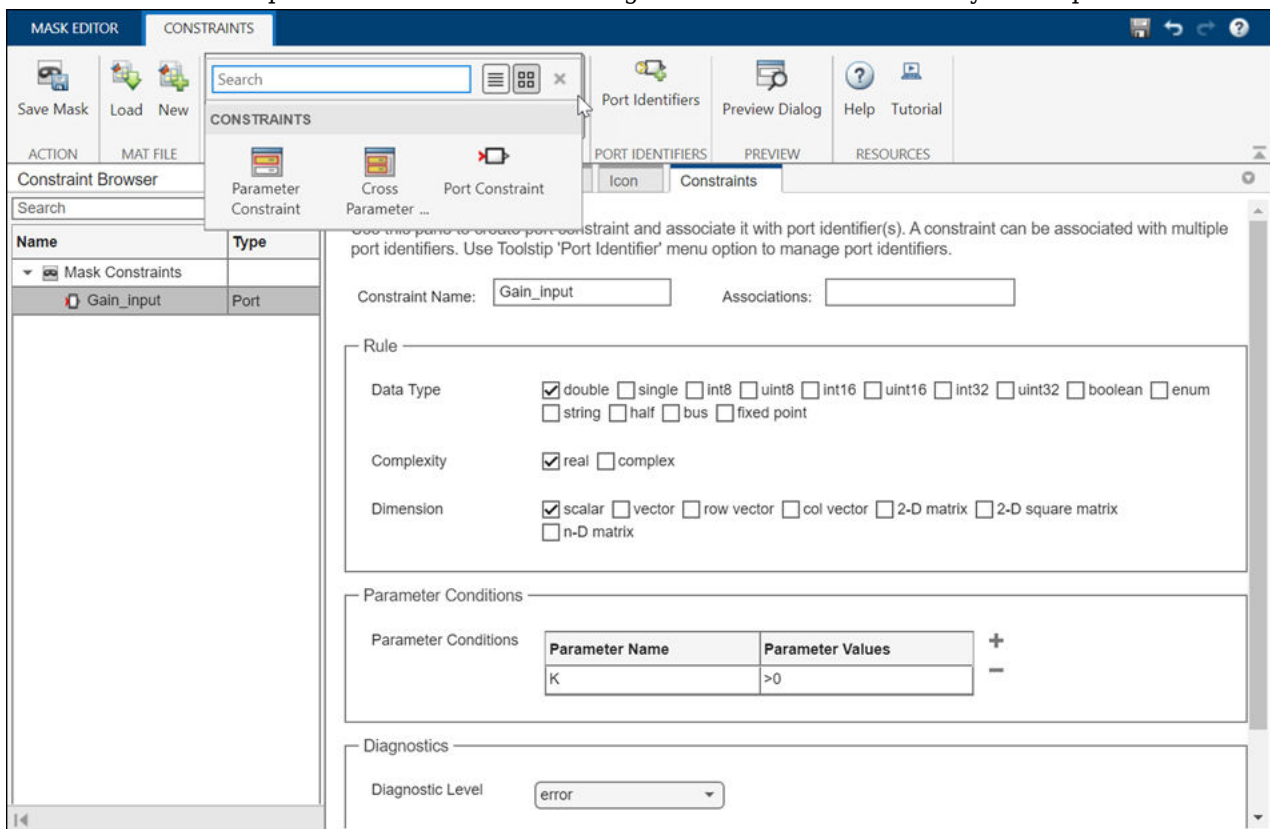
Range:

Custom Rule

MATLAB Expression:

Error Message:

- You can use port constraints to specify constraints on the input and output ports of a masked block. The port attributes are checked against the constraints when you compile the model.



Control Subsystem Reference programmatically

Use the APIs in the `Simulink.SubsystemReference` package in a scripted workflow to query and control the subsystem reference block diagram and blocks programmatically.

`convertAllSubsystemReferenceBlockToSubsystem`

`convertSubsystemReferenceBlockToSubsystem`

`convertSubsystemToSubsystemReference`

`getActiveInstances`

`getAllDirtyInstances`

`getAllInstances`

`getAllReferencedSubsystemBlockDiagrams`

`getNearestParentSubsystemReferenceBlock`

`getSystemOwningTheLock`

`isSystemLocked`

Variable-size signals in array of buses


Starting in R2022a, signals of a bus type containing variable-size signals can form an array of buses, and an array of buses can be a variable-size signal. See “Variable-Size Signal Basics” and “Group Nonvirtual Buses in Arrays of Buses”. Support for such signals is limited to normal mode simulation, and not all blocks are supported.

Designation of export-function model

Starting in R2022a, Simulink allows you to explicitly designate a model as an export-function model. See “Export-Function Models Overview”. Previously, Simulink attempted to determine whether or not a model was an export-function model by analyzing the contents of the model.

After you designate a model as an export-function model, Simulink performs compile-time checks to warn you if the model does not meet export-function model requirements.



Simulink displays a  badge in the lower-left corner of the canvas to indicate that a model is designated as an export-function model.

If you load a model that was created and saved in a prior release, Simulink designates the model as an export-function model if the model meets the requirements to be an export-function model. In some cases, you may need to manually designate such a model as an export-function model.

For more information, see “Designating an Export-Function Model”.

Redesigned Simulation Target Pane of Model Configuration Parameters

The **Simulation Target** pane of the Model Configuration Parameters dialog is changed for R2022a. The parameters that pertain to custom code are now together in one section of the dialog, which is organized into three tabs: **Code information**, **Additional source code**, and **Import settings**. Clicking on a tab shows the parameters that are listed under that tab and hides the contents of the other two tabs.

Some of the parameters have new names, and two parameters, **Compiler flags** and **Linker flags**, both found on the **Code information** tab, are new for R2022a.

In the **Compiler flags** field, you can enter any additional compiler flags to be added to the compiler command line when your custom code is compiled.

In the **Linker flags** field, you can enter any additional linker flags to be added to the linker command line when your custom code is linked.

GPU acceleration

Language: C

Custom Code

Code information Additional source code Import settings

Include headers Include directories Source files Libraries Defines Compiler flags Linker flags

Include headers:

Auto-fill from source files

Validate custom code

GPU acceleration

Language: C

Custom Code

Code information Additional source code Import settings

Initialize code Terminate code Additional code

Initialize code:

Validate custom code

GPU acceleration

Language: C

Custom Code

Code information Additional source code Import settings

Simulate custom code in a separate process

Enable custom code analysis

Enable global variables as function interfaces

Undefined function handling: Use Interface only

Deterministic functions: None Specify by function

Default function array layout: Not specified Exception by function...

Validate custom code

The following table lists the parameters in the new Simulation Target pane dialog that have different names or are in different locations as a result of this change.

Parameter name prior to R2022a	New name in R2022a	New location in R2022a
Insert custom C code in generated > Header file	Include headers	Code information tab
Insert custom C code in generated > Source file	Additional code	Additional source code tab
Insert custom C code in generated > Initialize function	Initialize code	Additional source code tab
Insert custom C code in generated > Terminate function	Terminate code	Additional source code
Additional build information > Source files	Source files	Code information tab
Additional build information > Include directories	Include directories	Code information tab
Additional build information > Libraries	Libraries	Code information tab
Additional build information > Defines	Defines	Code information tab
Import custom code		Advanced parameters
Simulate custom code in a separate process		Import settings tab
Enable custom code analysis		Import settings tab
Enable global variables as function interfaces		Import settings tab
Undefined function handling		Import settings tab
Deterministic functions and Specify by function		Import settings tab
Default function array layout and Exception by function		Import settings tab
Reserved names		Advanced parameters

Interactively update legacy code in a System object

Starting in R2022a, you can interactively update legacy code in a System object™.

In the MATLAB Toolstrip, the **Inspect** button replaces the **Analyze** button for System objects. The Analyzer that opens when you click the **Inspect** button now provides warnings for legacy base classes, properties with legacy attributes, and redundant methods. When a System object contains legacy code, the Analyzer displays an **Update** button that replaces or removes the legacy code.

For more information, see “Analyze System Object Code”.

Half-precision data type support for System objects

Starting in R2022a, MATLAB System objects support the half-precision data type for input, output, and System object property values. The half-precision data type is not currently supported for simulation or code generation in the MATLAB System block.

MATLAB class object is now a non-tunable Parameter

Starting in R2022a, MATLAB class objects are a non-tunable parameter in the MATLAB System block. This property can be set in the block dialog and is no longer read-only.

String support for messages in MATLAB System block

In R2022a, the MATLAB System block supports messages whose payloads contain strings. Message payloads containing strings are only supported in interpreted mode.

Enhancements to co-simulation numerical compensation user interface

In R2022a, the numerical compensation user interface has these enhancements:

- Right-click access in the menu of the co-simulated component under **Configure Cosimulation Signal Compensation**
- Settings for **Auto**, **Always**, or **Off**
- New user interface to see options and conditions for each port



Numerical Compensation User Interface

For more information, see “Numerical Compensation”.

In previous releases, these settings were in separate interfaces and you could only programmatically set the numerical compensation.

Multi-instance Simulink Function blocks

In R2021b, Simulink generated an error when you simulated a Stateflow Chart or MATLAB Function block that called a multi-instance Simulink Function block.

In R2022a, you can simulate a Stateflow Chart or MATLAB Function block that calls a multi-instance Simulink Function block. To specify a multi-instance Simulink Function block, configure the referenced model that includes the Simulink Function block. In the referenced model, set the model configuration parameter **Total number of instances allowed per top model** to `Multiple`.

New example of parsing NMEA GPS Text Messages Using Simulink and Python.

A new example shows how to parse text messages in NMEA GPS format using Simulink and Python®. For more information, see “Integrate Python GPS Text Message Parsing Algorithms in Simulink”.

New example of modeling a distributed wireless tire pressure monitoring system

A new example shows how to design, model, and simulate a distributed monitoring system. In the example, multiple sensors communicate wirelessly with a controller. The controller takes action when it detects a fault condition and maintains a log of fault conditions. The example also shows how to prepare this model to generate code that is deployable on a target hardware platform where the model algorithm is implemented as a component as part of a system.

For more information, see “Wireless Tire Pressure Monitoring System with Fault Logging” and “Prepare Sensor and Controller Models in a Distributed Monitoring System for Code Generation”.

Functionality being removed or changed

SortedOrder parameter no longer supported

Behavior change

Starting in R2019b, the `SortedOrder` parameter is no longer supported, and its value is ignored.

Bus Editor no longer supports custom import and export functions

Errors

The **Bus Editor** no longer supports custom import and export functions.

Remove the `cm.BusEditorCustomizer.importCallbackFcn` and `cm.BusEditorCustomizer.exportCallbackFcn` registration functions from the `sl_customization` function. To eliminate both customizations in one operation for a customization manager object named `cm`, enter:

```
cm.BusEditorCustomizer.clear
```

For more information about customizations, see “Register Customizations with Simulink”.

Project and File Management

Projects: Reduce test runtime in continuous integration workflows using the dependency cache

You can now specify where your project stores the dependency analysis results. In agile development workflows that use Git™ and a continuous integration (CI) server, share the dependency cache file (.graphml) to run an incremental dependency analysis and reduce the test suite runtime. See “Continuous Integration Using MATLAB Projects and Jenkins”.

To set the project dependency cache file, on the **Project** tab, in the **Environment** section, click **Details**. In **Dependency cache file**, browse to and specify a GraphML file. If the cache file does not exist, the project creates it for you.

Alternatively, you can create and set the project dependency cache programmatically:

```
matlab.project.example.timesTable
proj = currentProject;
proj.DependencyCacheFile = "myProjectCacheFile"
```

Project API: Export top-level and referenced project to previous version using Simulink.exportToVersion

You can now use `Simulink.exportToVersion` to export the top-level project with all references to open in a previous version of Simulink.

Dependency Analyzer: Save dependency graph as image

You can now save the dependency analysis results as an image. See “Export Dependency Analysis Results”.

Projects Examples: Determine optimal order for resolving conflicts

When collaborating on a project, a branch merge can lead to several conflicted files. Use the Dependency Analyzer to programmatically determine the optimal order in which to resolve conflicts. See “Determine Order for Resolving Conflicts Using Dependency Analyzer”.

Text Comparison: Save results as HTML report

You can now use the Comparison Tool to publish text comparison results in an HTML report. For more details, see “Compare Text Files”.

Folder Comparison: Compare folders in MATLAB Online

Starting in R2022a, you can compare folders and zip files in MATLAB Online.

You can access the comparison tool from:

- The MATLAB Current Folder browser context menu

- The Current Project browser context menu
- The MATLAB Command Window using the `visdiff` function

Functionality being removed or changed

Simulink will no longer support getting parameters of the default block diagram

Warns

Getting parameters of the default block diagram will not be supported in a future release. In R2022a, scripts that use `get_param(0,parameter)` continue to work but throw a warning.

Data Management

Import Simulink Design Verifier `sldvData` data into Signal Editor user interface

To edit data in MAT-files that contain Simulink Design Verifier™ `sldvData` structures, use the `Simulink.io.SLDVMatFile` class to create a custom reader that imports `sldvData` data into the Signal Editor user interface.

Signal Editor user interface tabular improvements

The Signal Editor user interface has these tabular improvements:

- You can now navigate through the Signal Editor user interface tabular area using **Tab** and **Shift+Tab**. For more information, see “Work with Basic Signal Data with a Tabular Editor”.
- To access a context menu of tabular area edits, right-click in the tabular area. You can:
 - Edit a row.
 - Insert or delete a row.
 - Replace the contents of a row with a MATLAB expression.

Root Import Mapper updates

- When you are ready to map your data to input ports, the Root Import Mapper now has a context menu that contains the same mapping options as the **Map to Model** button.
- The **Map to Model** options are now enabled by context. In previous releases, all options were enabled and an error message was returned if you selected an unavailable option.

Relax data consistency checking across model hierarchies

Previously, Simulink enforced data consistency across data sources in a model reference hierarchy. Duplicate data definitions could only exist in a hierarchy under these conditions:

- Each model in the hierarchy can see only one definition.
- Definitions must be the same across models in the hierarchy.

In R2022a, the model parameter `EnforceDataConsistency` provides the option to relax the second condition, which allows you to more easily integrate Simulink models for large-scale simulations. When you use the `set_param` function to set `EnforceDataConsistency` to `off`, the current model and the models below it in the model hierarchy can use symbols with the same name but different values as long as each model in the hierarchy can see only one definition of the symbol.

```
set_param(bdroot, 'EnforceDataConsistency', 'off');
```

By default, Simulink enforces data consistency, so the parameter is set to `on`. You cannot set `EnforceDataConsistency` to `on` for a model unless each of its referenced models and their variants also set `EnforceDataConsistency` to `on`.

Disable consistency checking only when simulating a model in Normal mode. Setting `EnforceDataConsistency` to `off` results in error for:

- Code generation
- SIL and PIL simulations
- Simulation of a model hierarchy that contains one or more models running in Accelerator or Rapid Accelerator mode
- Simulation of a protected model

For more information, see “Considerations before Migrating to Data Dictionary”.

Model argument enhancements for Simulink.LookupTable and Simulink.Breakpoint objects

In R2022a, these changes enhance the usability of Simulink.LookupTable objects configured as model arguments.

- When you configure a Simulink.LookupTable object as a model argument, you can now also configure a Simulink.Breakpoint object that is referenced by that Simulink.LookupTable as a model argument. Configure a Simulink.Breakpoint object as a model argument by selecting the **Argument** check box in the Model Explorer, Simulink.Breakpoint property dialog box, Model Data Editor, or Property Inspector.

A Simulink.Breakpoint object that is configured as a model argument can be used in a Simulink.LookupTable object when the **Breakpoints specification** property is set to Reference. When using a Prelookup block with an Interpolation Using Prelookup block in your model, this combination of a Simulink.LookupTable object with a Simulink.Breakpoint object allows you to share a breakpoint object between multiple Prelookup blocks and provide breakpoint values for each model instance.

When you configure a Simulink.Breakpoint object as a model argument, a parent model can provide the instance-specific value for the model argument by:

- Using a Simulink.Breakpoint object.
- Providing an array of numeric values that define the breakpoints. The array must match the design definition for the breakpoint object. Simulink synthesizes a Simulink.Breakpoint object by using the values in the array.
- When a parent model provides a Simulink.LookupTable object as an instance-specific value for a model argument, Simulink passes only the **Table** property of the lookup table object to the child model. This action applies to Simulink.LookupTable objects with **Specification** set to Reference. This behavior change allows you to configure model arguments for Simulink.LookupTable objects and Simulink.Breakpoint objects independently of each other and reuse these objects as instance parameters for multiple Model blocks.

For more information, see “Parameterize Instances of a Reusable Referenced Model”.

Block Enhancements

C Function block can interface with C++ classes

Starting in R2022a, code that you specify in the C Function block can interface directly with C++ classes defined in your custom code. You can:

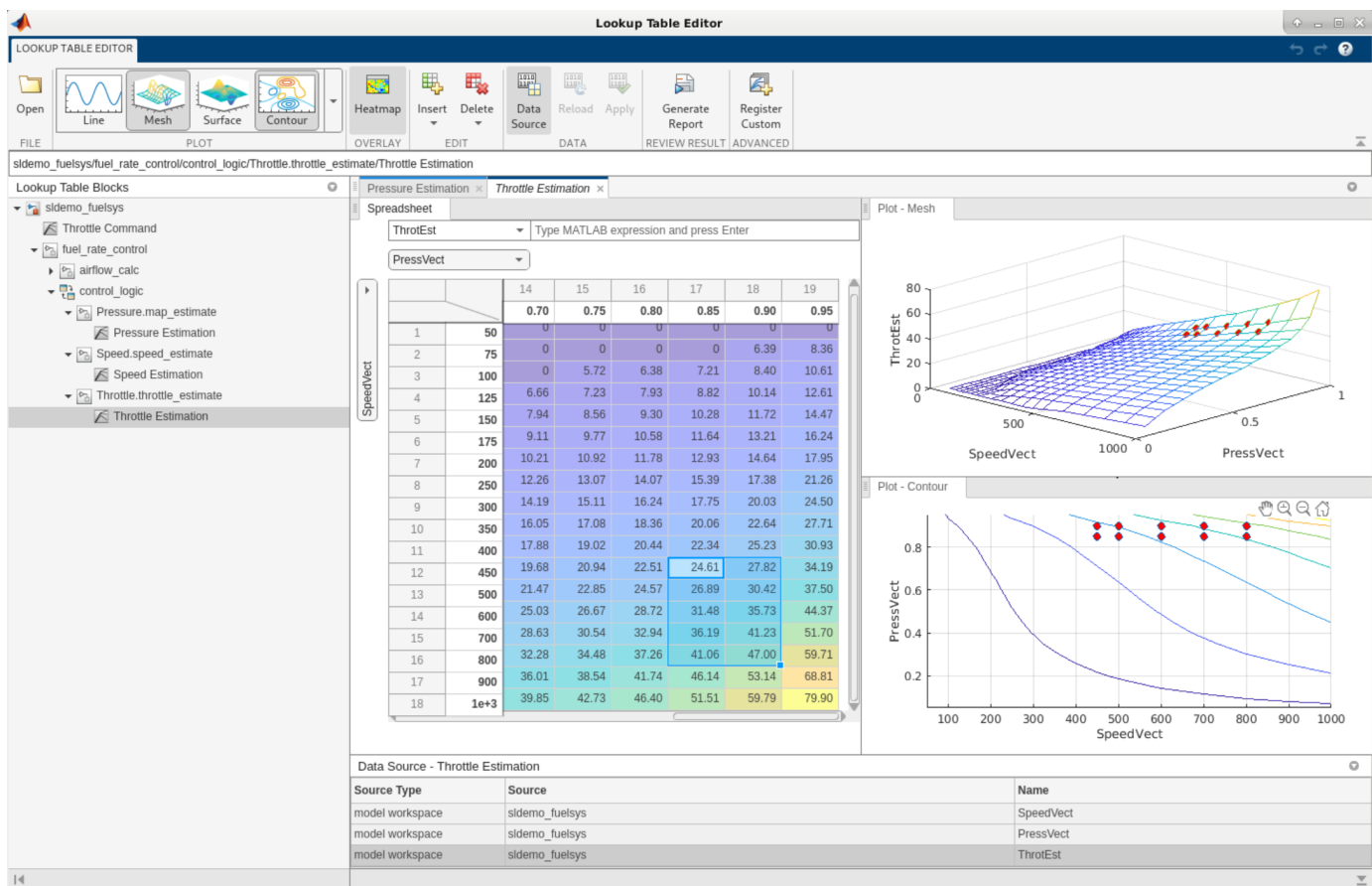
- Instantiate an object of a C++ class defined in your custom code
- Read and write to public data members of the class object
- Call public class methods of the object

Previously, it was necessary to write and call C wrapper functions to access C++ classes from the C Function block.

For more information, see “Interface with C++ Classes Using C Function Block”.

Enhanced Lookup Table Editor

The Lookup Table Editor has a new look and feel.



This graphic is a view of the Lookup Table Editor for the Throttle Estimation lookup table block in the `sldemo_fuelsys` model.

Highlights of the new interface include:

- Multiple ways to access the Lookup Table Editor, including from the Simulink Editor toolstrip, MATLAB command line, and lookup table block dialogs.
- From within the Lookup Table Editor, the ability to view lookup tables from models loaded in the current MATLAB session.
- Visualization and editing of lookup table data in a consistent way similar to Microsoft Excel®, regardless of the data source.
- A navigation field that lets you enter the path to models and subsystems that contain lookup table blocks, and lookup table blocks themselves.
- Report generation containing lookup table data and surface plots.
 - For 1-D data, the report generates line plots.
 - For more than 1-D data, the report generates surface plots, one plot for each 2-D slice of data.
- In the tabular area, the Lookup Table Editor contains the same lookup table spreadsheet as is available through the Simulink.LookupTable property dialog box (“Edit Lookup Table Data with Lookup Table Spreadsheet”).
- Easy registration of custom lookup table blocks.
- Generation of different plots for the data, including line, plot, surface, and contour plots.
- Heatmap views for data.

For more information, see “Use the Lookup Table Editor”.

Lookup Table blocks updated for better sharing and reuse

These blocks have been updated to better share and reuse utility functions in generated code:

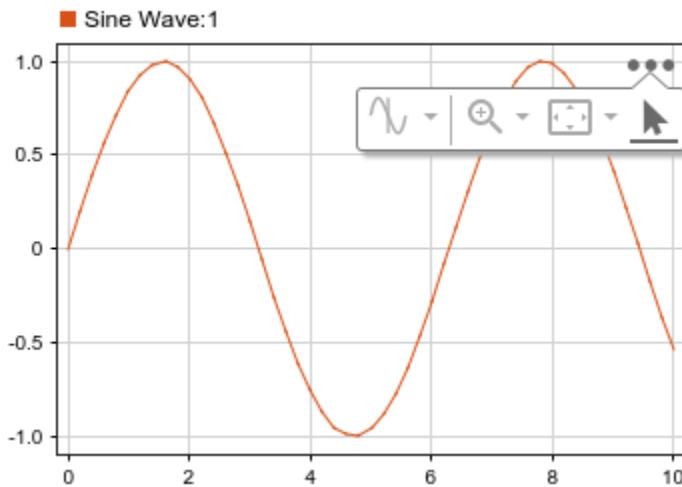
- n-D Lookup Table
- 1-D Lookup Table
- 2-D Lookup Table
- Interpolation Using Prelookup

Configure Dashboard Scope plots more easily using subplot menu

You can now use the new subplot menu for the Dashboard Scope to:

- Show or hide data cursors.
- Zoom out by a fixed amount.
- Configure the mouse interaction to pan and select or to zoom in time, in y , or in both time and y .
- Perform a fit-to-view in time, in y , or in both time and y .

To open the subplot menu on the Dashboard Scope block, select the block then click the three dots that appear when you pause on the plot area.



In previous releases, you accessed these options using the context menu for the Dashboard Scope block.

New Matrix Operations blocks

The Matrix Operations library has these new blocks:

- IsHermitian — Check if matrix is Hermitian or non-Hermitian
- Matrix Concatenate — Concatenate input signals of same data type to create contiguous output signal
- Matrix Multiply — Multiply and divide scalars and nonscalars or multiply and invert matrices
- Permute Matrix — Reorder matrix rows or columns
- Submatrix — Select subset of elements (submatrix) from matrix input
- Create Diagonal Matrix — Create square diagonal matrix from diagonal elements
- Extract Diagonal — Extract main diagonal of input matrix

New Logic and Bit Operations library blocks

The Bit to Integer Converter and Integer to Bit Converter blocks have been added to the **Simulink > Logic and Bit Operations** library and remain available in the **Communications Toolbox > Utility Blocks** library. All existing models continue to work.

Trigonometric Function block updates

The Trigonometric Function block now lets you:

- Change the lookup table algorithm table data types. When you select **Lookup** for the **Approximation method** parameter, use the **Data Types** tab. In previous releases, the table data type was derived from the input data type.
- Control how the lookup table algorithm interpolates the table value using neighboring breakpoints. Use the new **Interpolation method** parameter and select **Linear point-slope** or **Flat**. In previous releases, **Linear point-slope** was the only interpolation method.

New Message Triggered Subsystem and Message Polling Subsystem blocks

The Message Triggered Subsystem and Message Polling Subsystem blocks are added to the Messages & Events library for R2022a. Use these blocks to create a subsystem whose execution is controlled by message input. See Message Triggered Subsystem and “Using Message Polling and Message Triggered Subsystems”.

New Reinitialize Function block and reinitialize ports for Model and Subsystem blocks

The Reinitialize Function block is added to the User-Defined Functions library for R2022a. Use this block to create a subsystem that executes upon a model or subsystem reinitialize event. The block contains an Event Listener block with the **Event type** set to Reinitialize.

The Model block has a new parameter, **Show model reinitialize ports**, and the Subsystem block has a new parameter, **Show subsystem reinitialize ports**. In both cases, when the parameter is selected, the block displays a port or ports that can be used to generate a model or subsystem reinitialize event.

Port-scoped Simulink Function blocks

A new port-scoped visibility option for the Simulink Function block is added for R2022a. To create a port-scoped Simulink Function block, set the **Function visibility** parameter of the Trigger block inside the Simulink Function block to **port** and enter the port name in the **Scope to port** field. Use port-scoped Simulink Function blocks to implement functions called through exporting function ports created by Function Element blocks. See “Call Simulink Functions in Other Models Using Function Ports”.

New Function Element and Function Element Call blocks

The Function Element and Function Element Call blocks are added to the Ports & Subsystems library for R2022a. Use these blocks to create exporting function ports and invoking function ports, respectively, in referenced models. See “Call Simulink Functions in Other Models Using Function Ports”.

New capabilities for Parameter Writer block

Starting in R2022a, the Parameter Writer block can write to the following:

- Parameters of other blocks that are tunable during simulation
- Instance parameters belonging to a Model block referencing a model
- Masked subsystem parameters
- Model workspace variables

Previously, the Parameter Writer block could write only to instance parameters belonging to a Modelblock referencing a model.

Additionally, starting in R2022a, the Parameter Writer block can be used inside all conditionally executed subsystems, as well as inside Initialize Function, Reinitialize Function, and Reset Function blocks. Previously, the block could be used only inside Initialize Function and Reset Function blocks.

For more information, see “Initialize and Reset Parameter Values”.

C Function block can access custom code global ND arrays

Starting in R2022a, the C Function block can access global variables defined in your custom code that are multidimensional arrays or `struct` types with multidimensional array fields.

C Caller and C Function blocks support 64-bit integer types as `int64` and `uint64`

Starting in R2022a, you can import custom C code with 64-bit integer types and use such code with C Caller and C Function blocks, which support these types as `int64` and `uint64` types in Simulink. Previously, the C Function block did not support 64-bit integer types, and the C Caller block supported such types as `fixdt(1,64,0)` in Simulink.

C Function block can specify code to initialize conditions when subsystem states reset

Starting in R2022a, you can specify code in a C Function block to execute when enabling a subsystem or model in which the block is placed. Specify this code in the **Initialize Conditions Code** box in the block dialog. The code executes one time at the start of simulation, and if the block is inside a subsystem or model containing an Enable block with the **States when enabling** parameter set to **reset**, the code also executes each time the subsystem or model switches from disabled to enabled. You can use this code to set an initial output value or reset the value of a persistent variable.

Selector and Assignment blocks support out-of-range index checking in accelerator and rapid accelerator modes

Starting in R2022a, the Selector and Assignment blocks can perform run-time checking of index values to ensure that they are not out of range when simulating a model in accelerator or rapid accelerator mode. Previously, these blocks performed such run-time checking only when simulating in normal mode, and an out-of-range index in accelerator or rapid accelerator mode could produce unpredictable results or cause MATLAB to crash. See “Check for out-of-range index in accelerated simulation” (Assignment) and “Check for out-of-range index in accelerated simulation” (Selector).

For Each Subsystem block accepts variable-size signals

Starting in R2022a, you can use the For Each Subsystem block with one-dimensional variable-size signals, including variable-size arrays of buses. See “Limitations of For-Each Subsystems”.

Specify origin for value bar and needle on customizable dashboard blocks

You can specify an origin on the scale for several blocks in the Customizable Blocks library:

- Circular Gauge
- Vertical Gauge
- Horizontal Gauge
- Knob
- Vertical Slider
- Horizontal Slider


The origin specifies the value on the scale from which the needle moves and the value bar grows. For example, the origin for this gauge with a scale range from -100 to 100 is set to 0, such that the value bar indicates the value of the connected signal relative to zero.

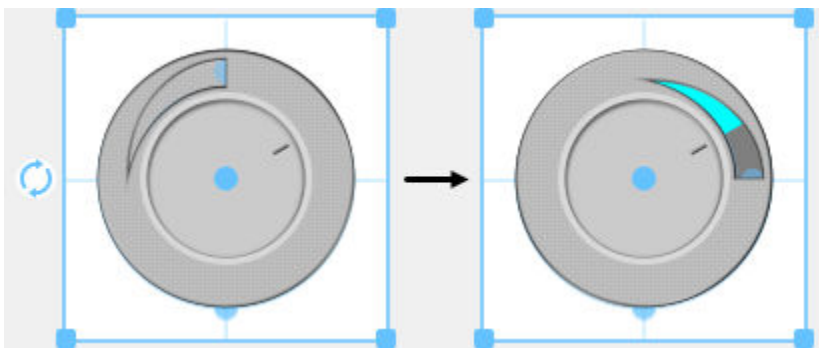


By default, the origin is specified as auto, and the value for the origin is the minimum value for the scale.

Resize and reposition foreground image for customizable dashboard blocks

The blocks in the Customizable Blocks library have an option to add a foreground image as part of the design for a block. Starting in R2022a, you can rotate, resize, and reposition the foreground image within the block design.

For example, if the foreground image has the wrong orientation when you add it to the block, click the rotation button  to rotate the image to the correct orientation in 90 degree increments.



Change scale direction for customizable dashboard blocks

Several dashboard blocks in the Customizable Blocks library allow you to specify the direction for the scale:

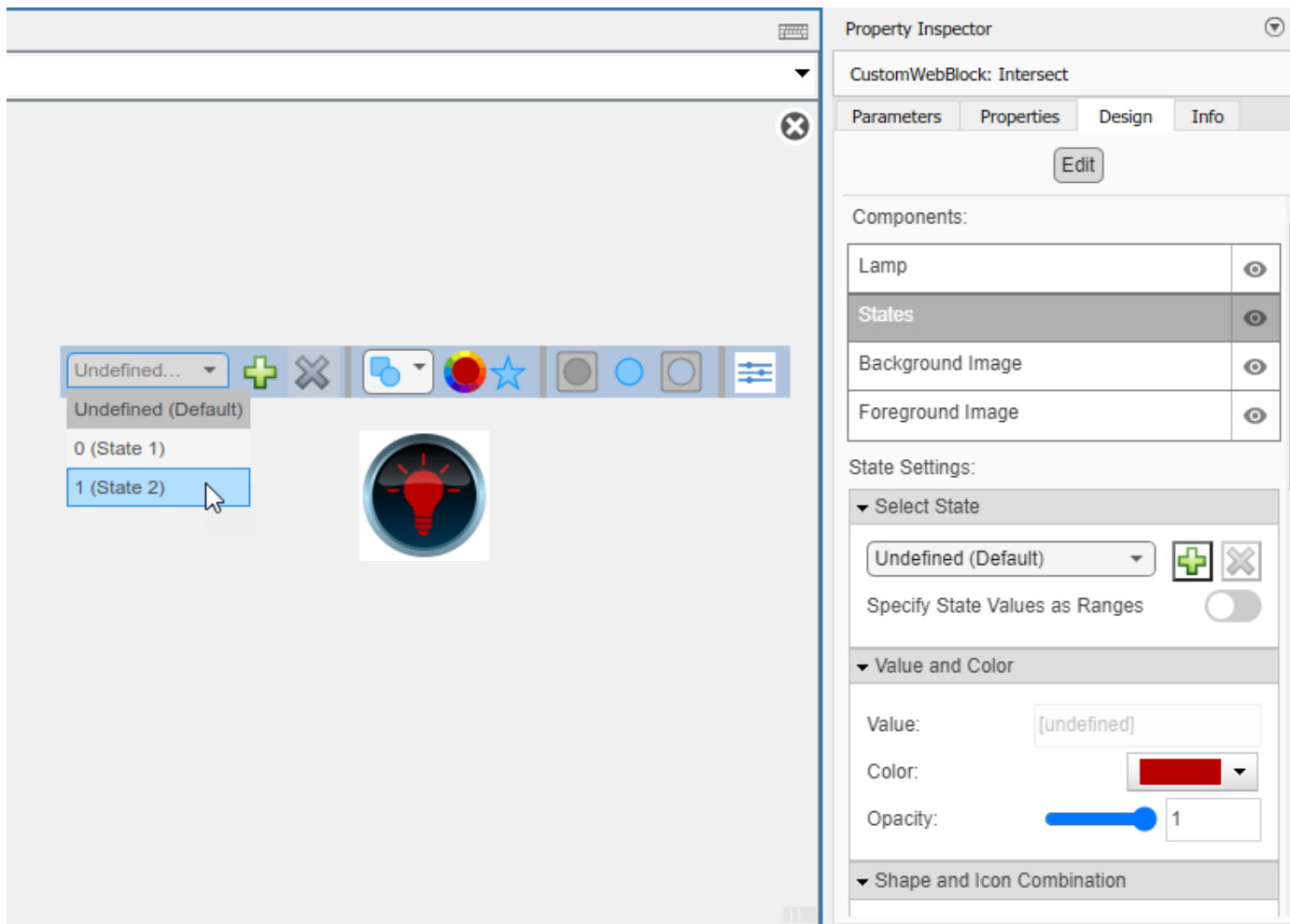
- Circular Gauge
- Vertical Gauge
- Horizontal Gauge
- Knob
- Vertical Slider
- Horizontal Slider

Lamp block in Customizable Blocks library supports specifying state values as ranges

The Lamp block connects to a signal and displays a color that reflects the value of the connected signal. In previous releases, you could only specify a discrete value for each state and design a Lamp block that indicated when the connected signal had that specific value. Starting in R2022a, you can specify each state value as a range and design a Lamp block that indicates when value of the connected signal is within each range.

View and edit each state for customizable Lamp block in design mode

Starting in R2022a, you can view and edit each state for the customizable Lamp block in design mode, and you can modify more aspects of the block appearance for each state. For example, you can now design a lamp that uses a different icon for each state.



Change opacity of state colors for customizable Lamp block

For the customizable Lamp block, you can now change the opacity of the color that you set for a state.

From Spreadsheet block Range Selection tool updates

Range Selection from the **Range** parameter of the From Spreadsheet block has these changes:

- The interface has an updated toolbar in the top right. There is no longer a **Help** icon for this interface. Instead, see the “Range” parameter documentation.
- The **Range** field is now **Selection/History**.
- If you do not specify a range in the **Range** parameter, **Range Selection** now displays all cell entries as selected.

Unit Conversion block now accepts multidimensional signals

The Unit Conversion block now accepts multidimensional signals at the block input port.

Simulink.ImageType data type for image signals

In R2022a, you can now use the Computer Vision Toolbox™ to specify an image of the `Simulink.ImageType` data type and simulate the model. Specify this data type for signals and other data in the model. The `Simulink.ImageType` data type is an encapsulated object that defines an image by using fixed meta-attributes specific to this data type.

The blocks listed in this table support simulation and code generation of a `Simulink.ImageType` object.

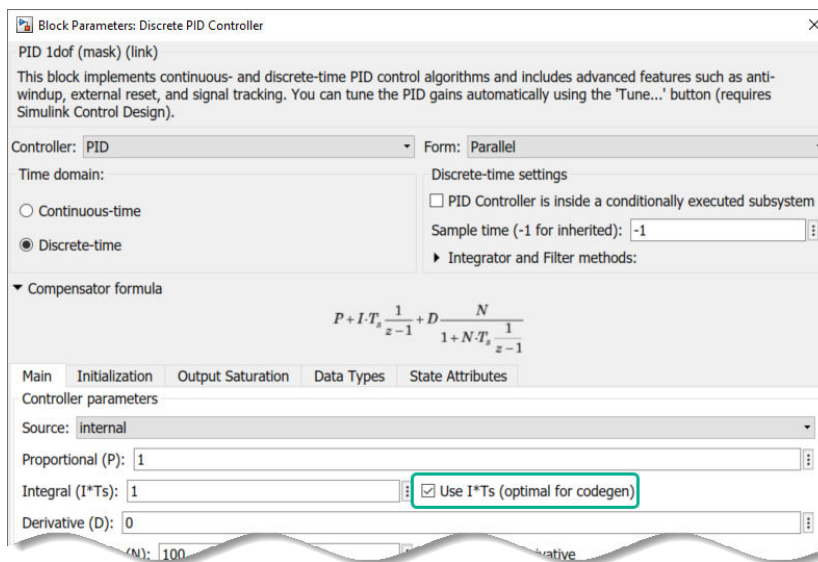
Block Library	Block Name
Sources	<ul style="list-style-type: none"> • Ground • Inport • Outport
Signal Routing	<ul style="list-style-type: none"> • Goto • From • Data Store Read • Data Store Write • Data Store Memory • Switch • Multiport Switch • Merge • Variant Source • Variant Merge (internal block added during code generation) • Mux • Demux • Vector Concatenate, Matrix Concatenate • Selector
Sink	<ul style="list-style-type: none"> • Terminator • To Workspace (simulation only) • Record (simulation only)
Ports & Subsystems	<ul style="list-style-type: none"> • Subsystem, Atomic Subsystem, CodeReuse Subsystem • Enabled Subsystem • Triggered Subsystem • Function-Call Subsystem • If • Switch Case • Resettable Subsystem • For Iterator Subsystem • Model

Block Library	Block Name
Discrete	<ul style="list-style-type: none"> Unit Delay
Signal Attributes	<ul style="list-style-type: none"> Signal Conversion Signal Specification
User-Defined Functions	<ul style="list-style-type: none"> Initialize Function, Reset Function, and Terminate Function Simulink Function block with side I/O and arguments

For more information, see the “Computer Vision with Simulink: Specify image data type in Simulink model” (Computer Vision Toolbox) release note.

Discrete PID Controller Blocks: Improve execution time of generated code

By default, the Discrete PID Controller and Discrete PID Controller (2DOF) blocks take the integral gain as input and multiply it by the sample time internally as a part of performing the integration. You can now configure the blocks to accept the integral gain premultiplied with the sample time as an input. Doing so improves the execution time of the generated code. Therefore, when you have limited processing power, enable the **Use I*Ts** parameter and specify the **Integral** parameter value as the integral gain multiplied by the controller sample time.



Additionally, if your controller uses an anti-windup mechanism, enhancements to the clamping method also reduce the execution time of the generated code.

Functionality being removed or changed

To Workspace block logging enhancements

Behavior change

Starting in R2022a, the To Workspace block supports:

- Logging an array of buses.
- Logging signals with `string` and `half` data types.
- Logging `int64` and `uint64` data using built-in data types.
- Using the `Timeseries` format in rapid accelerator simulations.
- Using the `Timeseries` format for To Workspace blocks inside For-Each subsystems.

Data logged using To Workspace blocks also streams to the Simulation Data Inspector during simulation.

Version History

- In prior releases, the To Workspace block logged `int64` and `uint64` as a `fi` object when a license for Fixed-Point Designer™ was available and as `double` data when the license was not available. Starting in R2022a, the To Workspace block always logs `int64` and `uint64` data using the built-in data types.
- In prior releases, data logged using the To Workspace block only logged to the workspace and was overwritten for each simulation unless you changed the logging variable names or saved the results yourself.

Starting in R2022a, data logged using To Workspace blocks logs to the Simulation Data Inspector as well as the workspace. When you run multiple simulations in a single MATLAB session, the Simulation Data Inspector automatically retains results from each simulation so you can analyze the results together. Starting in R2022a, because data logged using To Workspace blocks streams to the Simulation Data Inspector, this data is also retained.

Logging large amounts of data or running many simulations can produce large amounts of data that fill up disk space. To learn how to control the amount of data retained in the Simulation Data Inspector, see “Limit the Size of Logged Data”.

Access to Simulink Scope through Handle Graphics API will be removed



Still runs

The underlying graphics of the Simulink Scope block will change in a future release. Once that happens, you will not be able to access or modify the Simulink Scope block through the handle graphics API.

Update any undocumented use of handle graphics APIs that access the Scope window.

Floating Signal Selection will be removed

Still runs

The floating signal selection feature () in the Floating Scope block will be removed in a future release. To connect signals to your floating scope, use the existing signal selector () instead. For more details, see “Add Floating Scope Block to Model and Connect Signals”.

Ability to dock Scopes to MATLAB desktop will be removed

Still runs

The ability to dock the Simulink Scope, Time Scope, Floating Scope and Scope Viewer to the MATLAB desktop will be removed in a future release.

SampleInput parameter will be removed*Still runs*

The `SampleInput` parameter of the Simulink Scope, Time Scope, Floating Scope and Scope Viewer (currently accessed through the command line API) will be removed in a future release.

Connection to Hardware

Support added for 64-bit Android applications

Use the Simulink Support Package for Android™ Devices to build and deploy Android applications on your device that supports 64-bit Android application packages (APK). You can use the **Android Device (64-bit)** option under **Hardware board** in the Configuration Parameters dialog box to generate a 64-bit APK for your Android application. Alternatively, you can use the **Create 64 bit library as well** option under **Android Device** hardware board to generate a 32-bit and a 64-bit APK for your Android application.

Support added to integrate algorithm to existing Android application

Use the Simulink Support Package for Android Devices to integrate an algorithm developed in Simulink into an existing Android application using Android Studio. Use the **Create Android library module for Simulink model** option in the Configuration Parameters dialog box to create a library module for the Simulink model that can be further integrated with the code of an existing Android application. This option is available under **App** options in the Configuration Parameters dialog box for both **Android Device** and **Android Device (64 bit)** hardware options.

Better visual display and auto rotate applications on Android devices

Starting R2022a, you can now navigate easily through the tabs of an application deployed on your Android device. If your model contains a Camera or Scope block from Simulink Support Package for Android Devices and is deployed on your Android device, you can automatically rotate the application with the Android device. Previously, you could not automatically rotate the application.

Support added for Arduino compatible ESP32-WROOM boards

The Simulink Support Package for Arduino® Hardware now supports deploying Simulink models on the ESP32-WROOM boards. The **ESP32-WROOM (Arduino Compatible)** option has been added to the **Hardware board** drop-down list in the Configuration Parameters dialog box. Starting R2022b, you can use these ESP32-WROOM boards:

- ESP32-WROOM-DevKitV1(30 pin)
- ESP32-WROOM-DevKitV1(36 pin)
- ESP32-WROOM-DevKitC(38 pin)

The support package also provides real-time execution profiling for the Simulink models deployed on the ESP32-WROOM boards. You can use these blocks from the support package library with the ESP32-WROOM boards.

Digital Input	Protocol Encoder	BNO055 IMU Sensor	LSM6DS0 IMU Sensor
Digital Output	Protocol Decoder	Tachometer	LSM6DSR IMU Sensor
Analog Input	External Interrupt	APDS9960 Sensor	LSM303C IMU Sensor
Analog Output	WiFi TCP/IP Receive	MPU9250 IMU Sensor	LPS22HB IMU Sensor

PWM	WiFi TCP/IP Send	MPU6050 IMU Sensor	HTS221 Humidity Sensor
Serial Receive	WiFi UDP Receive	LSM9DS1 IMU Sensor	Encoder
Serial Transmit	WiFi UDP Send	LSM6DS3 IMU Sensor	ADXL34x Accelerometer
I2C Write	WiFi ThingSpeak Read	LSM6DS3H IMU Sensor	CCS811 Air Quality Sensor
I2C Read	WiFi ThingSpeak Write	LSM6DSL IMU Sensor	ICM20948 IMU Sensor
SPI Write Read	Ultrasonic Sensor	LSM6DSM IMU Sensor	VL530x Time of Flight Sensor

Hardware support extended for real-time execution of generated code on Arduino boards

The Simulink Support Package for Arduino Hardware now provides real-time execution profiling for the Simulink models deployed on these Arduino boards:

- Arduino Uno
- Arduino Mega ADK
- Arduino Mega 2560
- Arduino Leonardo

Previously, real-time execution profiling was supported only on ARM® architecture-based Arduino boards.

Select serial communication port for external and PIL modes on Arduino boards

For Simulink Support Package for Arduino Hardware, you can now select any serial communication port on your Arduino board to run the models in either external (Monitor & Tune) or PIL modes. Previously, you could select only serial port 0 on your Arduino board.

Support extended for executing multitasking models for Arduino Nano 33 BLE Sense board

Prior to R2022a, using the Simulink Support Package for Arduino Hardware for executing and deploying multitasking models on Arduino Nano 33 BLE Sense board was not supported.

Starting in R2022a, you can deploy models containing blocks running at different sample rates on your Arduino Nano 33 BLE Sense boards.

Control pan and tilt motions using PCA9685-based PWM driver

With the addition of the Pan Tilt Hat block to the Simulink Support Package for Raspberry Pi™ Hardware, you can now control the pan and tilt motions of the pan-tilt hardware using the PCA9685 kernel module.

Adjust and set video and camera parameters using V4L2 Video Capture block in Raspberry Pi

Using the V4L2 Video Capture block from Simulink Support Package for Raspberry Pi Hardware, you can now adjust various parameters for an input video such as brightness, saturation, contrast, sharpness, and flip along vertical and horizontal axes. You can also adjust various parameters of your camera such as pan and tilt angles, zoom-in and zoom-out, and enable manual focus using the block.

Improved build and deployment speeds for Simulink Support Package for Raspberry Pi Hardware

Before R2022a, in the Simulink Support Package for Raspberry Pi Hardware, the generated code and the driver source code were compiled sequentially. From R2022a, the code is compiled in parallel, thus reducing the time taken to build models. You can change this default behavior and choose to build the model sequentially by clearing the **Enable parallel build** parameter under **Build options** in the Configuration Parameters dialog box.

Manually set XCP-based polling time for Simulink model running on Raspberry Pi

You can now manually set the XCP-based polling time for Simulink models that contain blocks from the Simulink Support Package for Raspberry Pi Hardware running in the external mode (Monitor & Tune). A new parameter, **Set XCP target polling time**, has been added in the Configuration Parameters dialog box. Previously, the polling time for Simulink models running in external mode was calculated automatically.

Higher data acquisition rates using Connected I/O with Arduino boards

The Simulink Support Package for Arduino Hardware now supports higher data acquisition rates for sensor/peripheral blocks with Arduino boards using Connected I/O. In the Connected I/O mode, use the sensor/peripheral blocks in the model, enable **Simulation Pacing**, and then set the **Simulation time per wall clock second** to 1 for a higher data acquisition rate.

Read distance to target object using VL53L0X sensor

The Simulink Support Package for Arduino Hardware now supports interfacing the VL53L0X sensor with Arduino hardware. You can use the new VL53L0X Time Of Flight Sensor block to measure the distance to a target object for a complete field of view. The block also provides the option to select one of four ranging modes based on your requirements.

Read linear acceleration, angular velocity, magnetic field, and temperature using ICM-20948 sensor

The Simulink Support Package for Arduino Hardware now supports interfacing the ICM-20948 nine-axis sensor with Arduino hardware. You can use the new ICM20948 IMU Sensor block to measure linear acceleration, angular velocity, and magnetic field along the X, Y and Z-axes, and to measure temperature. The block also provides the option to generate the data ready interrupt.

Read eCO₂ and eTVOC using CCS811 sensor

The Simulink Support Package for Arduino Hardware now supports interfacing the CCS811 sensor with Arduino hardware. You can use the new CCS811 Air Quality Sensor block to measure the equivalent CO₂ (eCO₂) and the equivalent total volatile organic compound concentration (eTVOC) for indoor air quality monitoring. The block also provides the option to use temperature and humidity data to enable compensation (either by specifying the humidity and temperature values or by interfacing CCS811 with an external sensor).

Read linear acceleration using ADXL34x family of accelerometers

The Simulink Support Package for Arduino Hardware now supports interfacing the ADXL34x family of accelerometers (ADXL343, ADXL344, ADXL345, and ADXL346) with Arduino hardware. You can use the new ADXL34x Accelerometer block to measure linear acceleration along the X, Y and Z-axes. The block also provides the option to enable the data ready interrupt.

MATLAB Function Blocks

Cell Array Type for Non-Tunable Parameters

In MATLAB Function blocks, you can use cell arrays as non-tunable parameter data. To use cell-arrays as non-tunable parameters in your MATLAB Function block, see “Configure MATLAB Function Block Parameter Variables” and ensure that the **Tunable** option is cleared.

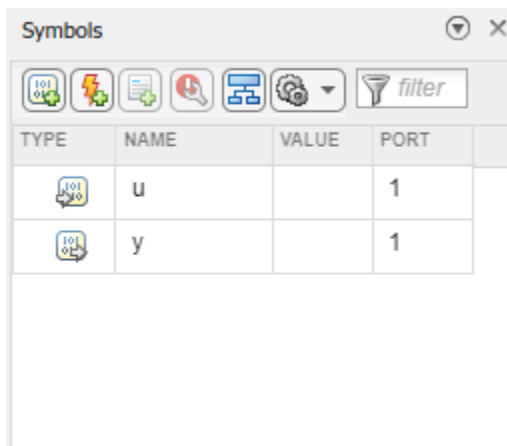
Variable Size property applies only to output variables

Starting in R2022a, the **Variable size** property only applies to MATLAB Function block variables with the **Scope** property set to **Output**. Input variables now inherit size variability from their corresponding signals. Prior to R2022a, this property could also be applied to MATLAB Function block variables with **Scope** set to **Input**. Updating a model created in a previous release disables this property for input variables.

You can still specify the property for variables used in both the input and output of the function declaration statement. For example, if you define the function $y = \text{myFunction}(y)$, the variable y has this property.

Symbols pane replaces Ports and Data Manager

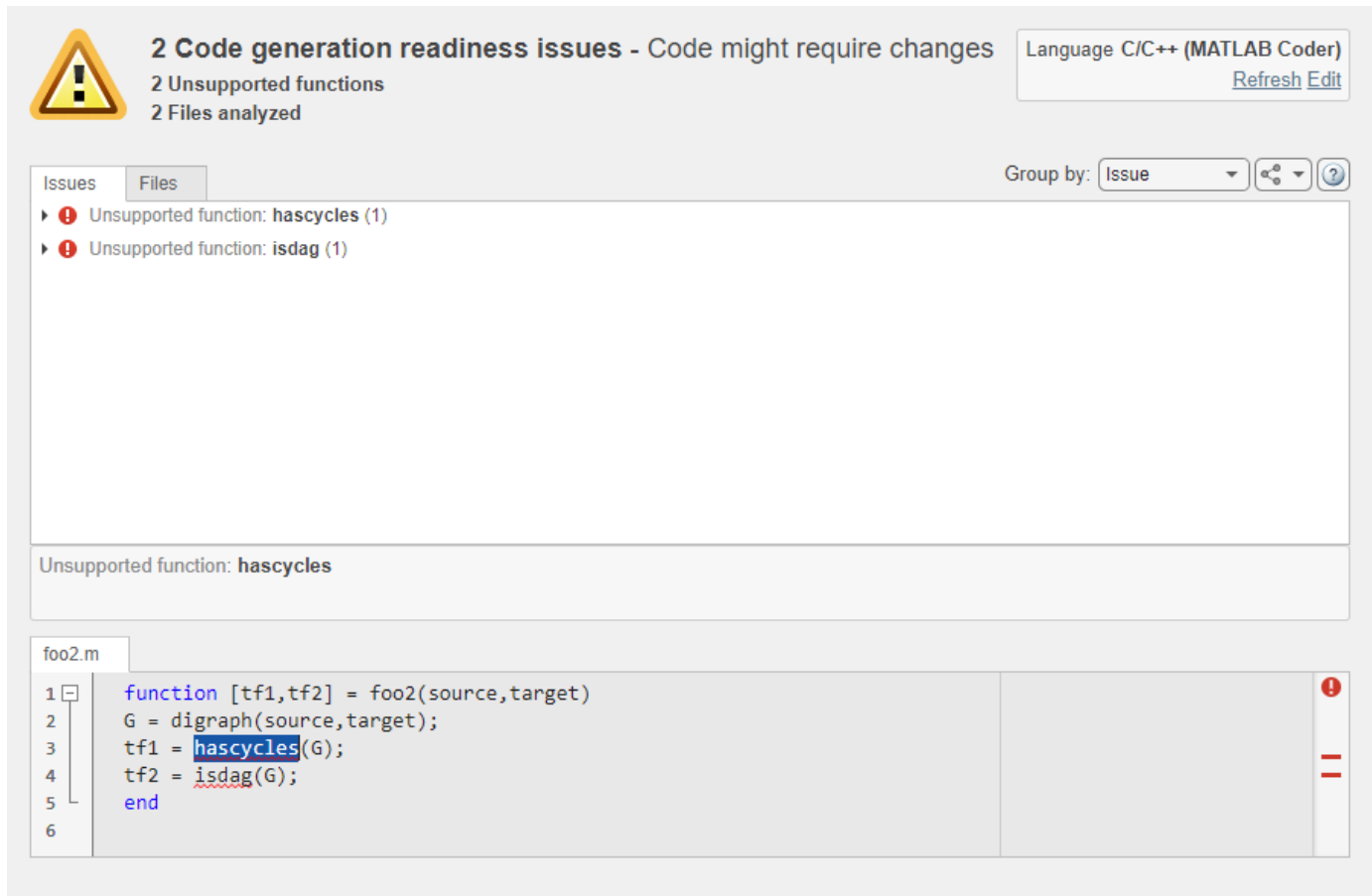
In R2022a, the Ports and Data Manager has been removed. To manage variables, function call outputs, and input triggers in MATLAB Function blocks, use the **Symbols** pane and the Property Inspector.



For more information on managing variables, function call outputs, and input triggers with the **Symbols** pane and Property Inspector, see “Create and Define MATLAB Function Block Variables”, “Manage Function Call Outputs of a MATLAB Function Block”, and “Manage the Input Trigger of a MATLAB Function Block”.

New Code Generation Readiness Tool: View more information and navigate through readiness results more easily

In R2022a, the Code Generation Readiness Tool has a new user interface, more information, additional functionality, and improved navigation. In addition, you can now use the Code Generation Readiness Tool in MATLAB Online.



The screenshot displays the Code Generation Readiness Tool interface. At the top left, there is a warning icon and the text "2 Code generation readiness issues - Code might require changes". Below this, it shows "2 Unsupported functions" and "2 Files analyzed". On the top right, the language is set to "C/C++ (MATLAB Coder)" with "Refresh" and "Edit" buttons. The main area is divided into "Issues" and "Files" tabs. Under the "Issues" tab, two issues are listed: "Unsupported function: hascycles (1)" and "Unsupported function: isdag (1)". Below the issues list, a detailed view of the "Unsupported function: hascycles" issue is shown. At the bottom, a code editor window for "foo2.m" is open, showing the following MATLAB code:

```

1 function [tf1,tf2] = foo2(source,target)
2   G = digraph(source,target);
3   tf1 = hascycles(G);
4   tf2 = isdag(G);
5   end
6

```

The code line `tf1 = hascycles(G);` is highlighted in blue, and a red exclamation mark icon is visible in the right margin of the code editor.

In addition to the existing functionalities, you can now:

- View your MATLAB code inside the Code Generation Readiness Tool. When you select an issue, the part of your MATLAB code that caused this issue gets highlighted.
- Group the readiness results either by issue or by file.
- Select the language that the code generation readiness analysis uses.
- Refresh the code generation readiness analysis if you updated your MATLAB code.

See “Check Code Using the Code Generation Readiness Tool” and `coder.screener`.

coder.ScreenerInfo object: Access code generation readiness information programmatically

In R2022a, you can export the code generation readiness information about your MATLAB code to a variable in your base workspace. This variable contains a `coder.ScreenerInfo` object whose properties contain information about:

- MATLAB files analyzed by the Code Generation Readiness Tool
- Code generation readiness messages
- Calls to functions not supported for code generation

To export code generation readiness information about your the files `foo1.m`, `foo2.m`, and `foo3.mlx` to the variable `info` in your base workspace, execute this function call:

```
info = coder.screener('foo1.m', 'foo2.m', 'foo3.mlx')
```

You can also export the entire report to a MATLAB string by executing the object function `textReport`:

```
reportString = textReport(info)
```

See:

- Reference pages: `coder.ScreenerInfo` Properties and `coder.screener`
- Example: “Access Code Generation Readiness Results Programmatically”

Code generation for more MATLAB functions

- `integral`
- `interpft`
- `ldl`
- `mustBeFloat`
- `mustBeNonmissing`
- `mustBeScalarOrEmpty`
- `mustBeText`
- `mustBeTextScalar`
- `mustBeVector`
- `groupcounts`
- `groupsummary`

Code generation for more toolbox functions

In R2022a, you can generate code for many additional toolbox functions and objects. For a list of all functions and objects that are supported for code generation, see:

- [Functions and Objects Supported for C/C++ Code Generation \(Category List\)](#)
- [Functions and Objects Supported for C/C++ Code Generation \(Alphabetical List\)](#)

Modeling Guidelines

High-Integrity Systems Modeling Checks: Improve quality and compliance with guidelines

In R2022a, you can use these high-integrity modeling guidelines:

High-Integrity Guideline	Equivalent Model Advisor Check
hisl_0075: Usage of library links	Check for disabled and parameterized library links
hisl_0101: Avoid operations that result in dead logic to improve code compliance	Check for unreachable and dead code

Version History

These guidelines were removed in R2022a:

High-Integrity Guideline	Corresponding Modeling Check
hisf_0003: Usage of bitwise operations	Check usage of bitwise operations in Stateflow charts
hisf_0009: Strong data typing (Simulink and Stateflow boundary)	Check for Strong Data Typing with Simulink I/O

S-Functions

Programmatically configure S-functions using the S-Function Builder block

Starting in R2022a, you can use functions to configure an S-Function Builder block to build an S-function for your model. You can use the functions to:

- Configure settings, including build options and options to support model coverage and design verification.
- Specify user code for the S-function methods.
- Compile the S-function source and MEX.

For more information, see “Implement C/C++ Code Using S-Function Builder”.

Functionality being removed or changed

Level-1 Fortran S-function will be removed in a future release

Still runs

Level-1 Fortran S-functions will be removed in a future release. Use Level-2 Fortran S-functions instead. To learn more about Level-2 Fortran S-functions, see “Create Level-2 Fortran S-Functions” and explore the `sfcndemo_atmos` example.

R2021b

Version: 10.4

New Features

Bug Fixes

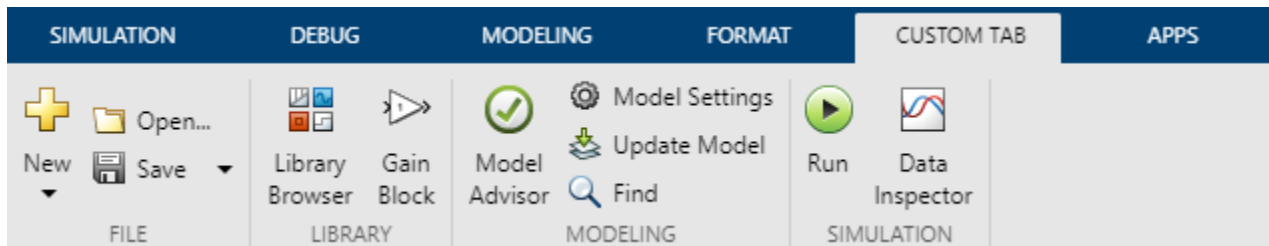
Version History

Simulink Editor

Create custom Simulink Toolstrip tabs

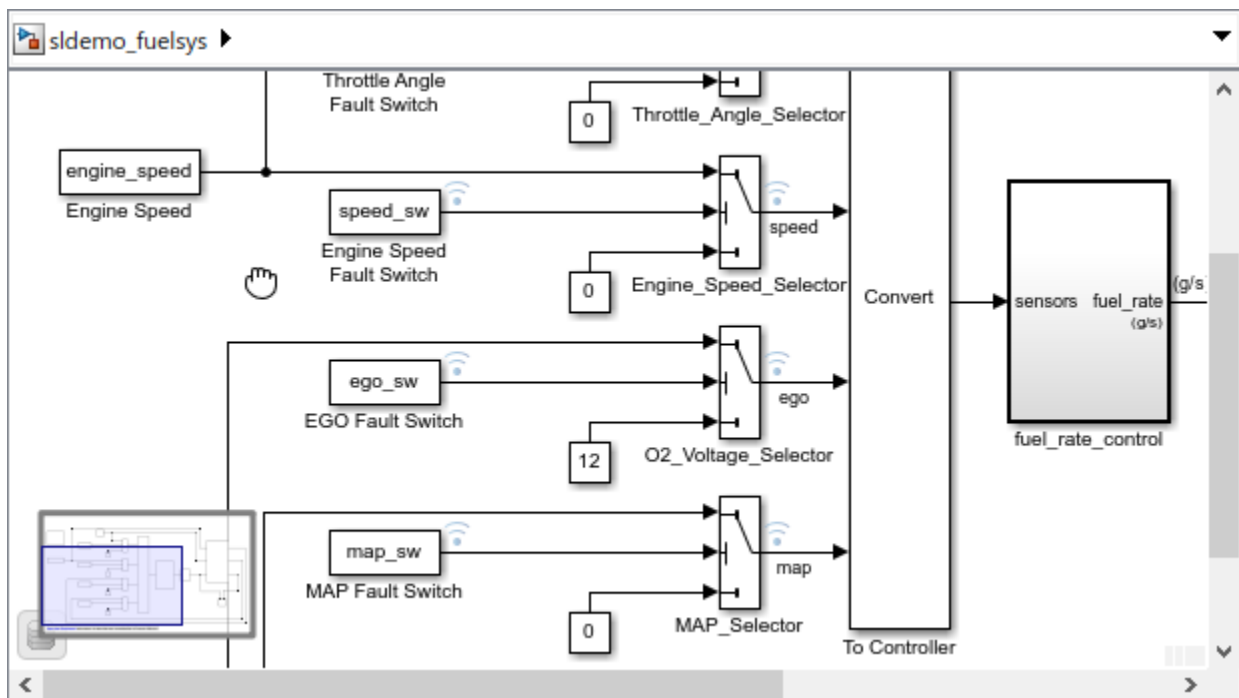
You can now customize the Simulink Toolstrip by adding tabs that focus on workflows specific to you and your team. Custom tabs provide a platform for performing custom actions that you develop. You can also populate custom tabs with built-in actions from the default Simulink Toolstrip tabs.

For more information about creating custom tabs, see [Create Custom Simulink Toolstrip Tabs](#). For an example of a prepopulated custom tab, see [Add Prepopulated Custom Tab to Simulink Toolstrip](#).



Miniature map provides context when viewing part of a block diagram

When you zoom or pan to part of a block diagram in the Simulink Editor canvas, a miniature map appears. The highlighted area in the map indicates the visible portion of the diagram relative to the full block diagram.



The miniature map helps you navigate large block diagrams.

- To view a different portion of the diagram, click the destination in the miniature map.
- To quickly pan around the diagram, drag the highlighted area in the miniature map.

When only part of the block diagram is visible in the canvas, display the miniature map without panning or zooming by pressing and holding the space bar until the miniature map appears.

Get names of built-in Simulink Toolstrip actions and icons



To get built-in action and icon names from the Simulink Toolstrip, use the new `slToolstripDeveloperMode` function. Built-in toolstrip actions define the behavior and appearance of controls, such as push buttons, in the Simulink Toolstrip.

Use the built-in action and icon names to:

- Add built-in actions to custom Simulink Toolstrip tabs. For more information, see [Create Custom Simulink Toolstrip Tabs](#).
- Disable built-in actions on default Simulink Toolstrip tabs. For more information, see [Disable Simulink Toolstrip and Context Menu Actions](#).

Import, export, reset, and reorder favorite commands in Simulink

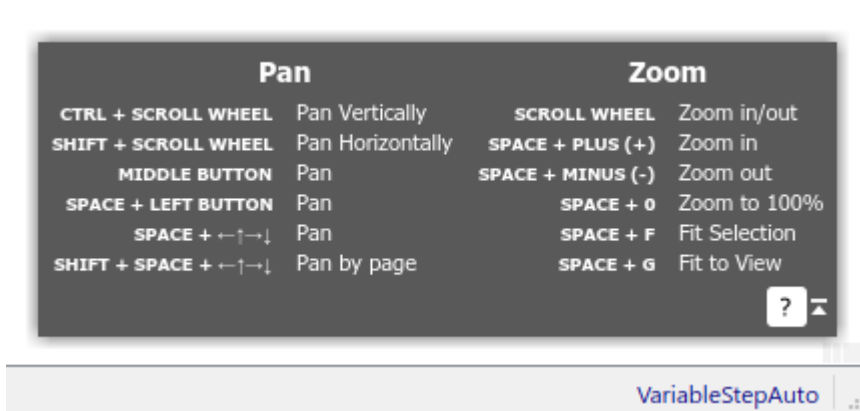
You can now perform these actions with the Simulink **Favorite Commands** gallery:

- Import favorite commands — Use the `slImportFavoriteCommands` function. During import, you receive the option to merge the imported commands with existing commands or completely overwrite the contents of the gallery.
- Export favorite commands — Use the `slExportFavoriteCommands` function. The generated file saves the contents of the gallery, including the categories and their order.
- Reset favorite commands — Use the `slResetFavoriteCommands` function.
- Reorder favorite command categories — Click the **Move category to top**  and **Move category to bottom**  buttons to the right of the category names.

For more information about favorite commands in Simulink, see [Access Frequently Used Features and Commands in Simulink](#).

View how to pan and zoom in the Simulink Editor by pressing and holding the space bar

To view keyboard shortcuts and mouse actions for panning and zooming in the Simulink Editor, press and hold the space bar until the hints panel appears at the bottom right of the canvas.



To hide the panel, release the space bar. To minimize the panel while you hold the space bar, click the panel or press the ? key.

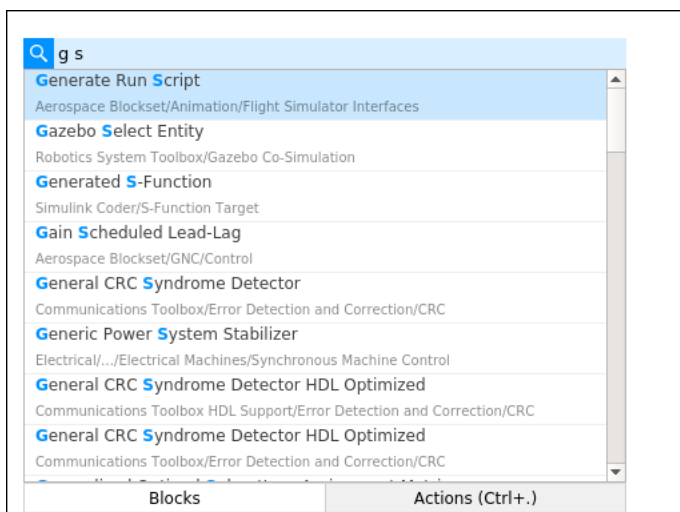
For additional keyboard shortcuts and mouse actions, see Keyboard Shortcuts and Mouse Actions for Simulink Modeling.

Signal Tracing Command Line API

Use the new signal tracing command line interface to programmatically trace a selected signal to its source or destination blocks. Create `sltrace.Graph` objects using the `sltrace` function.

Improved search experience in quick insertion of block

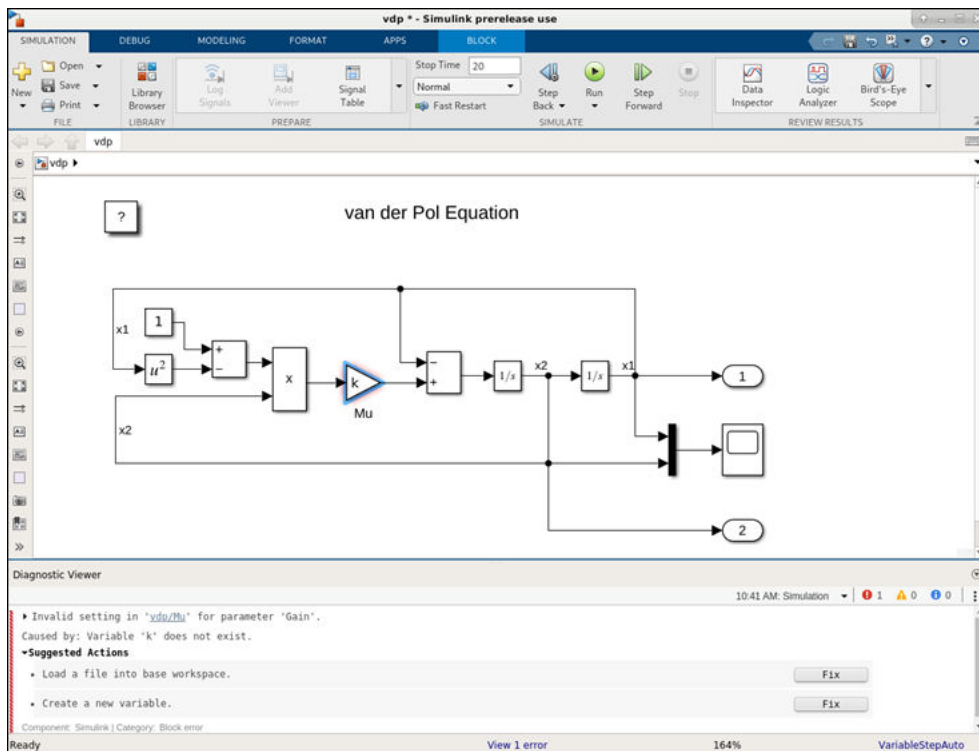
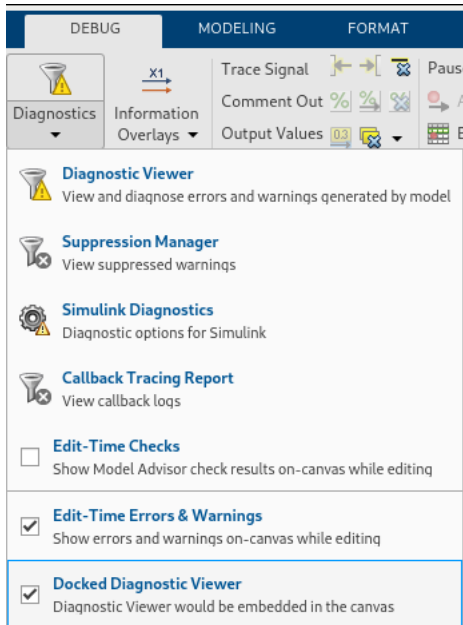
Search keywords, including special characters, are highlighted in search results. Search scope includes block type, block class, and block keywords.



Dock Diagnostic Viewer to a model

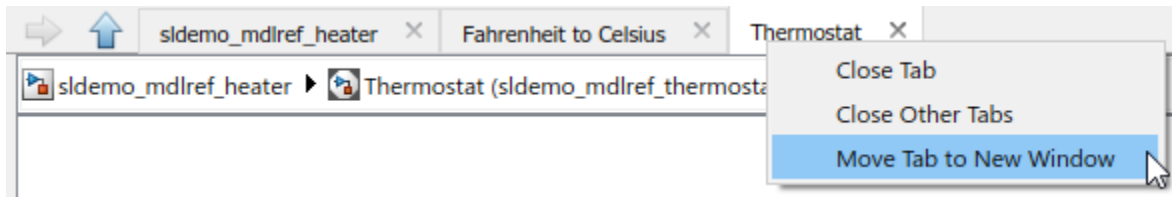
You can choose to dock the Diagnostic Viewer to a model or have a common Diagnostic Viewer for all currently open models. Docking the Diagnostic Viewer to a model allows you to quickly resolve the errors and warnings specific to the model.

To dock the Diagnostic Viewer to a model, go to **Diagnostic Viewer > Docked Diagnostic Viewer**. For more information, see View Diagnostics



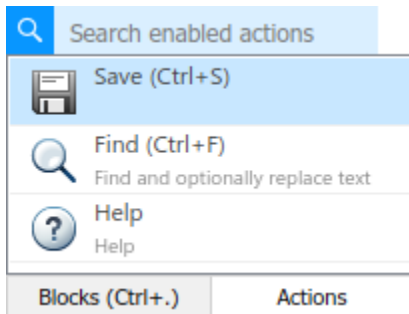
Move Simulink Editor tab to a new window

To move a Simulink Editor tab to a new window, you can now right-click the tab and select **Move Tab to New Window**.



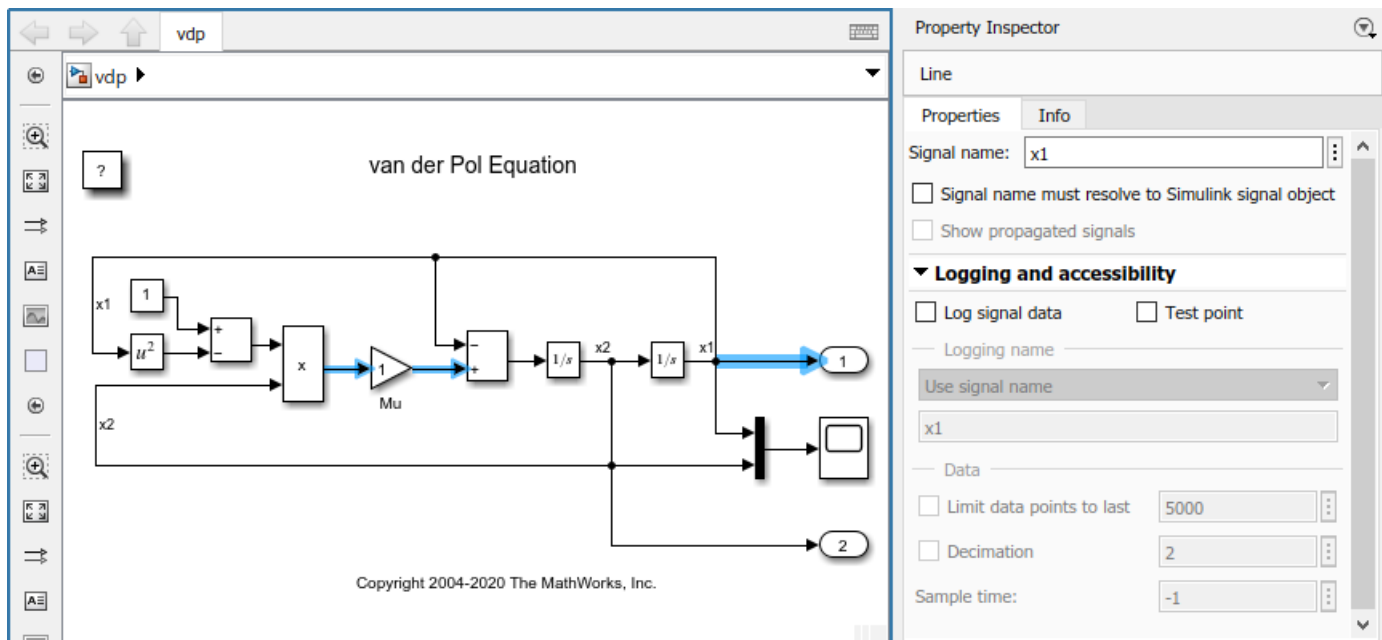
Quick action search menu provides default results


To help you get started with the quick action search menu, it now provides default results.




Visually identify line or port as primary selection

When the element you most recently selected is a line or a port, you can now visually identify it as the primary selection because it has a thicker outline than other selected lines and ports.



When a selection contains multiple blocks and the primary selection is not a block, alignment actions are based on all the blocks in the selection. For example, selecting **Align right**  on the **Format** tab of the toolstrip aligns the right edge of the selected blocks with the rightmost block edge in the

selection. Similarly, selecting **Align left**  aligns the left edge of the selected blocks with the leftmost block edge in the selection.

Display of port label information and automatic editing of port labels on Subsystem blocks

In R2021b, port label editing on Subsystem blocks is enhanced to:


- Indicate the port label information (port block name, signal name, bus port, or connection port) that is being edited.
- Switch between port block name and signal name for signal ports directly from the port label.
- Select consecutive ports using the arrow keys for quick editing.
- Enable automatic port label editing after port creation.

Previously, you could not identify the type of information displayed on the port labels of a Subsystem block and could only edit the displayed values. See [Edit Port Labels on Subsystems](#).

Display related blocks in open diagrams or new tabs

When you select one of the following blocks, you can now display each of its related blocks in an open diagram or new tab.

- Goto, From, or Goto Tag Visibility block
- Inport block that has duplicates
- In Bus Element block that has duplicates
- Data Store Read, Data Store Write, or Data Store Memory block
- Simulink Function or Function Caller block
- State Reader, State Writer, Initialize Function, Reset Function, Terminate Function, or a corresponding state owner block
- Connection Label (Simscape) block

After you select one of these blocks, pause on the ellipsis. Then, select **Related Blocks**  from the action bar. When multiple blocks correspond to the selected block, a list of related blocks opens. You can filter the list of related blocks by entering a search term in the text box. After you select a related block from the list, window focus goes to the open diagram or new tab that shows the related block.

Propagate tag change from Goto block to related From and Goto Tag Visibility blocks

When you change the tag on a Goto block icon, you can now propagate the change to the corresponding From and Goto Tag Visibility blocks by pressing **Shift+Enter**.

Enhancements to right-click context menu

To simplify the context menu that appears when you right-click in the Simulink Editor, menus with similar goals have been combined and uncommon actions have been removed.

Context	Removed Context Menu Item	Alternative
Model component	Open	Double-click the block.
Block	Signals & Ports menu	Select a signal or port to modify it.
Signal	Format menu	Arrange Automatically now appears at the top level of the context menu.
Simulink Editor canvas	Recently Added menu	Double-click the Simulink Editor canvas and use the quick insert menu, which prioritizes recently added blocks.
Selection with one or more elements	Delete	Press the Delete key.
Selection with one or more elements	Format > Font Style for Selection	In the toolbar, on the Format tab, click  .
Selection with one or more blocks or annotations	Format > Shadow	In the toolbar, on the Format tab, click Shadow .
Selection with one or more blocks or annotations	Rotate & Flip menu	Find the relocated menu items under Format in the context menu.
Selection with multiple blocks or annotations	Arrange menu	Find the relocated menu items under Format in the context menu.
Selection with multiple blocks or annotations	<ul style="list-style-type: none"> • Arrange > Distribute Horizontally • Arrange > Distribute Vertically • Arrange > Even Horizontal Gaps • Arrange > Even Vertical Gaps 	In the toolbar, on the Format tab, click the corresponding button in the Distribute section.
Selection with multiple blocks or annotations	<ul style="list-style-type: none"> • Arrange > Match Width • Arrange > Match Height • Arrange > Match Size 	In the toolbar, on the Format tab, click the corresponding button in the Match section.

Simulation Analysis and Performance

Run multiple simulations for different scenarios from the Simulink Editor

In R2021b, use the Multiple Simulations panel in the Simulink Editor to run multiple simulations for scenarios specified in the Signal Editor block. The Multiple Simulations panel also allows you to set up multiple simulations for a block parameter to perform operations like parameter sweep.

Select the block parameter from the Simulink canvas, specify values for the simulation, and use the Multiple Simulations panel to configure the simulations. Once the simulations start, use the Simulation Manager to monitor progress and results of your simulations.

Solver Switching with Fast Restart

Starting R2021b, you do not need to recompile the model every time the solver is changed. With fast restart you can change the solver from run to run without going through a recompile.

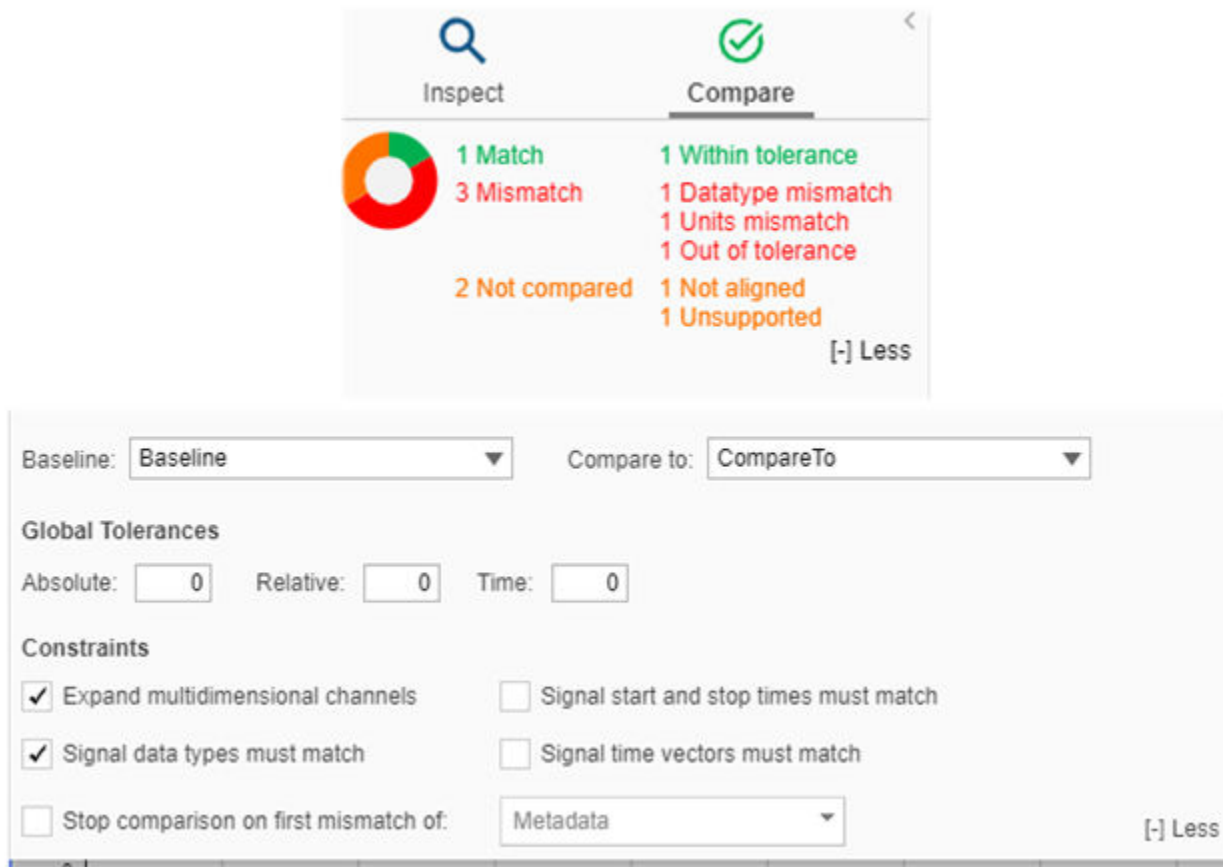
Configure comparison constraints using the Simulation Data Inspector UI

Starting in R2020a, you could specify constraints for run comparisons performed programmatically using the `Simulink.sdi.compareRuns` function. Starting in R2021b, you can configure the same set of constraints for comparisons you run using the Simulation Data Inspector UI. The results summary also includes details about the source of mismatches.

For example, you can add a comparison constraint that signal data types must match, which means the comparison checks whether the signal data types match before comparing the signals. If the signal data types do not match, the signal comparison does not proceed, and the signal comparison result has a status of `DataTypeMismatch`.

Using the `Simulink.sdi.compareRuns` function or the Simulation Data Inspector UI, you can configure:

- Whether to expand multidimensional signals during comparison to produce comparison results for each element, or channel.
- Whether comparisons are sensitive to signal data type, start and stop time, or time vector mismatches.
- Whether a run comparison continues after detecting a mismatch in signal data or metadata.

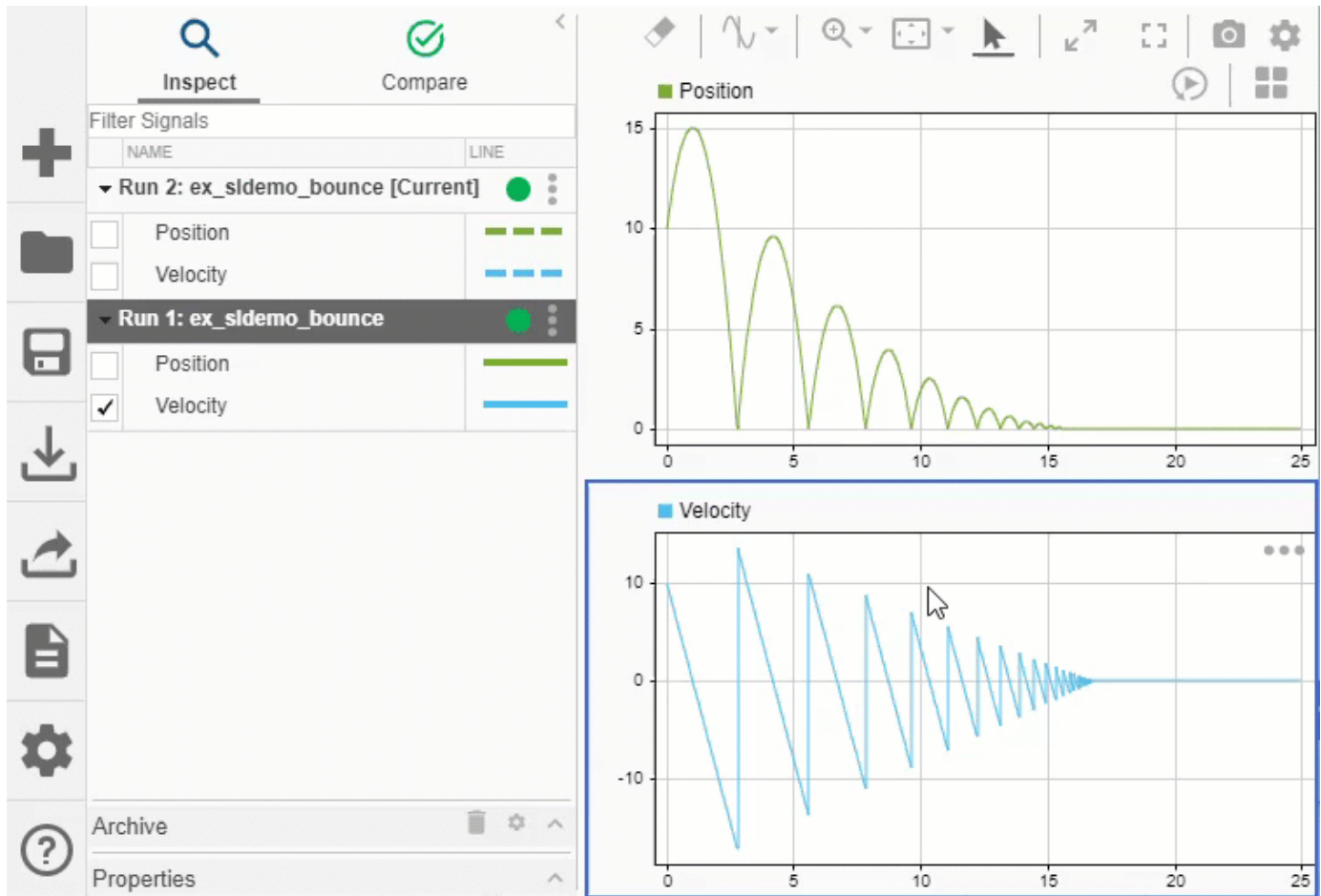


For more information, see [Compare Simulation Data](#).

Copy, cut, and paste plotted signal selection from one run to another in the Simulation Data Inspector

You can easily control which run data is plotted in the Simulation Data Inspector using new options to cut, copy, and paste the plotted signal selection from one run to another.

Suppose you want to view data from two simulations side by side in the Simulation Data Inspector. First, build the view you want by modifying the subplot layout and visualizations. Then, plot the data from the first run by selecting the subplot where you want to plot a signal and then selecting the check box for the signal you want to plot. Starting in R2021b, you can easily update the plots by cutting or copying the plotted signal selection from one run to paste onto another. For example, to plot data from both runs side by side, you can copy the plotted signal selection from the first run and paste it onto the second.

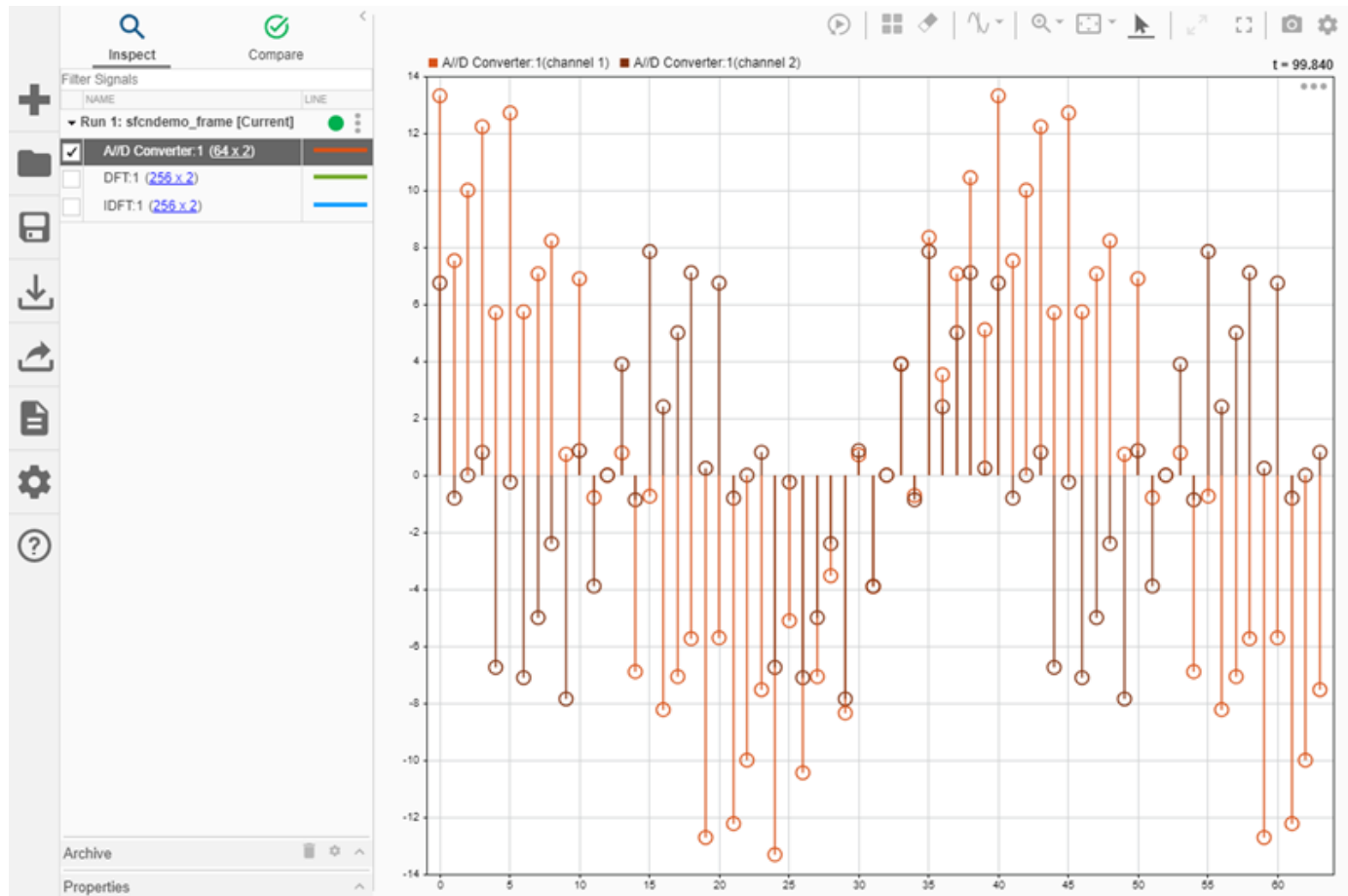


To swap which run data is shown on the plots, cut the plotted signal selection from a run and paste it onto another. You can paste a plotted signal selection onto any number of runs. The **Show run actions** menu also has options to clear plotted signals from the selected run or from all runs other than the selected run.

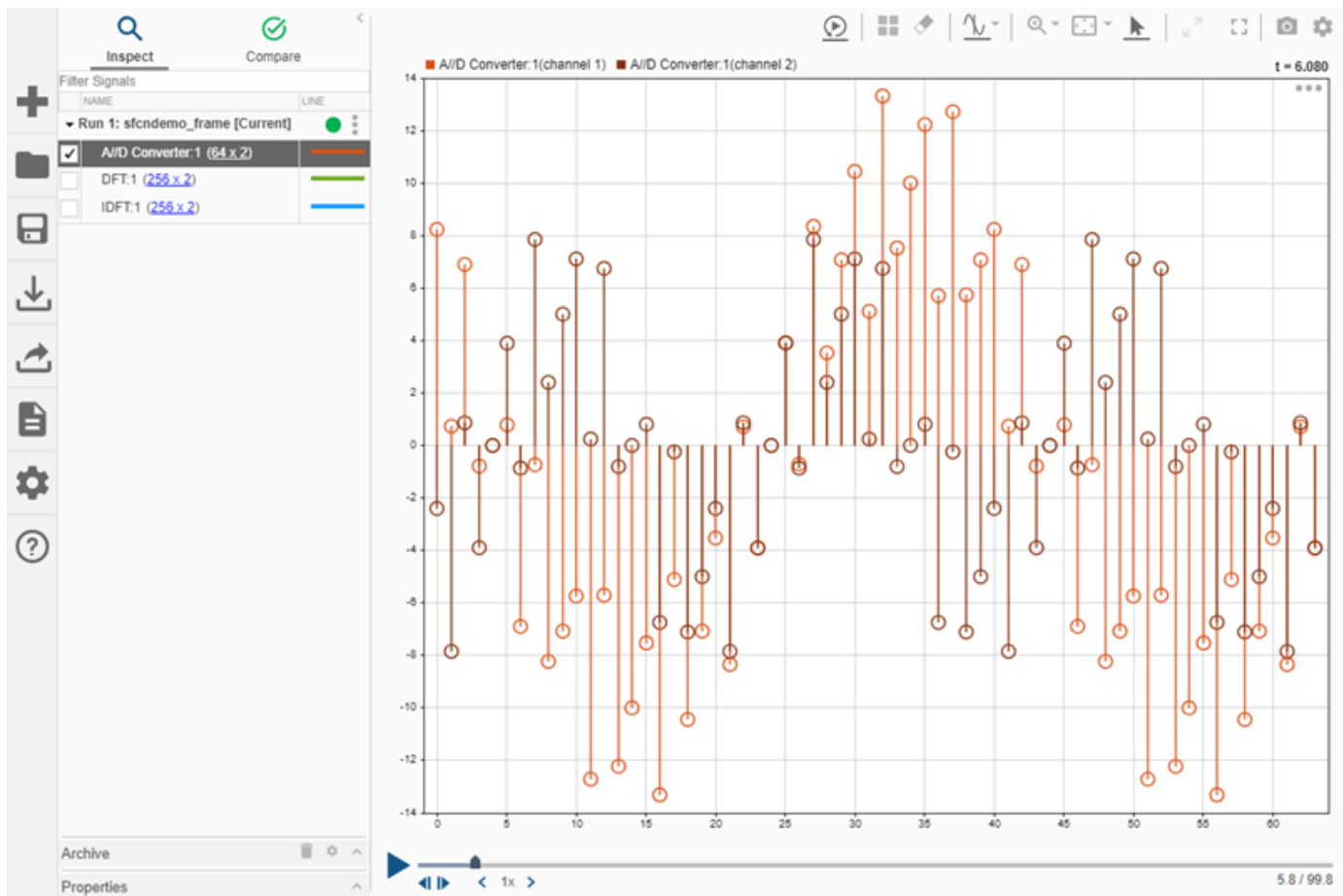
For more information, see [Analyze Data from Multiple Simulations](#).

View multidimensional signal data on an array plot in the Simulation Data Inspector

Starting in R2021b, the Simulation Data Inspector supports representing multidimensional time series data as a single signal with multidimensional sample values. You can view the multidimensional signal data on an array plot, also new in R2021b. The array visualization plots the vector or array values for a single sample of a multidimensional signal, evenly spaced along the x-axis. You can specify an offset for the x-axis and an increment value that defines the space between values on the x-axis.



The time that corresponds to the plotted sample is displayed in the upper-right corner of the array plot. To view how the multidimensional data changes over time, you can use the replay controls or add a time plot and use a cursor to control which sample is plotted on the array plot. When you use the replay controls, you can click the Play button to automatically sweep through time at a rate you can control, or you can use the Step forward and Step backward buttons to step through the signal values sample by sample.



For more information, see [View Multidimensional Signals Using the Array Plot](#).

Export video data to an MP4 file using the Simulation Data Inspector

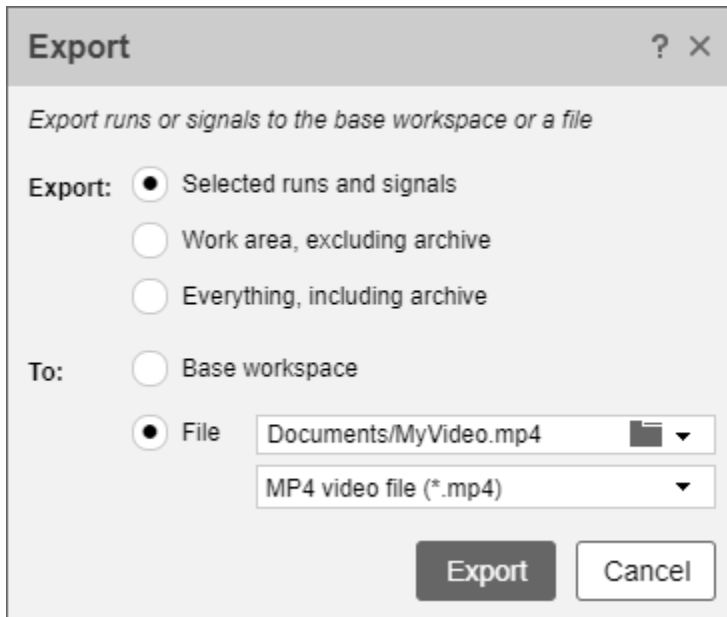
You can export a 2D or 3D signal that contains RGB or monochrome video data to an MP4 file using the Simulation Data Inspector. To export to an MP4 file, the multidimensional data must be represented in the Simulation Data Inspector as a single signal. If the data is represented as a set of signals, you can convert the data representation before exporting it.

Exporting to an MP4 file is not supported for Linux systems, and the signal data type must be `double`, `single`, or `uint8`. You can only export one video signal to an MP4 file at a time.

To export a video signal using the Simulation Data Inspector UI:

- 1 Right-click the signal and select **Export**, or select the signal and click **Export** in the toolbar on the left.
- 2 In the Export dialog box, choose to export **Selected runs and signals** to a file.
- 3 Specify a file name and the path to the location where you want to save the file.
- 4 Select **MP4 video file** from the drop-down and click **Export**.

The option to export to an MP4 file is only available when you choose to export **Selected runs and signals**, only one signal is selected, and the signal contains 2D or 3D data.



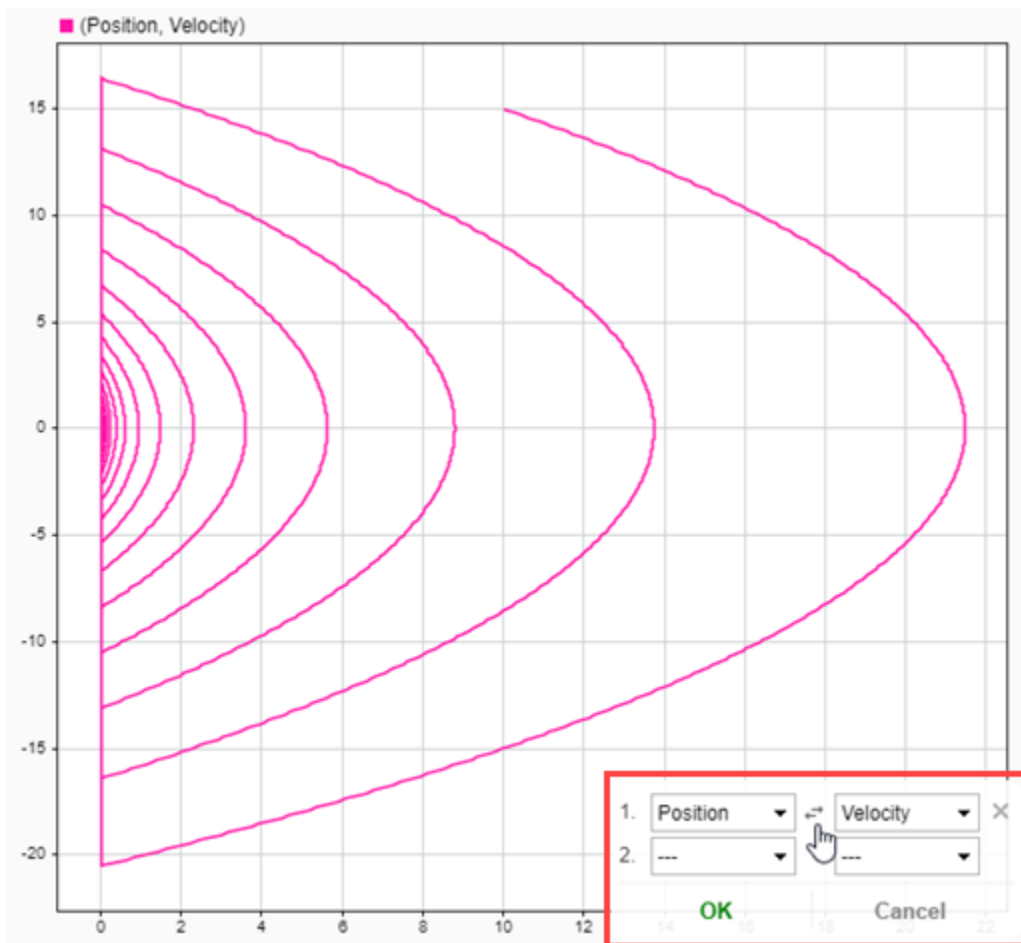
You can also export video data to an MP4 file programmatically using the `export` function for the `Simulink.sdi.Signal` object that contains the video data. Specify the `'to'` name-value argument as `'file'`, and specify the `'filename'` name-value argument as a path and file name that includes the `.mp4` extension. For example, to export video data contained in the `Simulink.sdi.Signal` object `sigObj` to the MP4 file `myVideo.mp4`, use this command:

```
export(sigObj, 'to', 'file', 'filename', 'myVideo.mp4');
```

Easily swap x and y data on XY plots in the Simulation Data Inspector

The XY visualization in the Simulation Data Inspector now allows you to easily swap which signal provides the x data for the plot and which provides the y data. After you plot the x and y data on the XY visualization, to swap which signal provides which data:

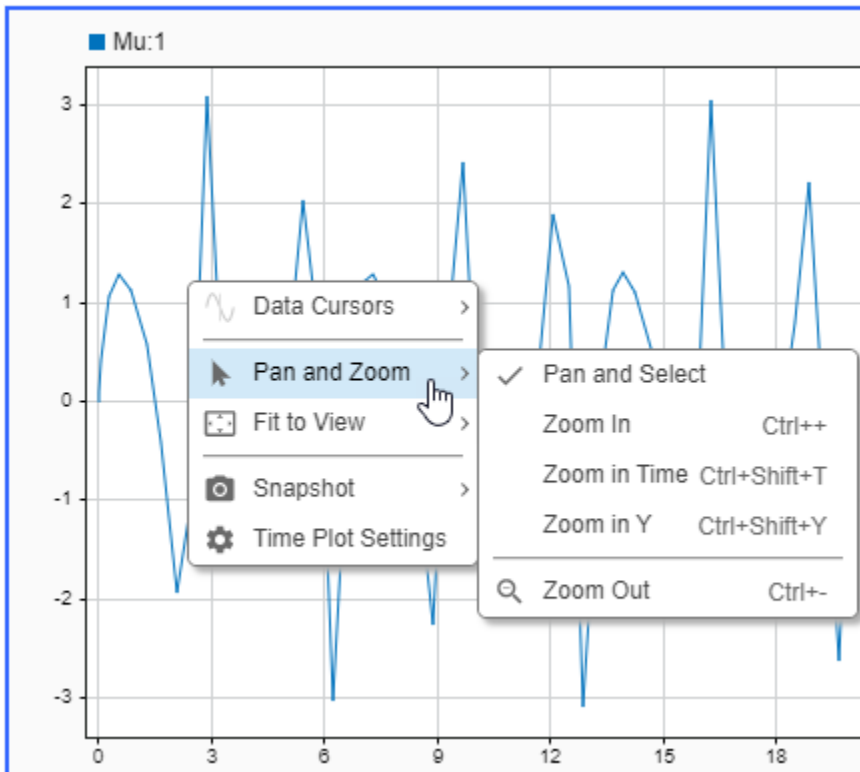
- 1 Right-click the XY plot and select **Show plotted signals**.
- 2 In the bottom-right of the XY plot, between the drop-down menus you use to select the x and y data, click **Swap**.
- 3 To see the updated plot, click **OK**.



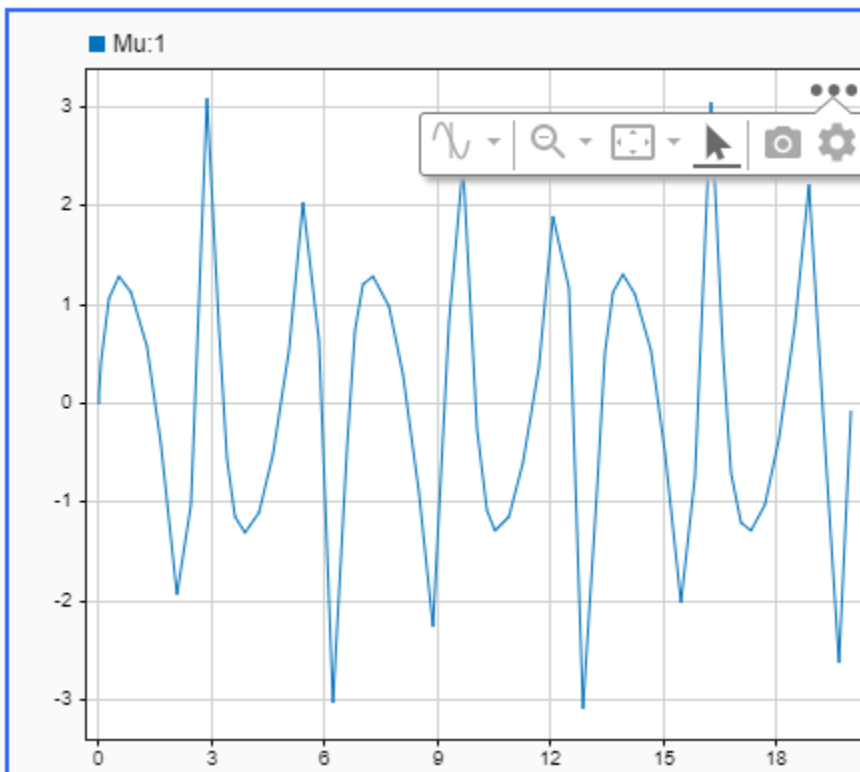
Simulation Data Inspector UI enhancements

The Simulation Data Inspector UI has several usability enhancements:

- Clear and consistent icons facilitate navigation.
- Condensed and consistent context menus enable quick access to functionality.



- A new subplot menu provides easy access to plot actions. To access the menu, click the dots that appear when you pause on a subplot.



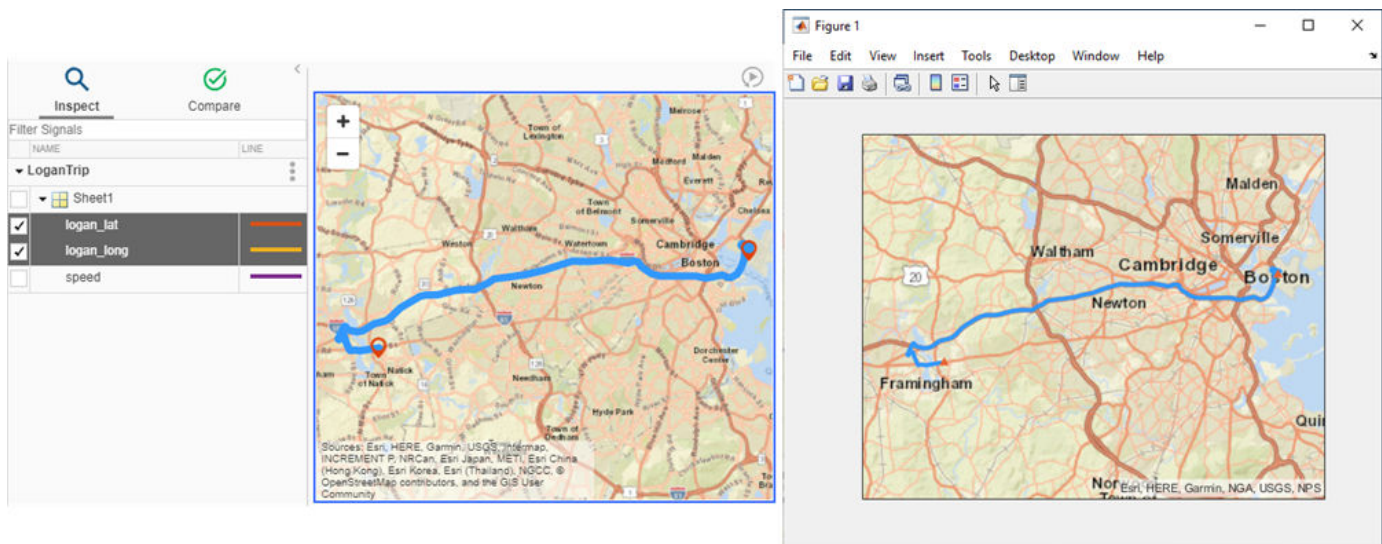
Interactive comparison report enhancements

Interactive comparison reports created using the Simulation Data Inspector have these enhancements:

- You can resize each column in the signals table and in the properties table by clicking and dragging between the column headers.
- When you resize the panes in the report, the headers for the signals table and the properties table maintain proper alignment with the table columns.

Simulation Data Inspector snapshot supports capturing a map visualization in a MATLAB figure

You can export a route plotted on the map visualization in the Simulation Data Inspector to a MATLAB figure. You can create the figure using the **Snapshot** option in the Simulation Data Inspector UI or with the snapshot programmatic interface by using the `Simulink.sdi.snapshot` function or a `Simulink.sdi.CustomSnapshot` object.



SIMD hardware acceleration support for Just-in-Time (JIT) simulation

In R2021b, SIMD is enabled by default for Accelerator mode simulation (JIT). To turn off SIMD for simulation targets including Accelerator mode (JIT), set the model parameter `SimHardwareAcceleration` to `Off` or on the **Modeling** tab, select **Model Settings**, then on the **Simulation Target** pane, under **Advanced parameters**, set the **Hardware acceleration** parameter to `Off`. For more information, see [Hardware acceleration](#).

Simulation enhancements for referenced models with multiple normal-mode instances

When simulating a model hierarchy with multiple normal-mode instances of a referenced model:

- The model compilation phase of simulation finishes faster. Previously, simulation compiled a block diagram for each normal-mode instance of a referenced model. Now, simulation compiles the block diagram once and reuses it for all normal-mode instances of the referenced model.
- Display blocks now show instance-specific data for each instance of the referenced model that simulates in normal mode. For more information, see [Simulate Multiple Referenced Model Instances in Normal Mode](#).

Extract data from multiple elements of a Simulink.SimulationData.Dataset object and put the data into timetables using the extractTimetable function

You can use the `extractTimetable` function to extract data from multiple elements of a `Simulink.SimulationData.Dataset` object into one or more timetables. You can return the data as:

- A single timetable that includes all elements that contain time series data, synchronized to the union of the time stamps of all signals
- A cell array of timetables that each contain data for signals that share the same sample time
- A cell array of timetables that each contain time series data for a single signal

Use runtime callbacks to set inport data and output data at the root level of a Simulink model

In R2021b, you can use runtime callbacks for interactivity with the `sim` function. Runtime callbacks enables you to set root inport data, and get root output data of a Simulink model. With runtime callbacks you can specify a different callback for each signal, specify and access user data within callbacks, and access port information within callbacks. Runtime callbacks also support the use of non-virtual buses.

Functionality being removed or changed

Improved handling of multidimensional data in the Simulation Data Inspector

Behavior change

The Simulation Data Inspector now supports representing multidimensional time series data as a single signal with multidimensional sample values. In prior releases, multidimensional data was represented as a set of signals with scalar sample values: one signal, called a channel, for each element of the multidimensional data.

By default, starting in R2021b, multidimensional data with more than four elements is represented in the Simulation Data Inspector as a single signal, and multidimensional data with four or fewer elements is still represented as a signal for each channel. Once the data is in the Simulation Data Inspector, you can convert the data representation as desired, including interpreting a multidimensional signal as frame-based.

You can view data for multidimensional signals using the array visualization, added in R2021b, or you can convert the multidimensional signal to channels and plot the channel data on a time plot, sparklines plot, XY plot, or map. When you apply a view, the Simulation Data Inspector converts multidimensional signals to channels if the view plots data for a channel in the multidimensional signal.

By default, multidimensional signals are converted to channels during comparison, and the comparison includes signal comparison results for each channel. If you do not need comparison results for multidimensional signals, you can configure the comparison to skip multidimensional signals instead of converting them to channels.

For more information, see [Analyze Multidimensional Signal Data](#).

Version History

The number of signals in a run may change for models that log multidimensional signal data. Depending on how you process the data, you may need to update scripts to access and process the data in a multidimensional signal or to convert the data representation.

- To access multidimensional signal data that is represented as channels using a single `Simulink.sdi.Signal` object, use the new `collapse` function to convert the data representation.
- To access channel data for a multidimensional signal represented as a single `Simulink.sdi.Signal` object, use the new `expand` function to convert the data representation.

Logging intervals configuration parameter applies to data logged to the workspace using the Record block

Behavior change

In R2021a, the value specified for the **Logging intervals** configuration parameter did not affect signals logged using the Record block. Starting in R2021b, the **Logging intervals** configuration parameter applies to data logged to the workspace using the Record block.

The **Logging intervals** parameter does not affect data logged to a file using the Record block or how signals are displayed in the Record block. You can also still access complete signal data for signals connected to the Record block in the Simulation Data Inspector.

Component-Based Modeling

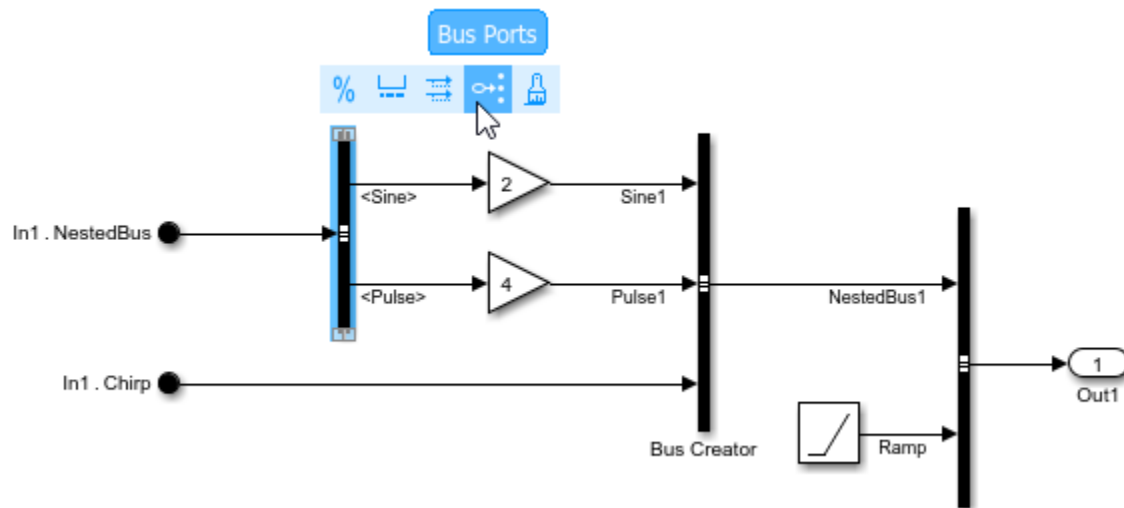
'startup' Variant Activation Time: Change active choice before simulation or in the generated code during model initialization

Starting in R2021b, use the `startup` variant activation time with Variant Subsystems, Variant Source, and Variant Sink blocks. The `startup` variant activation time allows you to change the active choice of the variant block before simulation or during model initialization in the generated code. Using the `startup` variant activation time offers multiple advantages for rapid prototyping workflows in model-based design.

- Iterative simulations — Use the `startup` variant activation time with the fast restart workflow to perform iterative simulations without recompiling a model or terminating the simulation each time. You compile a model only once. You can then change the active variant choice and simulate the model again without spending time on recompiling with different variant choice. Using the `startup` variant activation time with the fast restart workflow also makes Simulink Test™ iterative. You can create parameter sets with different variant configurations in fast restart mode to test your systems without needing to recompile the model. Using `startup` variant activation time with accelerator and rapid accelerator simulation targets enhances the speed of the simulation. For more information, see [Simulate a Variant Subsystem in Startup with Fast Restart Enabled](#), [Verify and Validate Variant Models with Startup Activation Time](#), and [Simulate Variant Blocks in Accelerator and Rapid Accelerator Modes with Startup Activation Time](#)
- Code generation — Use the `startup` variant activation time to generate code that contains all the variant choices and regular `if` statements. Code generation with all variants is supported with Embedded Coder and Simulink Coder. For more information, see [Generate Code from Variant Blocks with Startup Activation Time](#).
- Simulink Compiler™ — Simulink Compiler workflows are supported with `startup` variant activation time. You can create a standalone executable outside of Simulink with different variant configurations and simulate the standalone executable by changing the active variant via an input to the executable. For more information see [Create Standalone Executables with Variants using Simulink Compiler](#).
- Protected models — Use `startup` variant activation time to protect models that contain Variant Blocks with all variants. Without the `startup` variant activation time, the protected model contains only the active variant. For more information, see [Simulate a Protected Model with Code Compile or Startup Activation Time](#).
- Variant parameters — The `startup` variant activation time is supported with variant parameters.

Convert input and output ports to bus ports at root-level of model

You can now convert a model interface that uses Inport, Bus Selector, Bus Creator, and Outport blocks to use In Bus Element and Out Bus Element blocks.



The blocks you convert can specify `Simulink.Bus` object data types.

For more information, see [Simplify Subsystem and Model Interfaces with Buses](#).

Support for code generation workflows in Variant Reducer

Starting in R2021b, Variant Reducer supports reducing a model based on compilation for code generation. If the model contains variant blocks with the **Variant control mode** parameter set to `sim codegen switching`, you can choose to retain the code generation branch in the reduced model. Previously, Variant Reducer would always reduce the simulation branch of the top-level model.

From the Variant Reducer dialog box, you can select the compilation mode as `sim` or `codegen` using the **Compile mode** list. When reducing programmatically, you can specify the mode using the `'CompileMode'` name-value argument.

Use new field in CompiledVariantInfo parameter to identify execution status of a block

If the variant activation time for a variant block is set to `startup`, Simulink analyzes all the variant choices during compilation and sets the active choice at the start of simulation. You can use the `IsInStartup` field in the **CompiledVariantInfo** block parameter to identify if a block is active in simulation or can become active during simulation due to the `startup` variant activation time. After compilation, `IsInStartup` is set to `on` if:

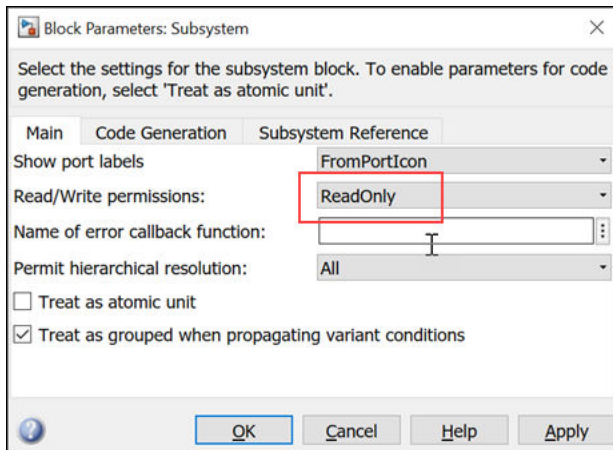
- The block is active in simulation.
- The block is currently inactive but can become active during simulation due to propagation of `startup` variant activation time.

For an example, see [Identify Variant Activeness of a Block After Model Compilation](#).

Prevent accidental edits to subsystem reference using the read-only property

You can now set the block parameters of a subsystem reference to read-only to prevent accidental edits. To set the block parameters to read-only, right-click on the subsystem. Select **Block**

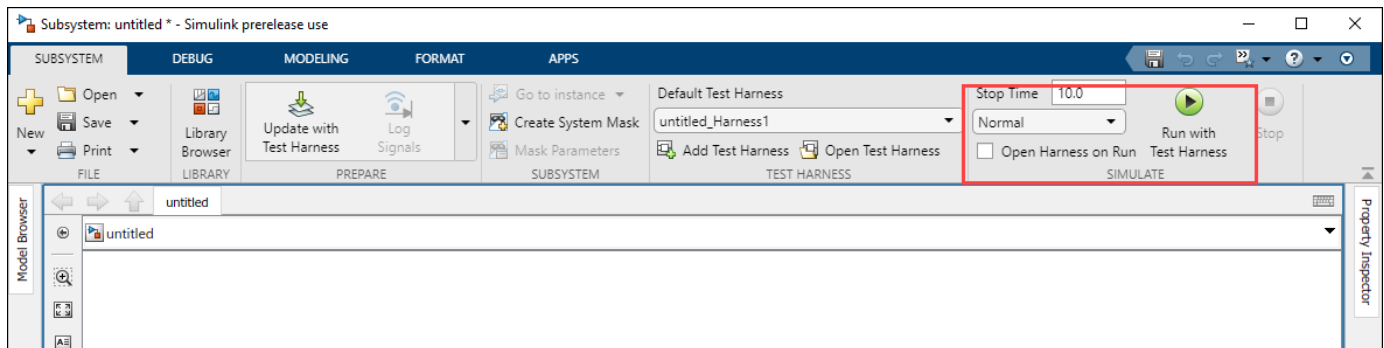
Parameters and under **Read/Write permissions**, select **ReadOnly**. You can also enter `set_param()` at the MATLAB command prompt to set the block parameters as read-only.



Open test harness automatically while running it on a subsystem

Starting in 2021b, you can choose whether to open a test harness automatically while it is running on a subsystem.

- To automatically open the test harness while running it on the subsystem, click **Open Harness on Run** checkbox.
- To run the test harness on the subsystem without opening the test harness, click **Run with Test Harness**.



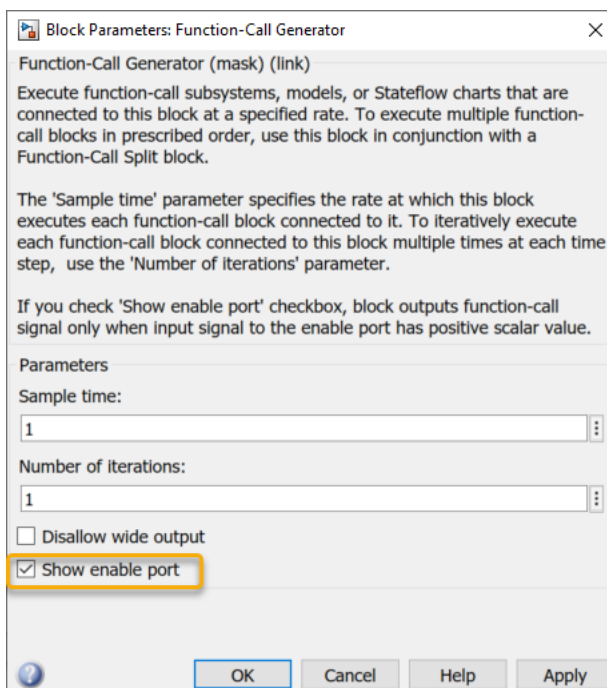
Schedule Editor: Use a global scoped event to combine multiple instances of an event

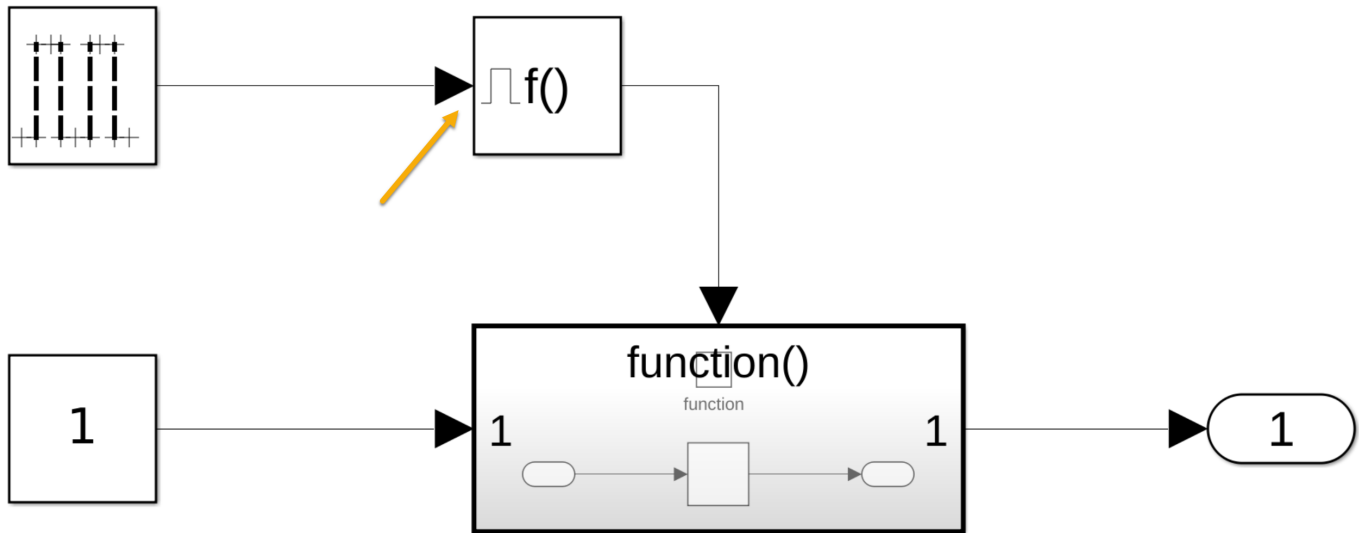
In R2021b, you can set a global scope of events in Schedule Editor. With a global scoped event, you can control how a particular event in a model is used by other models in the hierarchy. Use the global scope to combine multiple instances of the same event. These multiple instances of the events can be triggered by the same Stateflow chart to unify scheduling.

Function-Call Subsystem: Call periodic function-call subsystem intermittently using Function-Call Generator block

A function-call subsystem with the **Sample time type** of its Trigger block set to **Periodic** can now be called conditionally and does not need to be called at every time step. This feature allows you to simulate turning periodic tasks on and off intermittently. Previously, making aperiodic function calls to a periodic function-call subsystem resulted in an error that would stop a simulation. See [Using Function-Call Subsystems](#).

You can now configure a Function-Call Generator block to output function-call signals intermittently rather than at every time step. Selecting the new **Show enable port** parameter adds a scalar input port to the block and causes the block to output a function-call signal only when the value at the input port is a positive number.





Guard header files in generated code for S-Function blocks in variant models

If a model has S-Function blocks that are connected to variant blocks, Simulink guards the header files for each S-Function block with the corresponding variant condition in the generated code. See [Guarding S-Function Block Headers \(Embedded Coder\)](#).

Allow message root-level In Bus Element block to provide receive queue properties

Use an In Bus Element block to insert a message queue to the top model, in an external environment, or in middleware and specify queue attributes such as capacity, message sorting policy, or message overwriting policy. If you use multiple In Bus Element blocks, you can insert a message queue for each message element received by an In Bus Element block.

Variant Parameters: Enhanced simulation speed, improved code layout, and option to select active choice while running executable

Enhanced simulation and test speed with fast restart

Using fast restart, you can perform iterative simulations on variant parameters without compiling or terminating the simulation of the model each time. You compile a model once, and you can then change the values of variant parameters and simulate the model again without recompiling the model. The fast restart workflow also enables iterative workflows in Simulink Test.

You can simulate a model in fast restart only if you specify the activation time of the variant parameters as `startup`.

Consider the `slexVariantParametersCC` model. Change the activation time of the variant parameter to `startup` and simulate the model in fast restart mode. The iterative simulation is about 4 times faster in fast restart mode.

The approximate execution times on a Windows 10, Intel® Core™ i3-7020U CPU @ 2.30GHz test system are:

Mode	Iterative Simulation 1	Iterative Simulation 2	Iterative Simulation 3	Iterative Simulation 4
Without fast restart	0.2325	0.1414	0.1600	0.1415
With fast restart	0.1309	0.0404	0.0443	0.0392

For more information, see [How Fast Restart Improves Iterative Simulations](#).

Improved code layout

In the generated code, the variable definitions of all the variant parameters are grouped based on the associated variant condition expressions. When grouping, all the variable definitions that use the same variant condition expression are placed inside the same preprocessor `if` conditionals. This layout improves the readability and conciseness of the code.

This table compares code generated in R2021a to the code generated in R2021b for the `slexVariantParametersCC` model. In the model, variant parameters use variant condition expressions `V==1` and `V==2` to determine the active choice. Instead of three `if` conditionals for the definitions, the definitions are grouped together in a single `if` conditional statement.

MATLAB Code	R2021a Generated Code	R2021b Generated Code
<pre>% Variant control variable VCtrl = Simulink.VariantControl('ActivationTime', 'code compilation'); % Default storage classes MAX_LIFT = Simulink.VariantVariable('Choices', {'VCtrl == 1', 20}); SLIDER_POS = Simulink.VariantVariable('Choices', {'VCtrl == 2', 0.5}); % non default storage class Pspec = Simulink.Parameter(); Pspec.CoderInfo.StorageClass = 'ExportedGlobal'; T1Break = Simulink.VariantVariable('Specification', 'Pspec'); T1Break = addChoice(T1Break, {'VCtrl == 1', [-1, -0.99, 1], ... 'VCtrl == 2', 2*[-1, -0.99, 0.99]}); T2Break = Simulink.VariantVariable('Specification', 'Pspec'); T2Break = addChoice(T2Break, {'VCtrl == 1', -2:10:10, ... 'VCtrl == 2', -20:20:20}); T2Data = Simulink.VariantVariable('Specification', 'Pspec'); T2Data = addChoice(T2Data, {'VCtrl == 1', [4 5 6;16 19 20], ... 'VCtrl == 2', 2*[4 5 6;16 19 20]});</pre>	<pre>#if VCtrl == 1 /* Variable: T1Break Referenced by: '<Root>/1D Lookup' */ real_T T1Break[11] = { -5.0, -4.0, ..., 5.0 }; #elif VCtrl == 2 /* Variable: T1Break Referenced by: '<Root>/1D Lookup' */ real_T T1Break[11] = { -10.0, 0.0, 10.0 }; #endif Variable('Choices', {'VCtrl == 1', 20, ... 'VCtrl == 2', 0.5}); #if VCtrl == 1 /* Variable: T1Data Referenced by: '<Root>/1D Lookup' */ real_T T1Data[11] = { -1.0, -0.99, ..., 0.99 }; #else /* Variable: T1Data Referenced by: '<Root>/1D Lookup' */ real_T T1Data[11] = { -10.0, -8.0, ..., -2.0 }; #endif Variable('Specification', 'Pspec'); T1Break = addChoice(T1Break, {'VCtrl == 1', [-1, -0.99, 1], ... 'VCtrl == 2', 2*[-1, -0.99, 0.99]}); real_T T2Break[3] = { -10.0, 0.0, 10.0 }; #endif Variable('Specification', 'Pspec'); T2Break = addChoice(T2Break, {'VCtrl == 1', -2:10:10, ... 'VCtrl == 2', -20:20:20}); /* Variable: T2Break Referenced by: '<Root>/2D Lookup' */ real_T T2Break[3] = { -20.0, 0.0, 20.0 }; #endif Variable('Specification', 'Pspec'); T2Data = addChoice(T2Data, {'VCtrl == 1', [4 5 6;16 19 20], ... 'VCtrl == 2', 2*[4 5 6;16 19 20]}); /* Variable: T2Data Referenced by: '<Root>/2D Lookup' */ real_T T2Data[9] = { 4.0, 16.0, 5.0, ... }; #elif VCtrl == 2 /* Variable: T2Data Referenced by: '<Root>/2D Lookup' */ real_T T2Data[9] = { 8.0, 32.0, 10.0, ... }; #endif</pre>	<pre>#if VCtrl == 1 /* Variable: T1Break Referenced by: '<Root>/1D Lookup' */ real_T T1Break[11] = { -5.0, -4.0, ..., 5.0 }; #elif VCtrl == 2 /* Variable: T1Break Referenced by: '<Root>/1D Lookup' */ real_T T1Break[11] = { -10.0, 0.0, 10.0 }; #endif Variable('Choices', {'VCtrl == 1', 20, ... 'VCtrl == 2', 0.5}); #if VCtrl == 1 /* Variable: T1Data Referenced by: '<Root>/1D Lookup' */ real_T T1Data[11] = { -1.0, -0.99, ..., 0.99 }; #else /* Variable: T1Data Referenced by: '<Root>/1D Lookup' */ real_T T1Data[11] = { -10.0, -8.0, ..., -2.0 }; #endif Variable('Specification', 'Pspec'); T1Break = addChoice(T1Break, {'VCtrl == 1', [-1, -0.99, 1], ... 'VCtrl == 2', 2*[-1, -0.99, 0.99]}); real_T T2Break[3] = { -10.0, 0.0, 10.0 }; #endif Variable('Specification', 'Pspec'); T2Break = addChoice(T2Break, {'VCtrl == 1', -2:10:10, ... 'VCtrl == 2', -20:20:20}); /* Variable: T2Break Referenced by: '<Root>/2D Lookup' */ real_T T2Break[3] = { -20.0, 0.0, 20.0 }; #endif Variable('Specification', 'Pspec'); T2Data = addChoice(T2Data, {'VCtrl == 1', [4 5 6;16 19 20], ... 'VCtrl == 2', 2*[4 5 6;16 19 20]}); /* Variable: T2Data Referenced by: '<Root>/2D Lookup' */ real_T T2Data[9] = { 4.0, 16.0, 5.0, ... }; #elif VCtrl == 2 /* Variable: T2Data Referenced by: '<Root>/2D Lookup' */ real_T T2Data[9] = { 8.0, 32.0, 10.0, ... }; #endif</pre>

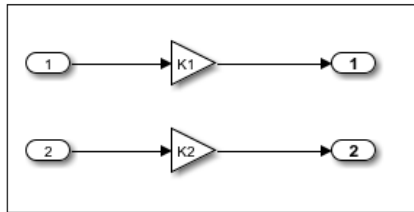
Select active choice while running executable

Prior to R2021b, the executable could contain only active choices. Switching between the choices while running the executable was not supported. Now, you can build an executable from generated code that contains all choices of variant parameters in regular `if` conditions. You build an executable once, and you can then choose the active choice of variant parameters by changing the value of the variant control variable before running the executable. You do not need to rebuild the executable for different variant parameter choices.

To build such an executable, you must set the activation time of the variant parameters in the model to startup.

Consider the `slexVariantParameters` model. In this model, change the activation time of the variant parameters to `startup` and then generate a C code using Simulink Coder or Embedded Coder. The generated code contains all the choices of the variant parameters in regular `if` conditions. The executable that you build from the generated code also contains all the choices of the variant parameters, thus enabling you to switch between the choices before running the executable.

For more information, see [Run Executable for Different Variant Parameter Values Without Recompiling Code](#).



```
void slexVariantParameters_initialize(void)
{
    rt_InitInfAndNaN(sizeof(real_T));
    (void) memset((void *)slexVariantParameters_M, 0,
                 sizeof(RT_MODEL_slexVariantParameter_T));
    rtmSetTFinal(slexVariantParameters_M, 10.0);
    slexVariantParameters_M->Timing.stepSize0 = 0.2;
    { ... }
    { ... }
    (void)memset(&slexVariantParameters_U, 0, sizeof(ExtU_slexVariantParameters_T));
    (void)memset(&slexVariantParameters_Y, 0, sizeof(ExtY_slexVariantParameters_T));
    if (V == 1) {
        slexVariantParameters_P.K1 = 3.5;
        slexVariantParameters_P.K2 = 4.5;
    } else if (V == 2) {
        slexVariantParameters_P.K1 = 8.5;
        slexVariantParameters_P.K2 = 9.5;
    }
}
```

Project and File Management

Model Comparison: Compare Simulink models in MATLAB Online

Starting in R2021b, you can compare Simulink model files in MATLAB Online using the `visdiff` function. To compare models, in the Command Window, type:





```
visdiff('filename1.slx','filename2.slx');
```

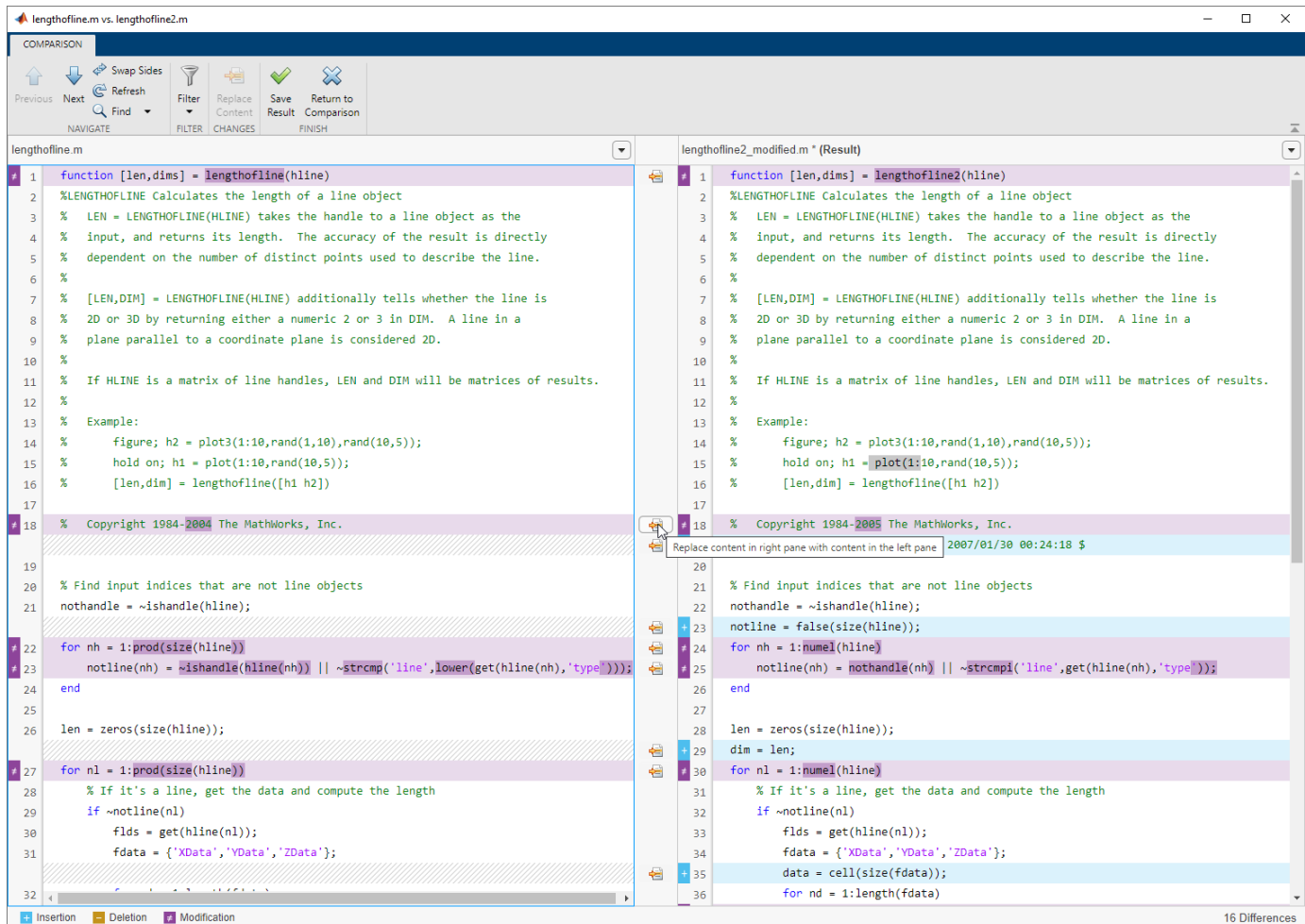
Projects: Collaborate using Projects in MATLAB Online

Starting in R2021b, MATLAB Online provides support for basic Projects workflows:

- Create an empty project and add files and folders.
- Clone a project from Git.
- Explore your project and run a dependency analysis.
- Create a project and manage your project files programmatically.

Text Comparison: Compare and merge text files with improved usability, appearance, and syntax highlighting

In R2021b, the comparison tool uses MATLAB Editor syntax highlighting. It is now easier to understand text changes and merge choices. Changes are highlighted with strong colors. Modified lines are highlighted and flagged with the comparison icons , , or . Merging line by line is straightforward, and merge choices are flagged with the **Replace Content** button .



Version History

Starting in R2021b, you no longer can save the comparison report as HTML or specify whether to show only the differences or the entire files.

OPC format for MDL files

Starting in R2021b, MATLAB provides an Open Packaging Conventions MDL format for model files. Newly created MDL files are written in plain text and contain the same information as SLX files. For MDL model files created before R2021b, open and resave the model files to use the new format. See [Open a Model That Has Different Character Encoding](#). If you use source control, resaving your files preserves their source control history.

The OPC MDL format lifts limitations in features and workflows that only supported SLX files:

- Simulink Test Harnesses
- Mask images saved inside model files
- AUTOSAR XML parts

- Simulink Requirements
- Viewmarks saved inside model files
- Library Browser repositories saved inside model files

Model Advisor checks for upgrading MDL files and checking for foreign characters are no longer needed.

Project Preferences: Detect project-wide references when renaming Simulink buses and bus elements

You can now disable automatic updates for projects when renaming Simulink buses or bus elements while keeping automatic updates enabled for other elements and files. For more details, see [Automatic Updates When Renaming, Deleting, or Removing Files](#).

Source Control: Work with files under Git source control in MATLAB Online

Starting in R2021b, MATLAB Online provides support for basic Git workflows:

- Cloning a remote Git repository.
- Committing files to Git.
- Pulling, pushing and fetching files with Git.

Functionality being removed or changed

Simulink will no longer support setting default block diagram parameters

Warns

Setting parameter "{0}" of the default block diagram will not be supported in a future release. New models inherit parameter values from templates. In R2021b, scripts that set default block diagram parameters continue to work but throw a warning.

Data Management

Specify signal properties based on application-specific value types such as wind velocity

Use the new `Simulink.ValueType` objects to specify an application-specific set of properties that corresponds to a type of value, such as wind velocity. With `ValueType` objects, you specify the properties once and reuse that specification for each signal that represents the same type of value.

`ValueType` objects can specify:

- Data type
- Minimum value
- Maximum value
- Physical unit
- Complexity
- Dimensions
- Whether signals are variable-sized or fixed-sized
- Description

You can use `ValueType` objects to enforce consistency between connected blocks at an interface.

For more information, see `Simulink.ValueType`.

Signal Editor user interface now accepts simple MATLAB expressions in tabular area

The Signal Editor user interface now accepts simple MATLAB expressions in tabular area. Prior to R2021b, entering any MATLAB expression in the tabular area had no effect.

Signal Editor adds timeseries data by default

The Signal Editor user interface now adds `timeseries` data by default when you add a new signal, for example, when you create freehand signals or create signals using MATLAB expressions and variables. Prior to R2021b, the Signal Editor user interface created these signals as logged `timeseries` (`Simulink.SimulationData.Signal`) data.

Correspondingly, the **Default Properties for Insertion** dialog box from **Defaults** on the Signal Editor toolstrip now lets you select a default data type for created signals:

- Timeseries (default)
- Timetable
- Logged timeseries
- Logged timetable

Prior to R2021b, setting a signal as `timeseries` or `logged` was not supported.

Simulink.io.FileType has new methods

`Simulink.io.FileType` has new methods:

- `Simulink.io.FileType.isInput` — Check if an input is appropriate for simulation.
- `Simulink.io.FileType.isSimulinkParameter` — Check if input is a supported simulation parameter, which means it is of type `Simulink.Simulation.ModelParameter`, `Simulink.Simulation.BlockParameter`, `Simulink.Simulation.Variable`, or `Simulink.op.ModelOperatingPoint`.

For an example, see `open('Simulink.io.BlockParameterSpreadsheet')`.

Prior to R2021b, `Simulink.io.FileType` did not support simulation parameters.

New Simulink.io.ModelWorkspace class

The `Simulink.io.ModelWorkspace` class reads data in a format used by model workspaces. Use model workspace file types to work with variables and parameters in the model workspace of the specified model.

Configure complexity for Simulink.Parameter objects

Previously, the `Complexity` property of a `Simulink.Parameter` object was a read-only property derived from the parameter value that you specified in the `Value` property. In R2021b, you can configure `Complexity` for a `Simulink.Parameter` object as either `real` (default) or `complex`.

When `Value` is numeric, Simulink still derives the complexity of a `Simulink.Parameter` object from that value. In this case, changing the complexity results in an error. Simulink does not consider complexity when `Value` is a structure or a mathematical expression. For more information, see `Simulink.Parameter`.

For more information on how to configure the `Complexity` property to define model arguments, see “Define complex model arguments without specifying a default value” on page 2-32.

Define complex model arguments without specifying a default value

Previously, you could configure a `Simulink.Parameter` object as a model argument without providing a default value when you specified `Data Type` and `Dimensions` for the argument. In R2021b, these model arguments can also be complex. To configure a parameter as a complex model argument with the default value left empty (`Value` is `''`), select `complex` as the value of `Complexity` for the object. Simulink uses the specified `Complexity` along with the `Data Type`, `Dimensions`, `Min`, and `Max` to synthesize a value.

For more information, see “Configure complexity for Simulink.Parameter objects” on page 2-32 and `Parameterize Instances of a Reusable Referenced Model`.

Import struct and enum types without typedef from custom C code

When you integrate custom C code into Simulink, you can now use the `Simulink.importExternalCTypes` function or the Simulink Code Importer to import `struct` and `enum` types that are defined in your custom code header files without a `typedef` statement, and

simulate and generate code from models that use the imported types. For a `struct`, the name of the imported `Simulink.Bus` object matches the tag name of the type in the header file. For an enum, the imported `Simulink.IntEnumType` or `Simulink.data.dictionary.EnumTypeDefinition` object matches the tag name of the type in the header file.

For example, consider a header file that contains the following definitions:

```
struct point {
    double x, y;
};

enum fault {
    PWR_LOSS = 0,
    OVERSPD,
    PRESS_LOW,
};
```

After importing the types from your custom code, the imported `Simulink.Bus` object has the name `point`, and the imported `Simulink.IntEnumType` or `Simulink.data.dictionary.EnumTypeDefinition` object has the name `fault`.

Block Enhancements

New Matrix Operations library

The new **Simulink** > **Matrix Operations** sublibrary contains blocks that implement common matrix algorithms:

- Cross Product — Cross product of two vectors
- Hermitian Transpose — Computer Hermitian transpose of matrix
- Identity Matrix — Create identity matrix
- IsSymmetric — Check if matrix is symmetric matrix or skew-symmetric
- IsTriangular — Upper or lower triangular matrix
- Matrix Square — Compute square of input matrix
- Transpose — Compute transpose of matrix

New button blocks in the Customizable Blocks library

The Customizable Blocks library inside the Dashboard library now contains a Callback Button block and a Push Button block. When you use a block from the Customizable Blocks library, you can use the default design or customize the appearance of the block.

When you design a Callback Button or Push Button block, you can:

- Add an image to define the appearance of the block in that state.
- Specify text for the button label and configure its color and position within the block.
- Add an icon and specify the position of the icon relative to the label.

New switch blocks in the Customizable Blocks library

The Customizable Blocks library inside the Dashboard library has several new switch blocks:

- Rocker Switch
- Slider Switch
- Toggle Switch

The switch blocks all provide identical customization options, with each block providing a different preconfigured design. You can use the blocks as they appear in the library or design their appearance to look like a real switch in your system.

When you design a switch block using the Rocker Switch, Slider Switch, or Toggle Switch block, you can:

- Add any number of states.
- Position and resize the click area, image, and label for each state.
- Add a background image or foreground image.

New Rotary Switch block in the Customizable Blocks library

The Customizable Blocks library inside the Dashboard library now includes a Rotary Switch block. When you use a block from the Customizable Blocks library, you can use the default design or customize the appearance of the block.

Use the Rotary Switch block to design a control that you can use to switch between any number of discrete states that are evenly distributed along a circular arc. When you design a Rotary Switch, you can:

- Position the start and end points for the arc.
- Add any number of states.
- Add your own handle image, background image, and foreground image.

New Lamp block in the Customizable Blocks library

The Customizable Blocks library inside the Dashboard library now has a Lamp block with customization options that enable you to design a lamp that looks like an indicator in a real system.

When you add a Lamp block to your model, the block is preconfigured with a default design. You can use the block with the default design or customize the appearance of the block. When you design a custom Lamp block, you can:

- Select a shape for the region that changes color to reflect the value of the connected signal.
- Select an icon from a built-in selection that includes wireless icons and automotive indicator lamps or upload your own custom icon.
- Specify how to combine the shape and the icon.



Simple



Intersect



Subtract

- Upload a background image or specify a background color to provide a background for the shape and icon.
- Upload a foreground image that overlays the rest of the lamp design.

Upgrade Signal Builder blocks to Signal Editor blocks with Upgrade Advisor

To upgrade Signal Builder blocks to Signal Editor blocks with Upgrade Advisor, use the Check model for Signal Builder blocks check.

Upgrade Advisor: Check output dimensions of MATLAB Function blocks

This check lists all the MATLAB Function blocks that have the **Interpret output column vectors as one-dimensional data** parameter selected. You can change the output dimensions of the listed blocks that do not produce one-dimensional output data, to a column vector of size N-by-1 by clicking the 'Fix it' button.

For more information, see Interpret output column vectors as one-dimensional data.

Lookup table viewer updates

The Lookup table viewer, shared by multiple Simulink interfaces (including the `Simulink.LookupTable` property dialog box), has been updated.

- You can now make row and column additions and deletions from anywhere in the tabular area. In previous releases, you could add or remove rows and columns only after the current last row or column of the table.
- Heatmap option available using a `parula` colormap. To enable the heatmap for the table data, right-click any cell and select **Heatmap On**.
- Multiple undo and redo actions are possible across slices before you need to click **Apply**. In previous releases, you could undo and redo within only one slice.
- Tabular entries feature improved display precision. To see the actual value, double-click the tabular entry.

3	3.50e+3	8.73	8.72840998619654052	8.
4	5.26e+3	10.49	10.49	6.

In previous releases, the displays were less precise even though the values were the same.

3	3504	8.7284099	8.7284099	8.728
4	5260	10.49	10.49	6

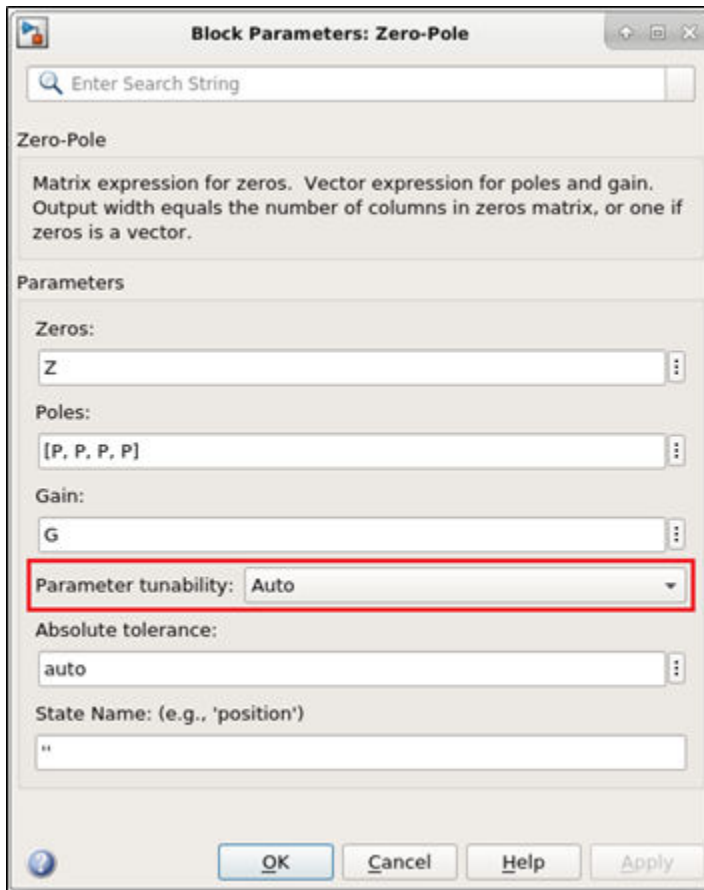
- Half data type is now supported.

Parameter Tunability Enhancements for Continuous Blocks

You can now consistently tune several parameters of the Zero-Pole block in the Simulink Compiler.

Set the **Parameter Tunability** setting in the Zero-Pole block to **Unconstrained** to allow full tunability of the parameter matrices or coefficients, respectively, in all simulation modes and in the Simulink Compiler.

When set to **Optimized**, Simulink optimizes the tunability of the block parameters for performance.



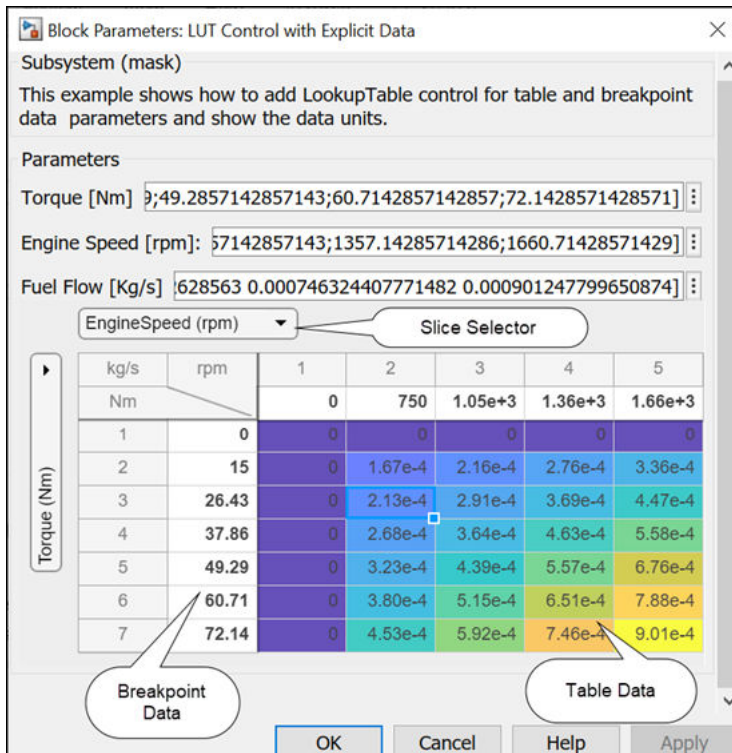
Signal Builder Import File dialog box updated

The Signal Builder **Import File** dialog box has been updated. Notably, the Status History pane has been removed.

For further updates, see Importing Signal Group Sets.

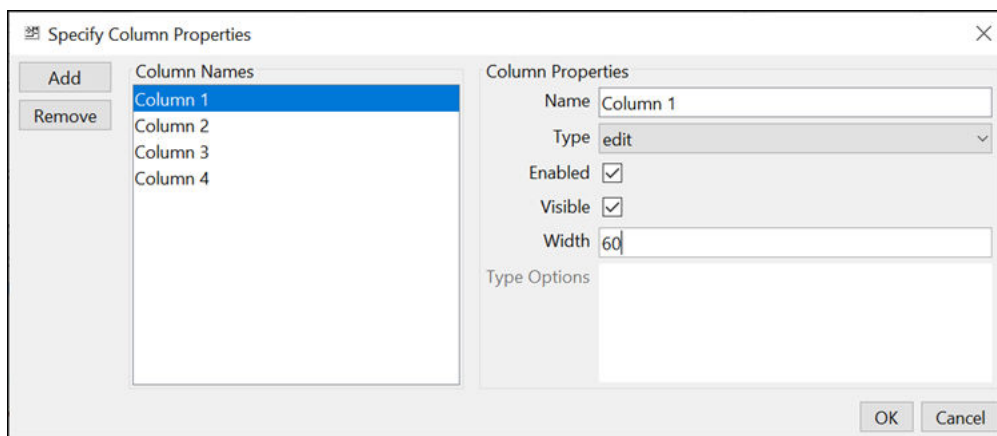
Visualize and edit n-dimensional table data in mask dialog control using lookup table editor

You can now seamlessly view and edit n-dimensional table data and breakpoint parameters in the mask dialog control using Lookup table editor. Use the widget in the **Parameters & Dialog** tab to specify the table parameters and breakpoints. For more information, see Visualize and Edit N-Dimensional Data Using Mask Lookup Table Control.



Control initial size of column in a custom table

Use the `Width` property to specify initial column width for a custom table. For more information, see [Customize Tables for Masked Blocks](#)



Specify Row and AlignPrompts for containers in System objects blocks

Starting in R2021b, for System objects blocks, you can specify the row and align prompts for containers using the `Row` and `AlignPrompts` properties in `Matlab.system.display.sectiongroup` and `Matlab.system.display.section` classes.

- **Row:** Specify the row in which the containers need to be placed (current/new). You can specify the row using the enum class `matlab.system.display.internal.Row`.

- **AlignPrompts:** Specify a boolean value to control align prompts within the containers.

Specify descriptions with In Bus Element and Out Bus Element blocks

You can now specify descriptions for the bus and bus elements associated with an In Bus Element or Out Bus Element block. Click the pencil button or the nondefault attribute summary next to the name of a bus or bus element. The dialog box that opens now contains a **Description** parameter.

When the block specifies a `Simulink.Bus` object data type, the descriptions from the `Simulink.Bus` and `Simulink.BusElement` objects appear in the dialog box.

String support for MATLAB System Block

In R2021b, the MATLAB System block supports the string data type for input and output. The string data type is supported for all simulation modes, including rapid accelerator. The string data type is not supported for code generation.

Push Button block enhancements

You can configure the behavior of the Push Button block to latch or momentarily apply the **On Value** when you click the block. You can also add an icon to display on the block alongside or instead of the button text.

You can choose an icon for the block from a predefined set or upload your own custom icon using the block dialog box or the **Button** tab of the toolstrip. You can also specify an on color and an off color so the icon color changes with the state of the block and specify the alignment of the icon relative to the button text.



FMU Import updates

The FMU import block can now be exported to co-simulation standalone FMU and protected model targets.

Functionality being removed or changed

XY Graph block replaced with alternative configuration of the Record block

Behavior change

The XY Graph block is replaced with a Record block that is preconfigured to display two input signals on an XY plot. Using the updated XY Graph block, you can:

- View data on the XY plot during simulation.
- Record the XY plot data to the workspace, a file, or the workspace and a file.

- Zoom, pan, fit to view, and adjust the axis limits for the XY plot without having to simulate again.
- Analyze the XY trace by using cursors and by adding a trend line.
- Add additional ports and subplots to visualize additional simulation data alongside the XY plot on several visualization types, including time plots, maps, or additional XY plots.
- Visualize multiple traces on the XY plot.

Upon simulation, the prior version of the XY Graph block opened a new window to display the connected signal data during simulation. To view data on the updated XY Graph block, double-click the block in the model. To view the XY plot in a new window, right-click the block and select **Open in New Window**.

XY Graph blocks in existing models are automatically replaced with the new XY Graph block. XY Graph viewers are also replaced with the new XY Graph block. Goto and From blocks connect signals to the XY Graph block that replaces the XY Graph viewer.

Starting in R2021b, signals connected to an XY Graph block always stream to the Simulation Data Inspector.

For more information, see Record, XY Graph.

Editor specification with docblock does not persist between previous releases and R2021b *Behavior change*

If you used docblock to specify the editors that a DocBlock block uses for HTML, Rich Text Format (RTF), or ASCII documents before R2021b, you must specify the editors again in R2021b. Otherwise, a DocBlock block uses the default editor for the document type.

Environment Controller block has been removed

Warns

The Environment Controller block has been removed from the Signal Routing library. Existing models that use this block continue to work with a warning. Use the Variant Source block with the **Variant control mode** parameter set to `sim_codegen_switching` instead of the Environment Controller block. You can use Model Advisor to make this replacement automatically.

Variant blocks offer these advantages over the Environment Controller block:

- Variant Source block with the **Variant control mode** parameter set to `sim_codegen_switching` allows you to automatically switch between simulation and code generation workflows.
- Variant Source block supports different activation times for variant choices using the **Variant activation time** parameter.
- Variant Sink block enables branching on the output side (destination of a signal).
- Variant blocks highlight the path of the active variant choice during simulation and code generation.
- Variant Source block offers optimal code generation compared to the Environment Controller block. The Environment Controller block can retain blocks connected to the Sim port in the generated code.

PID Controller Simulink Blocks: ReferenceBlock parameter returns different path *Behavior change*

Starting in R2021b, for PID Controller Simulink blocks, the `get_param` function returns a different value for the `ReferenceBlock` parameter. The `ReferenceBlock` parameter gives the path of the library block to which a block links. Previous and new paths for each block are as follows.

Block	Before R2020b	R2020b and R2021a	R2021b
PID Controller	'simulink/ Continuous/PID Controller'	'pid_lib/PID Controller'	'slpidlib/PID Controller'
PID Controller (2DOF)	'simulink/ Continuous/PID Controller (2DOF)'	'pid_lib/PID Controller (2DOF)'	'slpidlib/PID Controller (2DOF)'

The Discrete PID Controller and Discrete PID Controller (2DOF) blocks link to the same path as the corresponding continuous blocks because the blocks are identical, except for the default value of the `Time domain` parameter.

This change does not affect any other functionality or workflows. You can still use the previous paths with the `set_param` function.

Connection to Hardware

Deploy customizable Gauge and Knob Dashboard blocks on Android device

The Simulink Support Package for Android Devices now supports deploying customizable Gauge and Knob Dashboard blocks on your Android device. You can customize the visual aspects of the Gauge and Knob Dashboard blocks in the Simulink model and obtain the what you see is what you get (WYSIWYG) visualization on your Android device as well as on web pages.

Generate code from Simulink model to run directly on Android devices using PIL simulation

The Simulink Support Package for Android Devices now supports generating code from a Simulink model to run directly on your Android device using processor-in-the-loop (PIL) simulation. The results of the PIL simulation are transferred to Simulink to verify the numerical equivalence of the simulation and the code generation results.

Read and write commands to TalkBack ID on ThingSpeak TalkBack for Android devices

The Simulink Support Package for Android Devices now includes the TalkBack Read and TalkBack Write blocks. You can now read and write commands to TalkBack ID on ThingSpeak™ TalkBack.

Send and receive data using Bluetooth low energy protocol on Arduino boards

The Simulink Support Package for Arduino Hardware now includes BLE Send and BLE Receive blocks. You can now send and receive data using the Bluetooth® low energy (BLE) protocol on Arduino hardware. Only the Arduino Nano 33 IoT board, the Arduino Nano 33 BLE Sense board, and the Arduino MKR Wifi 1010 board support this feature.

Create interactive display of Dashboard Push Button block on Arduino boards

The Simulink Support Package for Arduino Hardware now supports generating code and deploying the Dashboard Push Button block on the Arduino boards. You can use the Push Button block in the Simulink model to create an interactive display dashboard and deploy the block on the Arduino boards.

Deploy Input Capture and SD Card File Read blocks on Arduino Nano 33 BLE Sense boards

The Simulink Support Package for Arduino Hardware now supports the Input Capture and the SD Card File Read blocks on the Arduino Nano 33 BLE Sense board. You can now generate C code and deploy these blocks on the Arduino Nano 33 BLE Sense board.

Measure rotation of motor interfaced with rotary encoder on Arduino boards

The Simulink Support Package for Arduino Hardware now includes the Encoder block. You can now measure the incremental position and direction of a motor interfaced with a rotary encoder.

Read and write from ThingSpeak data channels simultaneously on Arduino boards

The Simulink Support Package for Arduino Hardware now supports simultaneous read and write operations on fields of the ThingSpeak data channels for the ThingSpeak Read and the ThingSpeak Write blocks, respectively.

Encode and decode data packets based on any messaging format for Arduino-based peripheral communication

The Simulink Support Package for Arduino Hardware now provides the Protocol Encoder and Protocol Decoder blocks to encode and decode data for Arduino-based peripheral communication based on any messaging format. You can use these blocks to encode input data to a `uint8` byte stream or to decode a `uint8` byte stream by specifying each packet field and its sequence as defined in the messaging format of the protocol definition. The blocks provide the option to specify the header, terminator, and the logic for checksum validation (either standard logic or custom logic) for the data. The Protocol Decoder block also supports decoding data from a comma-separated variable length packet.

SocketCAN support added for CAN Transmit and CAN Receive blocks on Raspberry Pi boards

The CAN Transmit and the CAN Receive blocks from the Simulink Support Package for Raspberry Pi Hardware now use SocketCAN, a device independent driver for implementing the CAN interface that supports all CAN shields. Previously, these blocks supported only MCP2515-based CAN shields.

Deploy customizable Gauge and Knob Dashboard blocks on Raspberry Pi boards

The Simulink Support Package for Raspberry Pi Hardware now supports deploying customizable Gauge and Knob Dashboard blocks on your Raspberry Pi hardware boards. You can customize the visual aspects of the Gauge and Knob Dashboard blocks in the Simulink model and obtain the what you see is what you get (WYSIWYG) visualization on a web browser you launch from the Raspberry Pi terminal.

Read and write commands to TalkBack ID on ThingSpeak TalkBack on Raspberry Pi boards

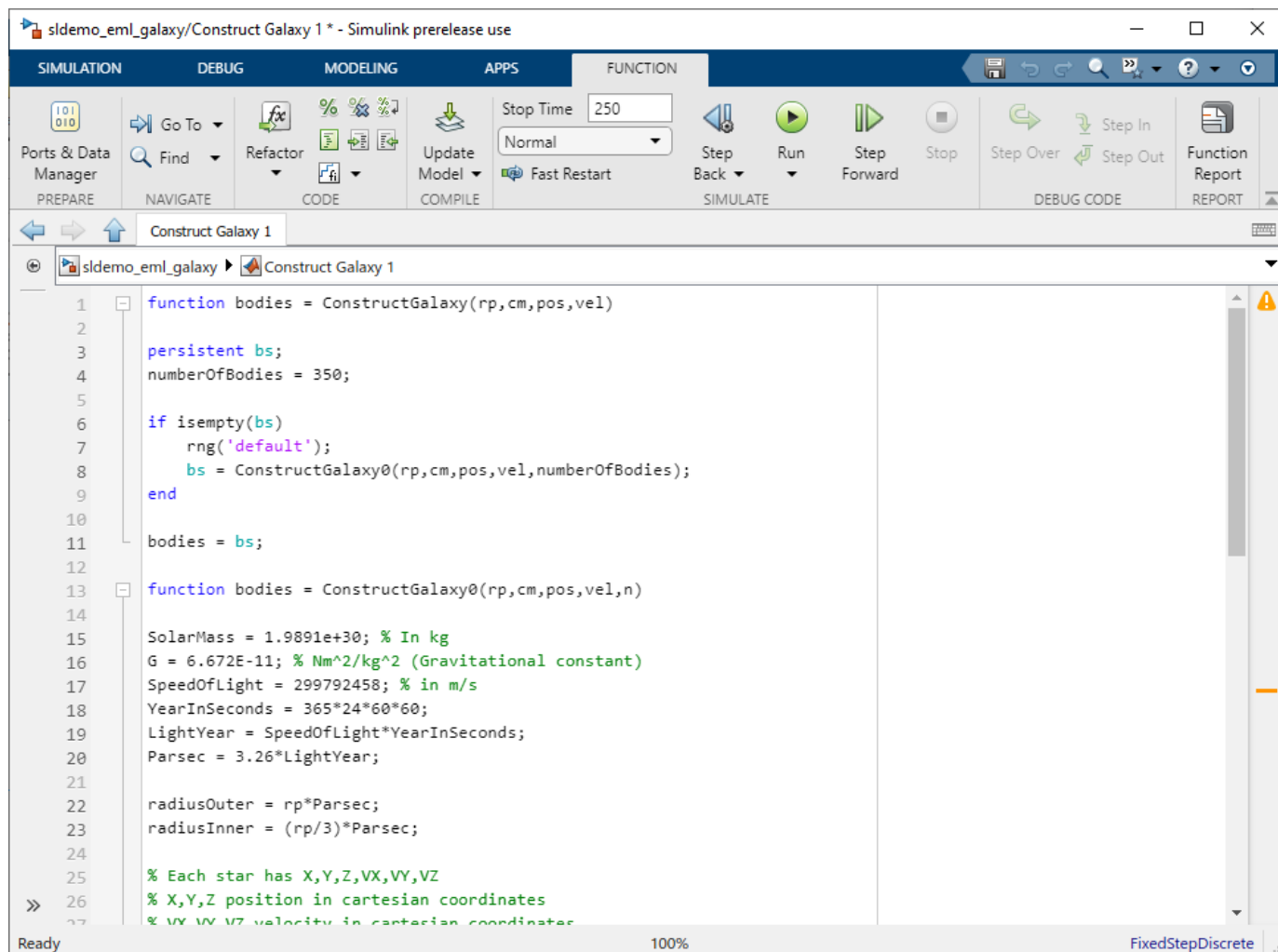
The Simulink Support Package for Raspberry Pi Hardware now includes the TalkBack Read and TalkBack Write blocks. You can now read and write commands to TalkBack ID on ThingSpeak TalkBack.


MATLAB Function Blocks

MATLAB Function Block Editor in Simulink window

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

- Easily navigate from the MATLAB function back to the parent model in the same window.
- Access and edit properties of the model while viewing the MATLAB function on the canvas.
- Run and step the simulation while debugging the MATLAB function in the MATLAB Function block debugger.
- Edit your function using the tools that were previously available in the **MATLAB Function Block Editor**.



To open the editor, in a Simulink model, double-click a MATLAB Function block. The **Function** tab opens and the canvas shows the MATLAB function. To navigate back to the model, use the navigation buttons  or click the name of the model in the hierarchy.

If you use Embedded Coder to generate code from your MATLAB Function block, you can view the generated code alongside the function by using the Code view. For more information, see “Code view for MATLAB Function block” (Embedded Coder).

Output column vectors as two-dimensional or one-dimensional data in MATLAB Function blocks

You can now output column vectors in MATLAB Function blocks as two-dimensional or one-dimensional data with the **Interpret output column vectors as one-dimensional data** property.

Version History

Before R2021b, MATLAB Function blocks always output column vectors as one-dimensional data. After R2021b, MATLAB Function blocks output column vectors as two-dimensional data by default. To maintain the original behavior of the block, select this property.

Expanded code generation for tables and timetables

In R2021b, code generation supports more capabilities and MATLAB toolbox functions when you use tables and timetables.

The supported functions for tables and timetables are:

- `innerjoin`
- `mergevars`
- `outerjoin`

For more information, see [Code Generation for Tables](#) and [Code Generation for Timetables](#).

Code generation for more MATLAB functions

- `digraph`
- `graph`
- `hess`
- `ode78`
- `ode89`

Code generation for more toolbox functions

In R2021b, you can generate code for many additional toolbox functions and objects. For a list of all functions and objects that are supported for code generation, see:

- [Functions and Objects Supported for C/C++ Code Generation \(Category List\)](#)

- [Functions and Objects Supported for C/C++ Code Generation \(Alphabetical List\)](#)

These are links to the release notes of some toolboxes that added code generation support in R2021b:

Computer Vision Toolbox

See [Generate C and C++ Code Using MATLAB Coder: Support for functions](#).

Image Processing Toolbox

See [C Code Generation: Generate code from five functions using MATLAB Coder](#).

Signal Processing Toolbox

See [C/C++ Code Generation Support: Code generation for filtering, spectral analysis, and vibration analysis](#).

Wavelet Toolbox

See [C/C++ Code Generation: Automatically generate code for wavelet functions](#).

MATLAB Function blocks support implicit expansion

MATLAB Function blocks now support implicit expansion for simulation and code generation. For more information, see [Compatible Array Sizes for Basic Operations](#).

Modeling Guidelines

Use modeling guidelines to verify compliance of your models

In R2021b, these high-integrity modeling guidelines are added:

Guideline	Equivalent check
hisl_0072: Usage of tunable parameters for referenced models	Check for parameter tunability ignored for referenced models (Simulink Check)
hisl_0073: Usage of bit-shift operations	Check usage of bit-shift operations (Simulink Check)
hisl_0074: Configuration Parameters > Diagnostics > Modeling issues related to variants	Check safety-related diagnostic settings for variants (Simulink Check)
himl_0012: Usage of MATLAB functions for code generation	Check MATLAB functions not supported for code generation (Simulink Check)
himl_0013: Limitation of built-in MATLAB Function complexity	Metrics for generated code complexity (Simulink Check)

This table identifies modeling guidelines that were modified in R2021a.

Modeling Guideline	Rationale
himl_0003: Complexity of user-defined MATLAB Functions	Updated the title.
hisl_0070: Placement of requirement links in a model	Updated information on System Composer.
hisl_0016: Usage of blocks that compute relational operators	Added conditions to prevent unexpected results.
hisl_0002: Usage of remainder and reciprocal operations	<ul style="list-style-type: none"> Updated the title, examples. The Guideline now supports Stateflow and MATLAB domains.
hisl_0003: Usage of square root operations	<ul style="list-style-type: none"> Updated the title, examples. The Guideline now supports Stateflow and MATLAB domains.
hisl_0004: Usage of natural logarithm and base 10 logarithm operations	<ul style="list-style-type: none"> Updated the title, examples. The Guideline now supports Stateflow and MATLAB domains.
hisl_0019: Usage of bitwise operations	The Guideline now supports Stateflow and MATLAB domains.

Version History

This table identifies changes to the modeling guidelines that may impact compatibility, including removed guidelines or conditions.

Modeling Guideline	Corresponding Model Advisor Check	Rationale
hisf_0064: Shift operations for Stateflow data to improve code compliance	Check usage of shift operations for Stateflow data	The new guideline hisl_0073: Usage of bit-shift operations covers MATLAB and Simulink domains also.
himl_0009: MATLAB code with equal / not equal relational operators	Check usage of equality operators in MATLAB Function blocks	Updated guideline hisl_0016: Usage of blocks that compute relational operators supports Simulink and Stateflow domains also.
hisf_0003: Usage of bitwise operations	Check usage of bitwise operations in Stateflow charts	Updated guideline hisl_0019: Usage of bitwise operations covers Simulink and MATLAB domains too.

S-Functions

S-Function Builder: Support for array of buses, nested array of buses, and single-word fixdt data types

In R2021b, the S-Function Builder block supports arrays of buses, nested arrays of buses, and single-word `fixdt` data types, such as input and output.

For more, see [Use Array or Nested Array of Buses with S-Function Builder to Create an S-Function](#)

R2021a

Version: 10.3

New Features

Bug Fixes

Version History

Simulink Editor

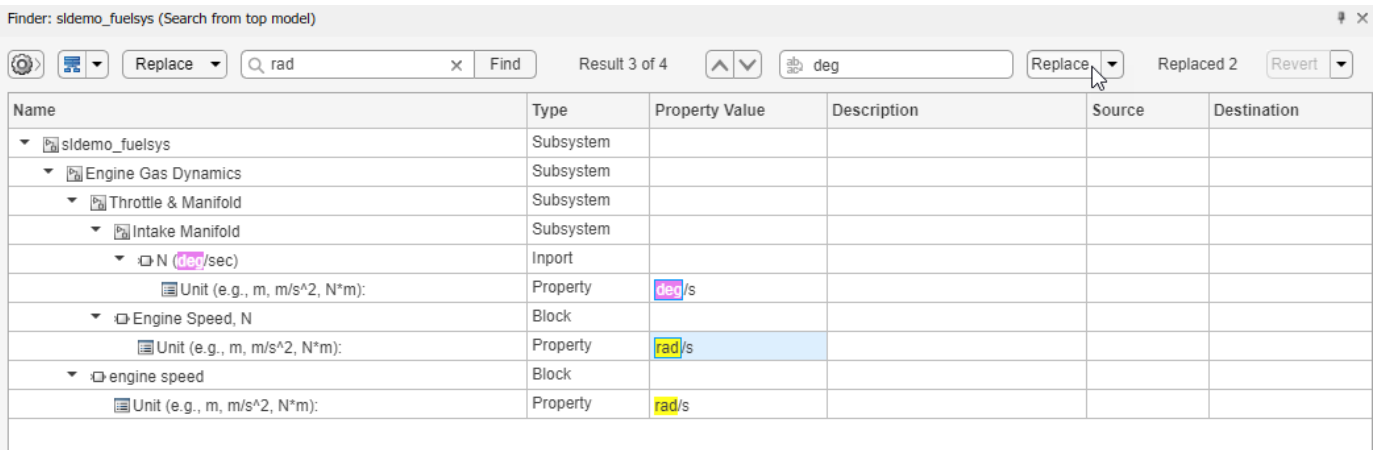
Schedule Editor: View connections between Stateflow charts and corresponding partitions with Schedule Connectors

In R2021a, Schedule Connectors allow you to view relationships between Stateflow charts calling scheduling events and the components driven by those events.

To view the connections, on the **Design** tab, under **Diagnostics** section, select **Information Overlays**. From the drop down, under **Blocks** section, select **Schedule Connectors**.

Find and replace text in Simulink

You can now replace text that you find in Simulink models, subsystems, and libraries.



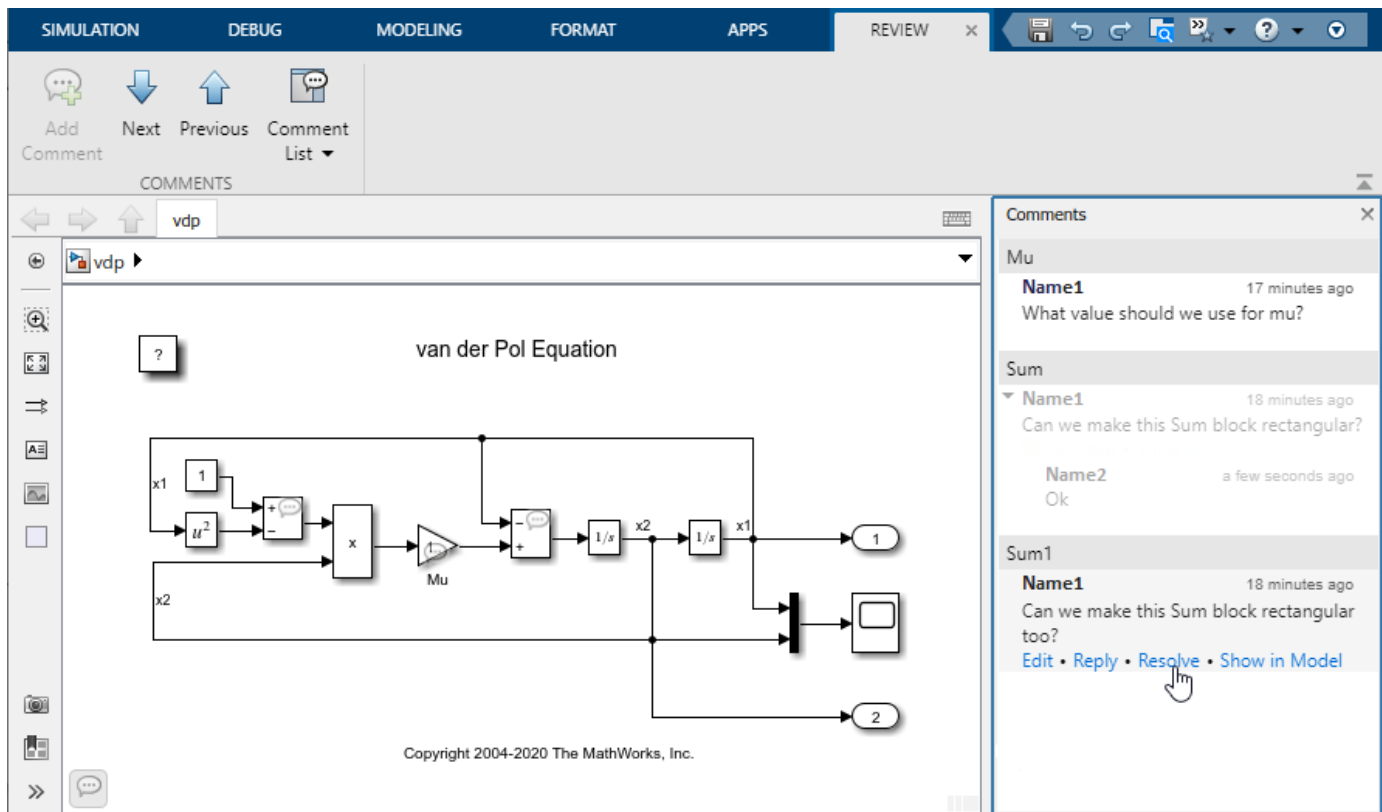
For search results to support replacement:

- You must have write access for the field.
- The field value must be the same as when you performed the search.
- The text formatting must be plain.

For more information and an example, see **Finder**.

Add comments to blocks for collaboration

You can now add comments to blocks.



To view and interact with comments, open the **Review Manager** perspective by performing one of these actions:

- In the Simulink Toolstrip, on the **Modeling** tab, in the **Design** gallery, select **Review Manager**.
- In the lower-right corner of the Simulink Editor canvas, click the perspectives control and select the **Review Manager** perspective.

Both of these actions open the **Review** tab and **Comments** pane.

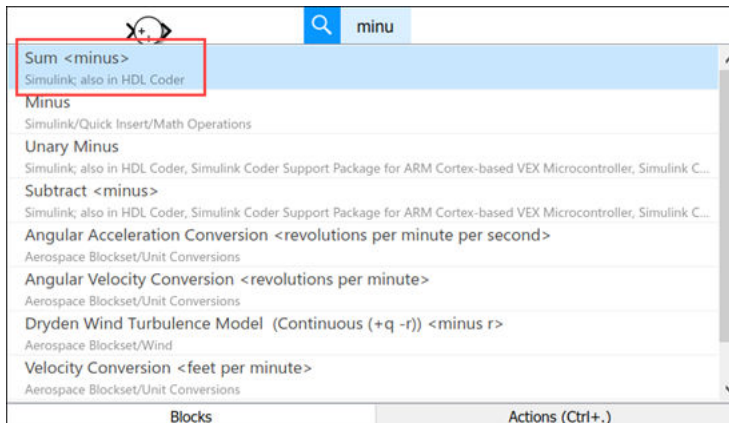
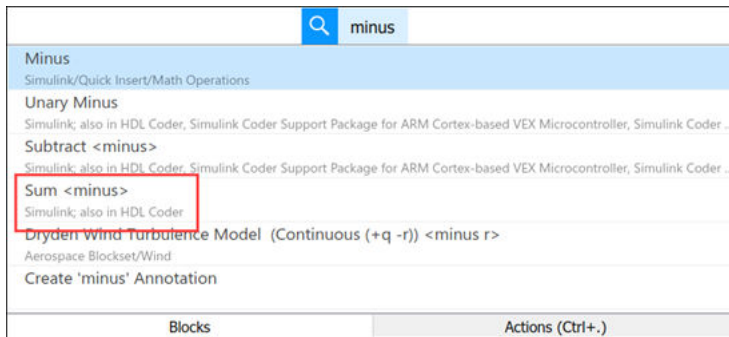
When you close a model, subsystem, or library file that has comments, the comments are saved in an MLDATX file in the same location as the parent. For example, for a model named `mymodel.slx`, the comments file would be named `mymodel_comments.mldatx`.

For more information, see [Collaborate by Adding Comments to Blocks](#).

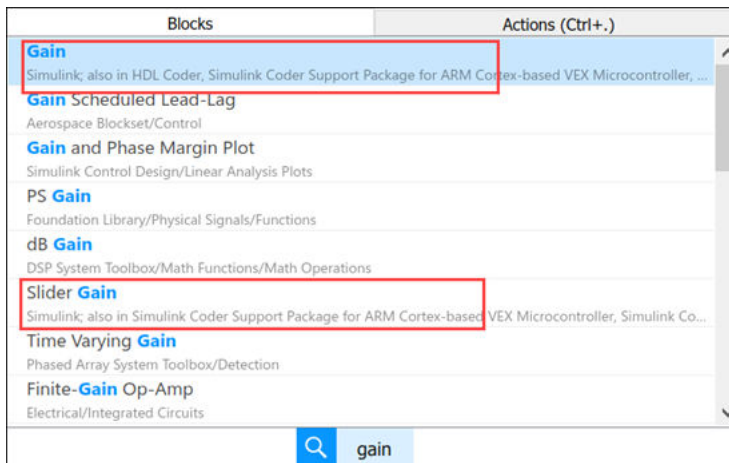
Improved experience in quick insertion of blocks

Quick insert search results have several enhancements for R2021a.

- Prior to R2021a, dynamic ranking was applied only to the first word of the block names. From R2021a onward, dynamic ranking is applied to all words of block name and keywords of the blocks so that they appear at the top in the search result. For example, insert the block Sum using the keyword `minus`. Frequent use of the keyword `minus` enables Simulink to display the block Sum at the top of the search result.

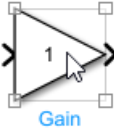


- Simulink has more than five thousand blocks. Some blocks are duplicated in different libraries and appear in quick insert search results. From R2021a onward, duplicate blocks are grouped together and appear once in the quick insert search results, provided the block is available in the Simulink library.



Pause on block diagram to view and edit block names

Starting in R2021a, you can pause on a block in a block diagram to show the block name. You can edit the block name when it displays. This feature provides a convenient way to view and edit block names without changing the name settings at the block and model level.



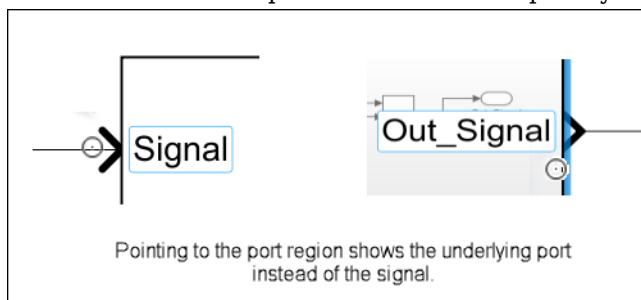
The position and style settings for the block name follows the block level settings.

Select and interact with ports

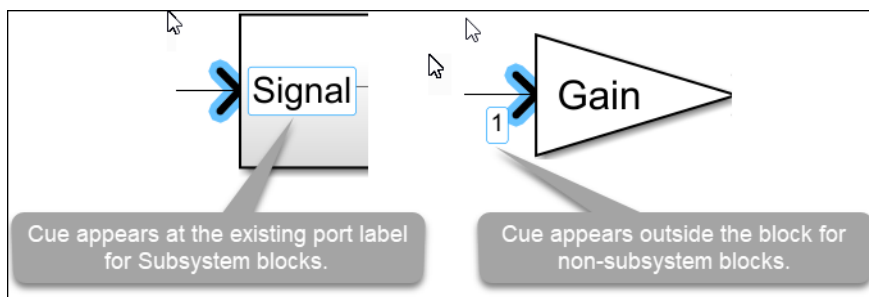
Simulink now supports selection and enhanced interaction with ports.

You can select ports with the following actions:

- Point to the port region of a connected port to view and interact with the underlying port. All interactions with the port show consistent port symbols depending on the type of port.



- Point to a port or select it to see the port label cue. The cue makes the port labels readable in case of overlapping ports and at low zoom levels.
- Click the port label cue to select the port. Selecting the port highlights it.

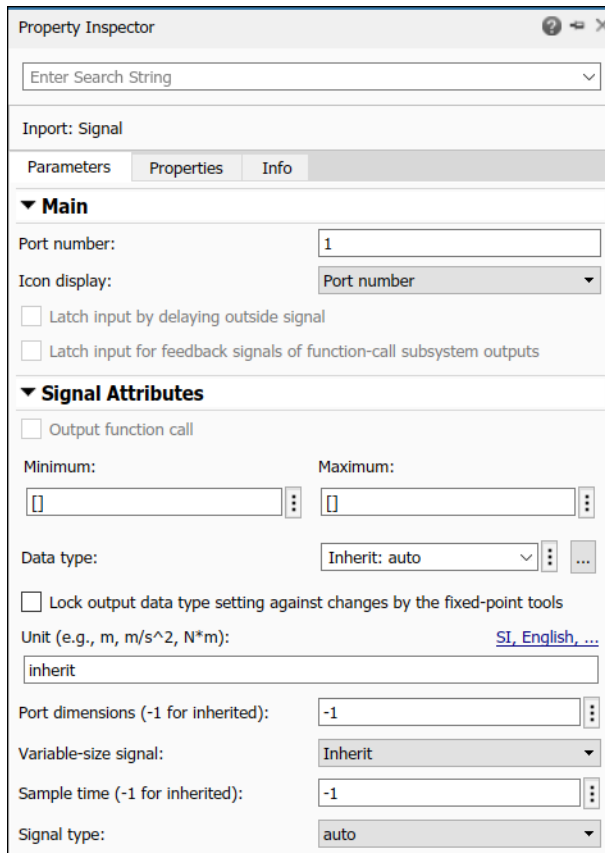


You can delete and move ports on a Subsystem block:

- Delete a selected port by pressing the **Delete** button or selecting the **Delete** option from the context menu. You can delete ports for blocks that support automatic port creation.
- Move a selected port to any side of the block using arrow keys.

You can modify properties of ports and related elements with Property Inspector:

- Modify the properties of port objects, signals, and port blocks using the Property Inspector.



Edit port labels on subsystem blocks

Starting in R2021a, you can directly edit the port labels on these subsystem blocks:

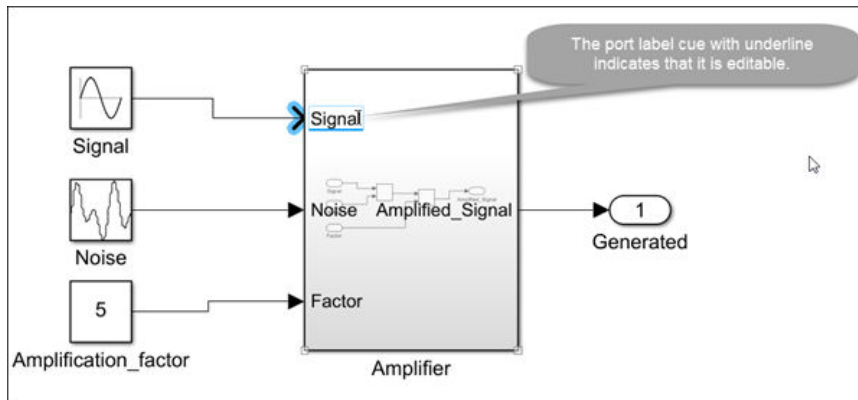
- Regular subsystem
- Subsystem with simple mask
- Variant subsystem
- Chart subsystem

Prior to R2021a, you had to open the subsystem and locate the required port block to rename the port and its related elements.

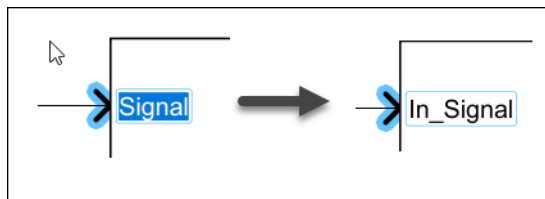
To edit the port label on a subsystem block in the current model:

- 1 Select the port or the subsystem block and point to the port label. An interactive cue appears on the port label. Click the cue to edit the port label.

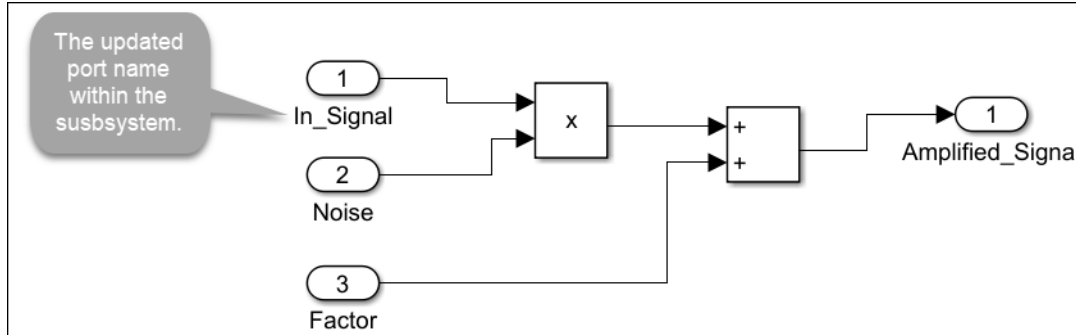
Alternatively, select the port and press **F2**.



2 Enter the name and press **Enter** or **Esc**, or click away.



After the port label is edited, Simulink automatically updates the name of the related port or signal within the subsystem.




This table lists the action depending on the type of port label defined on the subsystem block:

Type of port label	Action
None	Edit not supported.
Port icon	<p>Updates the names of elements in the subsystem depending on the Icon display parameter of the port.</p> <p>For example, if the value of Icon display parameter of the port is Signal name, then both the icon on the port block and the name of the linked signal are updated.</p>
Port block name	Updates the block name of the port in the subsystem.

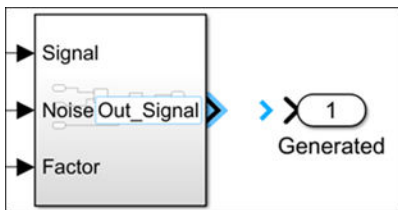
Type of port label	Action
Signal name	Updates the name of the signal linked to the port.


Updates to compatible port connection cues

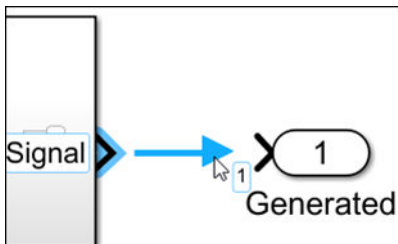
Prior to R2021a, when you selected a port, segment, or terminator in a model, compatible elements that could be connected to the selected element were indicated by highlighting.

Starting in R2021a, you can identify compatible elements in a model by a blue chevron symbol  that appears near them. This enhancement enables you to:

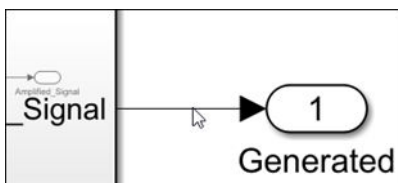
- Distinguish selected elements from compatible elements
- Select the compatible element without making a connection



- 1 Point to  to see the connection cue.



- 2 Click the cue to connect the ports.



Edit enumerated block parameters on block icons

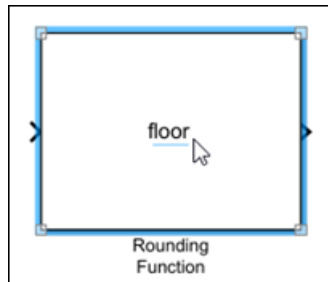
Starting in R2021a, you can edit enumerated block parameter values on the block icon in the Simulink Editor. Previously, you could only edit string block parameters.

- Fixed parameter enumeration

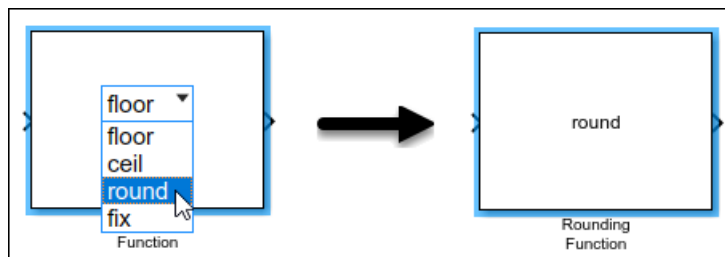
For the block parameters that have a fixed set of values, you can select the value from a list of allowed values using the cursor or arrow keys.

For example, consider a Rounding Function block that allows a list of values for the **Function** parameter.

- 1 Select the block and point to the block icon. The editable parameter value appears underlined.



- 2 Click the underlined text to edit the value. A list appears with allowed values for the **Function** parameter. Select the required value from the list to update the block parameter value.



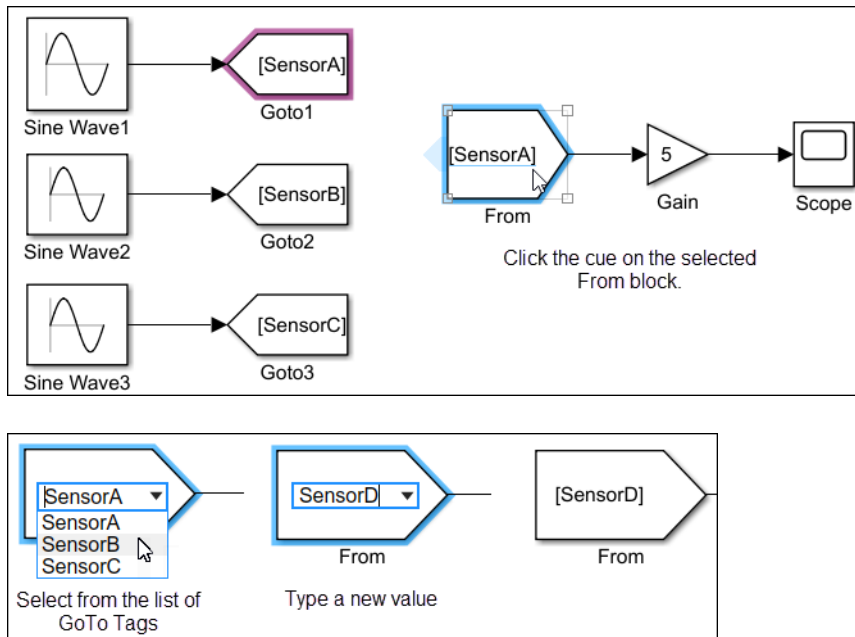
This feature is enabled for these blocks:

- Logical Operator
- Rounding Function
- MinMax
- Sqrt
- Relational Operator
- Math Function
- Complex to Real-Imag
- Dynamic parameter enumeration

For block parameters that allows dynamic values, you can select a value from the list, or you can type the new value. Typing a new value removes unmatched values from the list.

The feature is currently enabled for the From block.

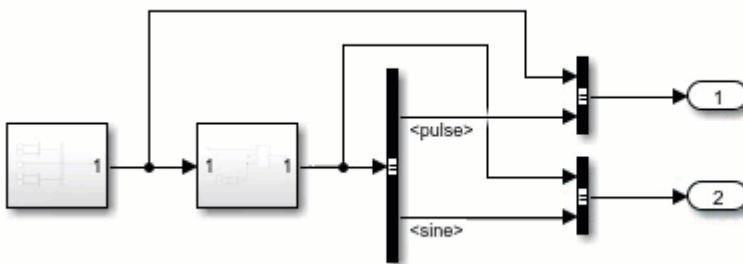
For example, consider a From block with three values for the **GoTo Tag** parameter.



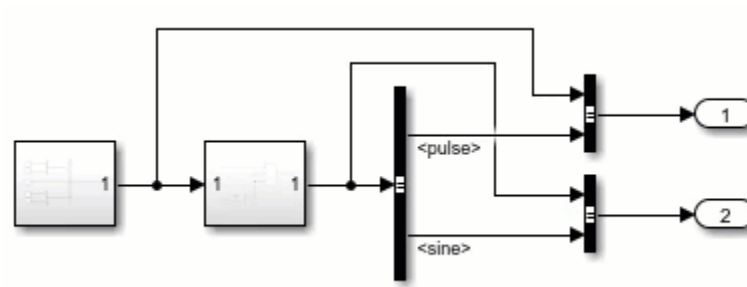
Move multiple lines and blocks together

You can now move the contents of a selection box as a unit. The relationship among elements in the selection box remains the same. Line segments outside of the selection box reroute.

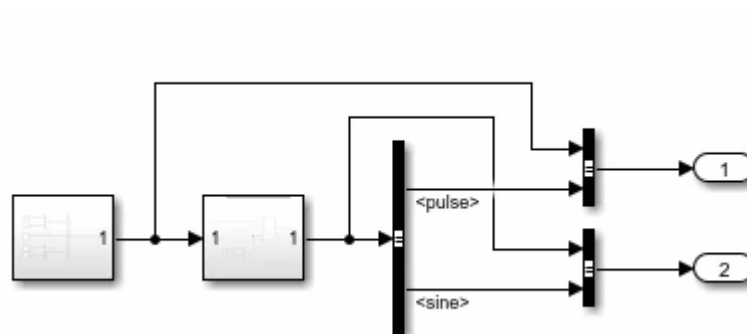
For example, when you drag a selection box that includes line bends, the included line bends remain the same as you move the selection.



By comparison, when you drag a selection box that excludes line bends, the excluded line bends may change as you move the selection.



To move multiple lines together, drag the selection box around the line segments you want to move.



Enhancements to favorite commands in Simulink

In the Simulink Favorite Command Editor, you can now:

- Create favorite functions in addition to favorite scripts.

Functions run in a limited scope and receive callback information from the *cbinfo* variable. You must specify when a favorite command is a function by using the **Type** parameter.

- Specify icons for favorite commands by using the **Icon** parameter.

Custom icons must be PNG files. If you rename, move, or delete a custom icon, the icon no longer appears in the icon list, and any favorite commands that used the icon revert to the default icon.

- Add favorite commands to the quick access toolbar by selecting **Add to quick access toolbar**.

When you add a favorite command to the quick access toolbar, display its label by selecting **Show label in quick access toolbar**. Deleting a favorite command removes the command from the quick access toolbar.

For more information, see [Access Frequently Used Features and Commands in Simulink](#).

Keyboard shortcuts for selecting ports and outermost elements

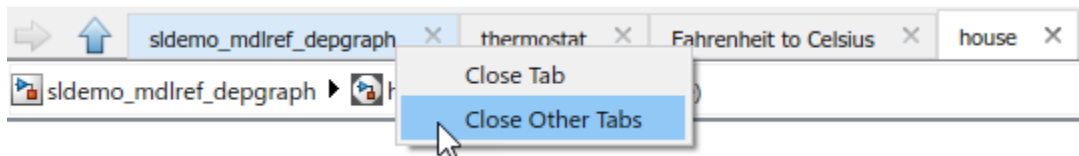
New keyboard shortcuts are available for selecting elements in the current block diagram.

Task	Action
Move cursor to closest element in chosen direction, regardless of similarity	Ctrl+Shift + left, right, up, or down arrow
Change selection to element at top-left of canvas	Ctrl+Home On a Mac, use home .
Change selection to element at bottom-right of canvas	Ctrl+End On a Mac, use end .
Change selection to leftmost element relative to cursor	Home On a Mac, use control+left .
Change selection to rightmost element relative to cursor	End On a Mac, use control+right .

For more information, see Keyboard Shortcuts and Mouse Actions for Simulink Modeling.

New context menu for closing Simulink Editor tabs

In the Simulink Editor, right-click a tab that corresponds to a model component. Select **Close Tab** to close the selected tab or **Close Other Tabs** to close all tabs in the active window except the selected tab.

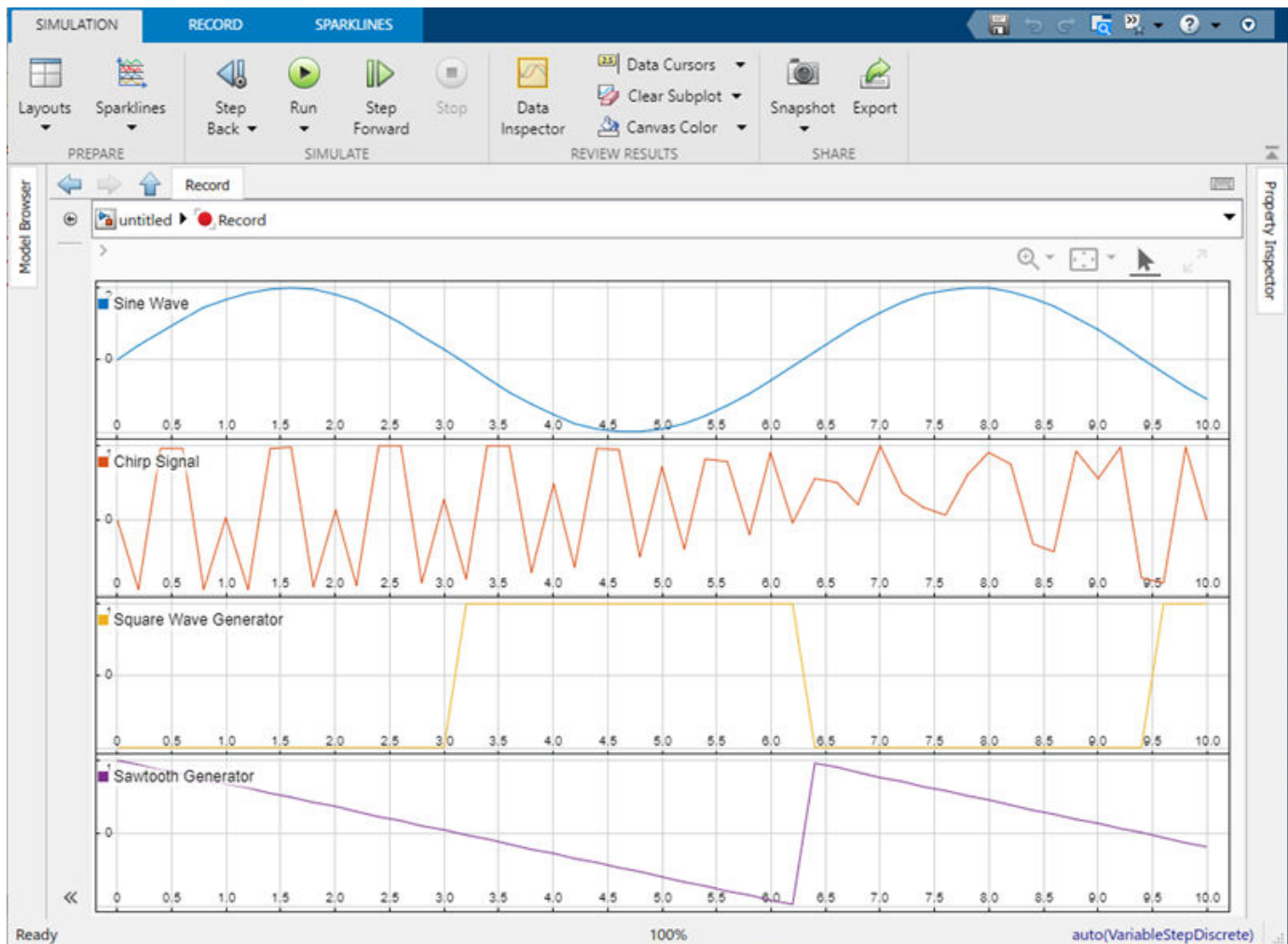


Simulation Analysis and Performance

Log data to the workspace and to a file using the Record block

With the Record block, new in R2021a, you can configure a single block to log data for connected signals to the workspace and a file for all simulation modes. The Record block logs workspace data to disk and supports logging data to MAT, MLDATX, and XLSX files. You can configure the Record block to log data to the workspace, a file, the workspace and a file, or to not record data at all.

The record block also visualizes data for the current simulation.



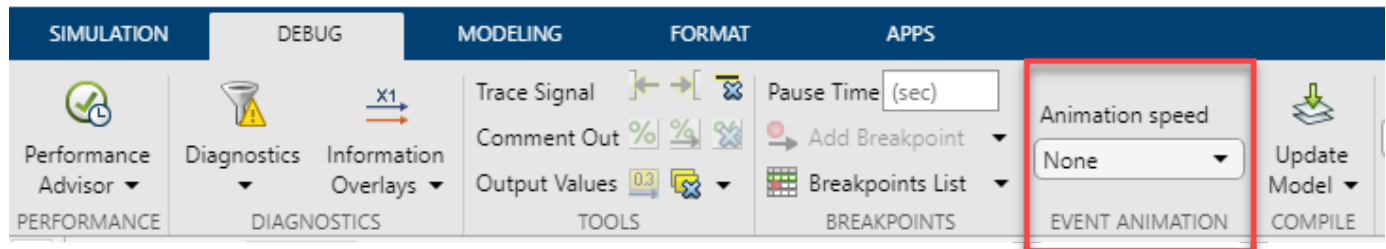
You can customize the data view using all visualization types supported by the Simulation Data Inspector, including:

- Time plots
- Sparklines
- XY plots
- Maps

Record block signal data is also available in the Simulation Data Inspector for further analysis.

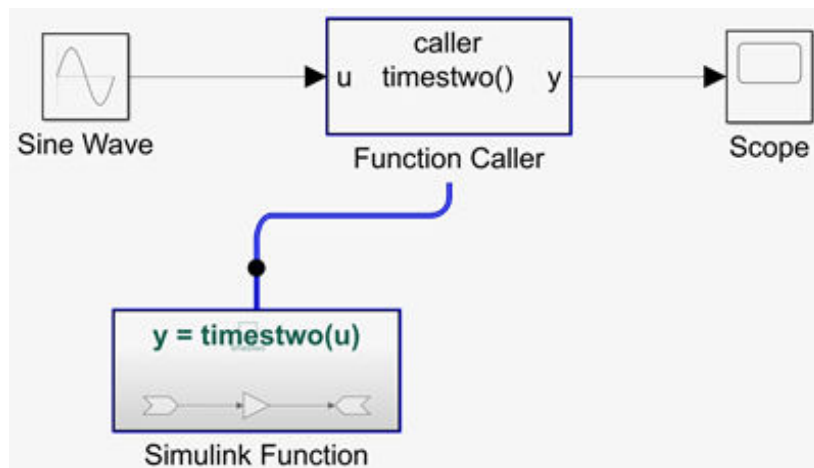
Event Animation: Access event animation via the Simulink toolstrip

Starting in R2021a, the Simulink toolstrip supports animation that visualizes message transitions, events, and Simulink functions during simulation. In the toolstrip, on the **Debug** tab, you can turn on the animation and specify its speed from the **Event Animation** section.



Event Animation is visible when you have event blocks in your model, such as blocks from the Messages & Events library, Stateflow charts, Function-Call Subsystem blocks, Simulink functions, or SimEvents® blocks.

This image shows a still from an animated Simulink function call.



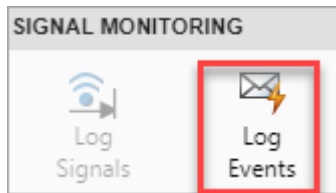
For more information, see [Animate and Understand Sending and Receiving Messages](#).

Event Inspection: Log and visualize events using the Sequence Viewer tool

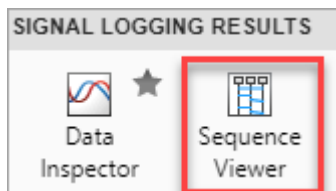
Use the new **Sequence Viewer** tool on the Simulink toolstrip to see the interchange of messages, events, and function calls in Simulink models, System Composer architecture models, and Stateflow charts.

To access the **Sequence Viewer** tool:

- 1 Activate logging events in your model. In the Simulink Toolstrip, on the **Simulation** tab, in the **Prepare** section, click **Log Events**.



- 2 Simulate your model.
- 3 To open the tool, in the Simulink Toolstrip, on the **Simulation** tab, in the **Review Results** section, select **Sequence Viewer**.



The **Log Events** button and **Sequence Viewer** tool are visible when you have event blocks in your model, such as blocks from the Messages & Events library, Stateflow charts, Function-Call Subsystem blocks, Simulink functions, or SimEvents blocks. For more information, see [Animate and Understand Sending and Receiving Messages](#).

Use hardware acceleration to improve simulation execution performance by leveraging SIMD instructions

In R2021a, the new model configuration parameter, **Hardware acceleration** lets you control leveraging SIMD instructions to improve simulation execution performance. For more information, see [Hardware acceleration](#).

To change hardware acceleration settings, on the **Modeling** tab, select **Model Settings**. On the **Simulation Target** pane, under **Advanced parameters**, select **Hardware acceleration**. The parameter has three options.

- **Off** — Hardware acceleration is disabled and no SIMD is used.
- **Leverage generic hardware (Faster, no rebuild)** — Leverage SIMD Instructions for hardware generic to Simulink system requirements. This option does not require rebuilding the model for simulation when the host computer changes.
- **Leverage native hardware (Fastest, rebuild allowed)** — Leverage SIMD instructions for hardware native to the host computer for best performance. This option may require rebuilding the model for simulation when the host computer changes.

You can also change this setting using this command, which takes the values **off**, **generic**, and **native**.

```
set_param(model, 'SimHardwareAcceleration', 'off')
```

A new performance advisor check compares the three hardware acceleration options based on their simulation times. You can select the hardware acceleration level based on this comparison. For more information, see [Select hardware acceleration setting](#).

Multithread co-simulation changes

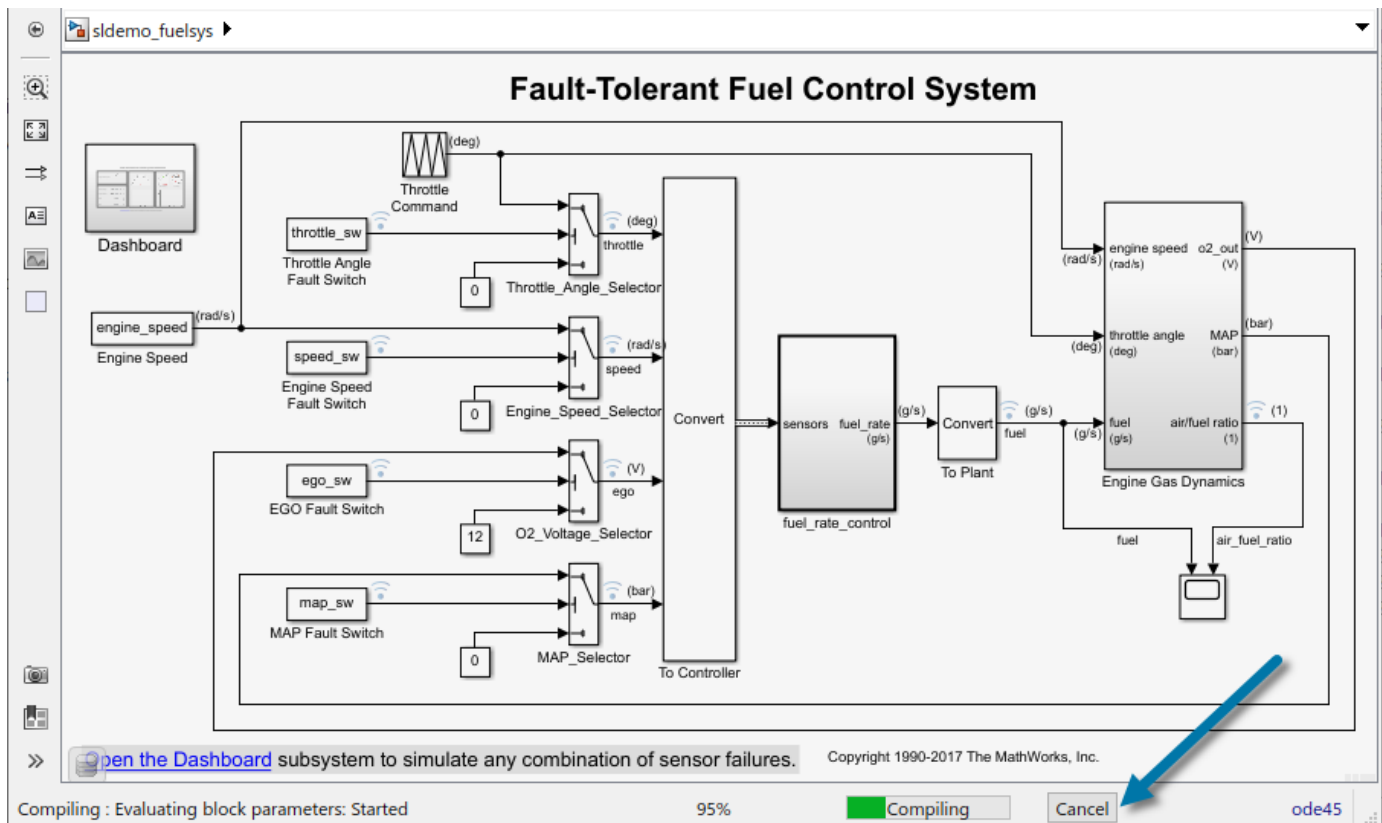
Simulink multicore co-simulation now uses graph-based parallelism to run models for better performance. For multicore simulation, Simulink uses this method to automatically partition nondirect feedthrough co-simulation blocks followed by direct feedthrough co-simulation blocks into different groups. For more information, see [Run Co-Simulation Components on Multiple Cores](#).

For Each subsystem multicore execution

You can now use multicore processing to speed up For Each subsystem execution in Rapid Accelerator mode. See [Multicore Execution Support](#).

Canceling model compilation is now supported

When you compile a model or model hierarchy, you can now cancel model compilation by clicking the **Cancel** button next to the progress bar.



Control maximum size of logged data

Most logging in Simulink logs data to disk, and signal data is read into memory on demand. Under certain conditions, logging can produce so much data that you run out of disk space. The Simulation Data Inspector includes new preferences that allow you to directly control the size of logged data.

Depending on your requirements, you can specify:

- Whether to log data to disk or to memory.
- Where to store logged data.
- A minimum amount of free disk space.
- A maximum size for logged data on disk.
- What to do when the size of logged data approaches your specified limits.

You can choose to immediately stop recording simulation data or to delete data from prior runs in the Simulation Data Inspector to free disk space for continued logging in the current simulation.

When you do not need logged data after simulation, you can choose to view data only during simulation. When you choose this option, the logged data is *not* available in the Simulation Data Inspector or other visualization tools after the simulation ends, and no data is logged to the workspace.

Log variable-size signals using signal logging

Signal logging now supports logging variable-size signals. The data for a variable-size signal is always logged to a `timetable` object, with each signal value logged as a `cell`.

Signal logging does not support variable-size arrays of buses.

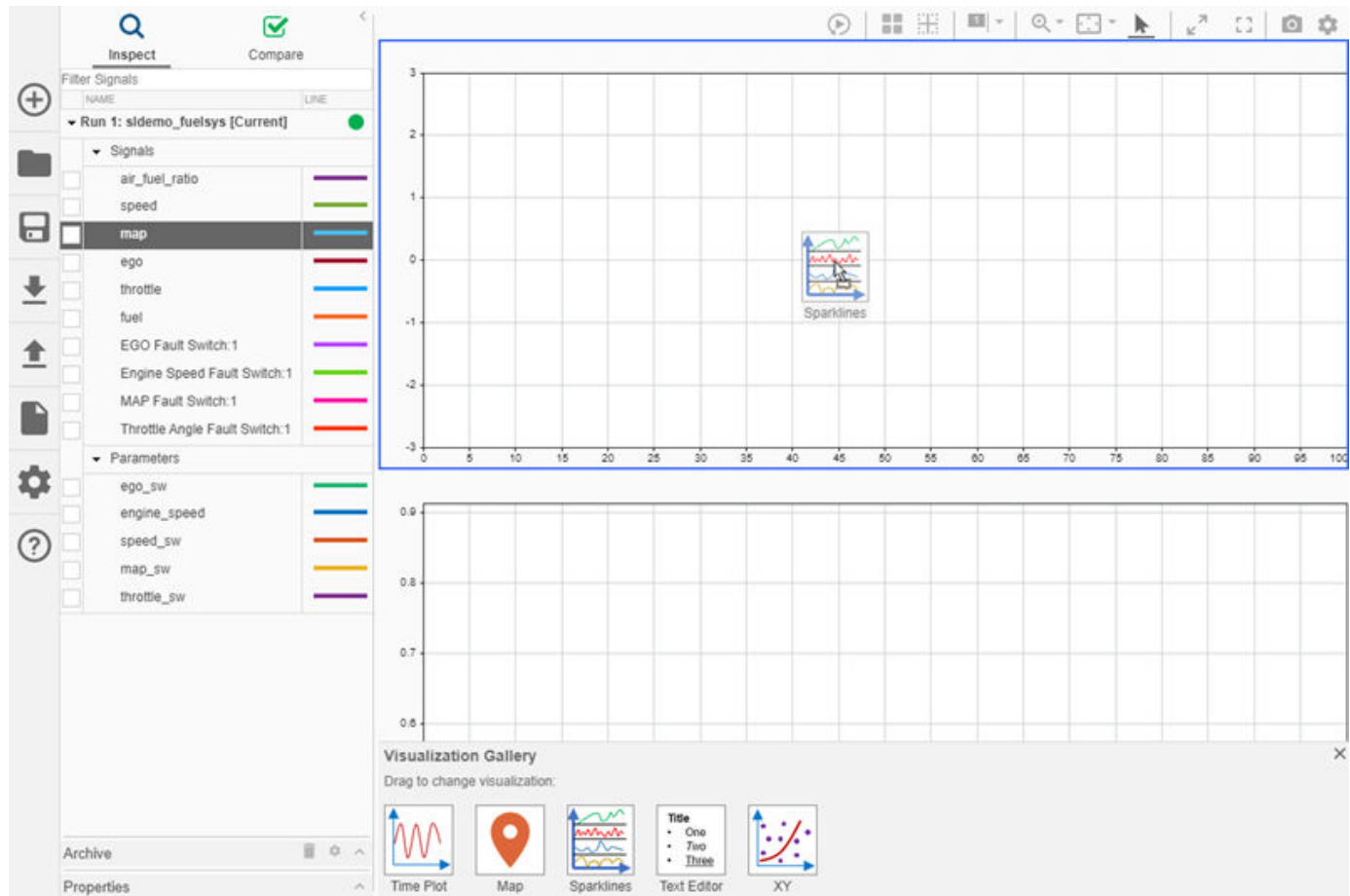
Data access callback supports nonvirtual buses

You can use the data access callback in the Instrumentation Properties for a logged signal to access logged nonvirtual bus data in a MATLAB function during simulation. The data access callback receives the nonvirtual bus data as a structure of arrays that matches the hierarchy of the bus. For more information about using the data access callback, see [Access Data in a MATLAB Function During Simulation](#).

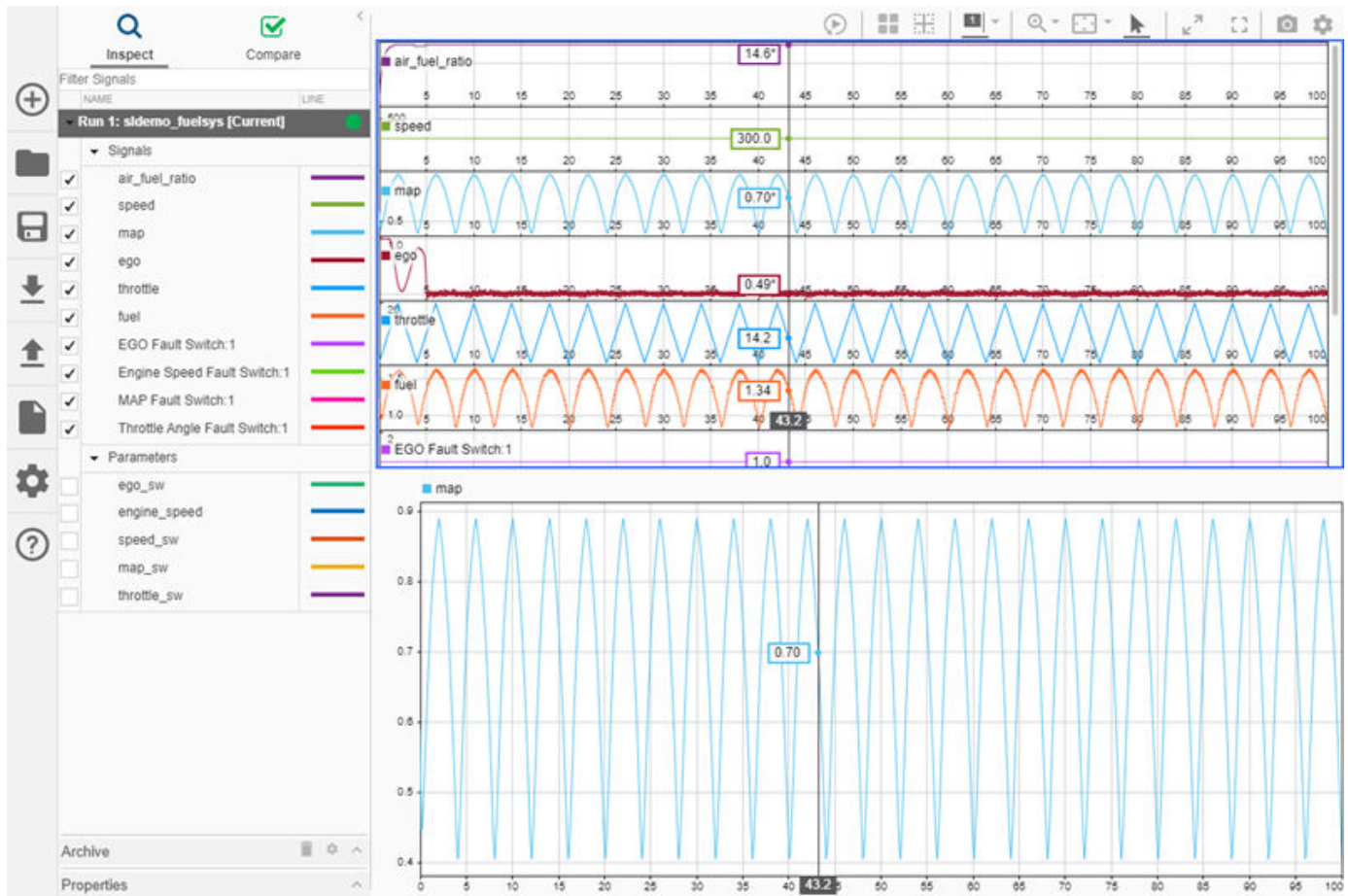
Sparklines visualization in the Simulation Data Inspector

Use a sparklines visualization in the Simulation Data Inspector to easily plot and view many signals at once. The sparklines visualization adds a subplot for each signal you plot. You can specify the minimum height for a subplot. After the subplots fill the visible space in the sparklines visualization, a scroll bar allows you to continue plotting signals.

To add a sparklines visualization to your layout, expand the **Layouts** menu and select **Edit View**. Drag and drop the **Sparklines** icon onto the subplot where you want the sparklines visualization.

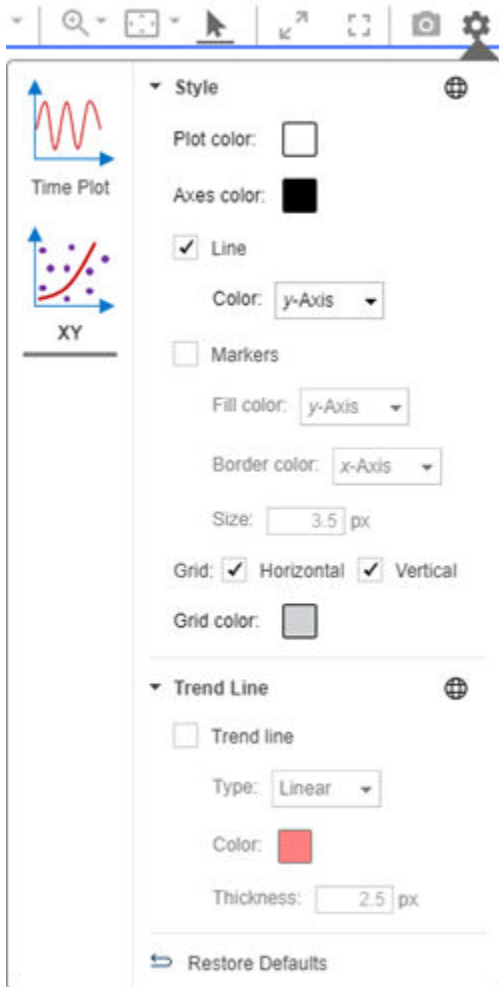


To plot signals on the sparklines visualization, you can select the check box next to each signal you want to plot, or you can select one or more signals to drag and drop onto the sparklines visualization.



Configure colors for plot area, grid, and tick labels on XY plots in the Simulation Data Inspector

You can now configure color options for XY plots in the Simulation Data Inspector and the Record block. Color settings are in the **Visualization Settings** for the XY plot. **Plot color** sets the color for the plot area. **Axes color** sets the color for the ticks, tick labels, and legend labels. **Grid color** specifies the color for grid lines.



Simulation Data Inspector user interface enhancements

The Simulation Data Inspector has several new UI enhancements:

- Fit plots to view on only the t -axis or y -axis in addition to the existing capability to fit to view on both axes.
- Long signal and property names display in a tooltip when you pause on the table entry.
- By default, signals are grouped by their domain, which indicates how the signal was logged. Domains include **Signals** for signals created through signal logging, **Outputports** for signals created by output logging, and **States** for signals created by states logging.
- You can specify the line width for plotted signals.

Import ULG files into the Simulation Data Inspector

If you have the UAV Toolbox, the Simulation Data Inspector supports importing flight log data from a ULG file for visualization and analysis.

Programmatic interface for Simulation Data Inspector cursors

New functions allow you to access and control the cursors in the Simulation Data Inspector. Using the new functions, you can:

- Control the number of cursors visible in the Simulation Data Inspector.
 - `Simulink.sdi.setNumCursors`
 - `Simulink.sdi.getNumCursors`
- Specify and access cursor positions in the Simulation Data Inspector.
 - `Simulink.sdi.getCursorPositions`
 - `Simulink.sdi.setCursorPositions`
- Register a callback that executes when cursors move in the Simulation Data Inspector. You can use a cursor callback to synchronize cursors between the Simulation Data Inspector and a custom MATLAB visualization.
 - `Simulink.sdi.registerCursorCallback`
 - `Simulink.sdi.unregisterCursorCallback`

Programmatically specify Simulation Data Inspector time plot axis limits

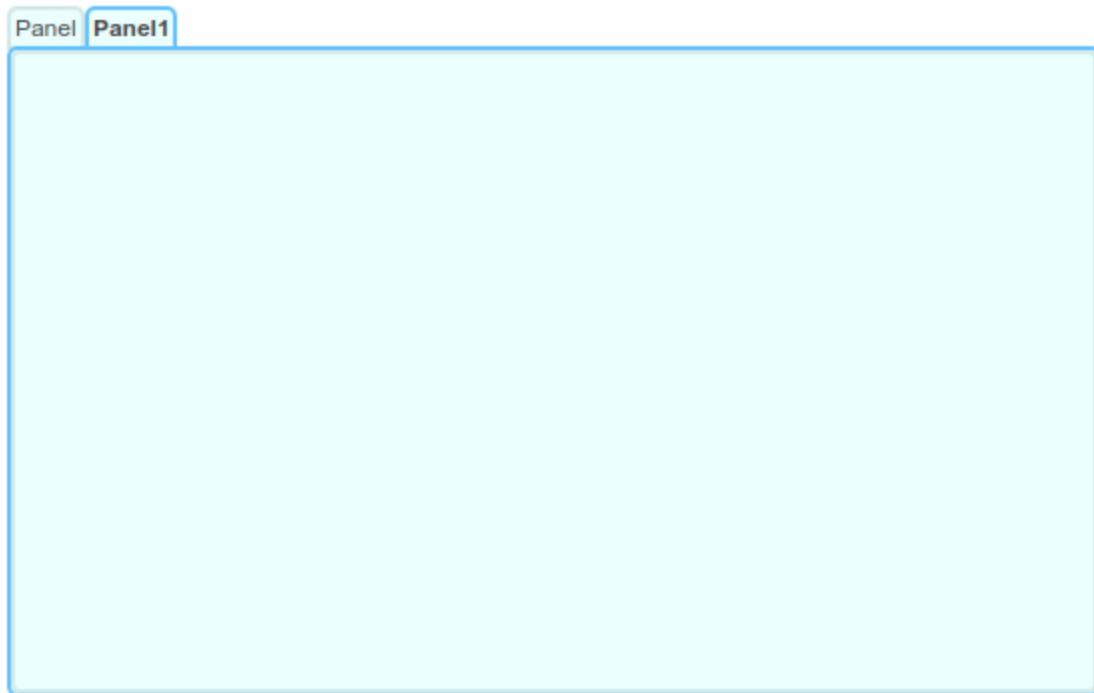
New functions allow you to control the axis limits for time plots in the Simulation Data Inspector programmatically on both the **Inspect** and **Compare** panes.

- `Simulink.sdi.getSubplotLimits`
- `Simulink.sdi.setSubplotLimits`

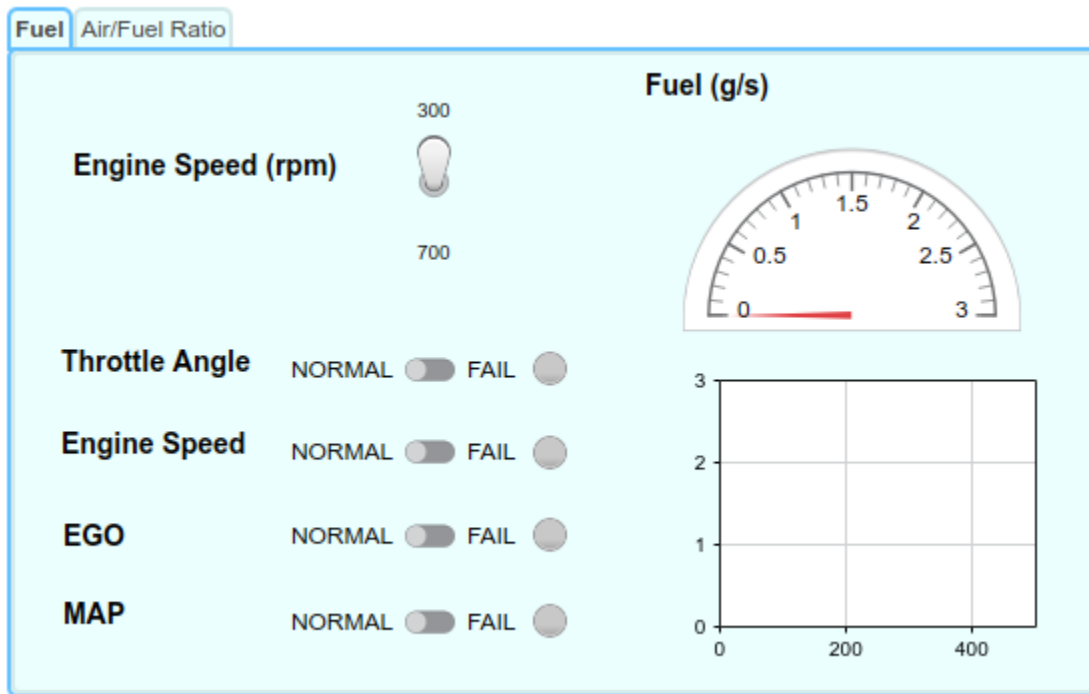
You can specify limits for the time axis and y-axis separately or together. You can specify the limits as a minimum and maximum or as a range.

Manage multiple panels using tabs

You can manage multiple panels in your model using tabs. To add a tab to a panel, select the panel. Pause on the three dots that appear above the panel and click **Add Tab**.



The tabs are visible along the top of the panel when the panel is selected. You do not need to enter panel edit mode to add a tab or to switch to another tab. Each panel within a group of tabbed panels has its own name property, and tabs are labeled with the panel name. You can rename a panel using the Property Inspector.



Generate C++ code for half-precision floating-point data types in Simulink

You can now generate C++ code for the half-precision floating-point data type in Simulink. The half-precision data type occupies only 16 bits of memory, but its floating-point representation enables it to handle wider dynamic ranges than integer or fixed-point data types of the same size. Using the half-precision data type in Simulink requires a Fixed-Point Designer license.

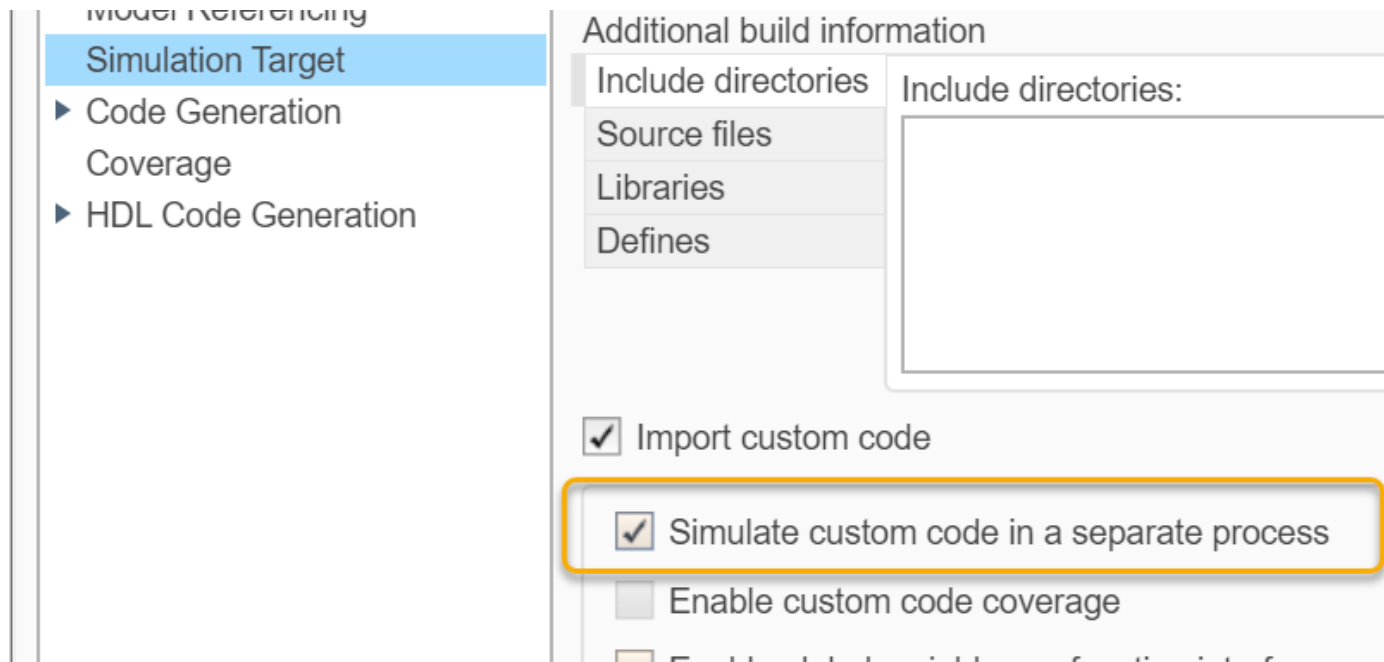
Run custom C/C++ code in a separate process during model simulation

A new option is available to have external C/C++ code run in a separate process during simulation of a Simulink model. Previously, a run-time error in external C/C++ code could cause MATLAB to crash during simulation.

You can integrate custom C/C++ code into a Simulink model by using these blocks:

- C Caller
- C Function
- MATLAB Function
- MATLAB System
- Stateflow chart

To enable this feature, in the **Simulation Target** pane of the model configuration parameters, select **Simulate custom code in a separate process**.



This feature enables custom code to run in a separate process during model simulation, thus preventing a MATLAB crash due to bad custom code. A run-time exception in the custom code produces an error message in Simulink with detailed information about the exception, such as which block or line number is responsible, to help resolve any issues with the custom code, and allows you to launch an external debugger. For more information, see [Simulate Custom Code in a Separate Process](#).

Report Generation for Simulink Profiler

You can now generate a standalone report from the Simulink Profiler. After profiling your model, click



and select **Generate Report**.

Discrete State Linearization in Referenced Models

Simulink now supports linearization of discrete states inside referenced models configured to simulate in Accelerated mode.

Dynamic Switching of the Algebraic Loop Solver

Simulink now automatically switches between the Trust-Region or Line-Search algebraic loop solver during simulation of your model for the best simulation performance.

Functionality being removed or changed

Simulink will no longer support loading ModelDataLogs data as simulation input

Warns

In R2016a, the ability to log data using the `ModelDataLogs` format was removed. In a future release, the ability to load existing data stored in the `ModelDataLogs` format as input for a simulation will be removed.

Starting in R2021a, you see a warning when you load simulation input data that uses the `ModelDataLogs` format.

You can convert data stored using the `ModelDataLogs` format to use the `Dataset` format by using the `Simulink.ModelDataLogs.convertToDataset` function.

Signals connected to dashboard blocks that display instantaneous values no longer log to the Simulation Data Inspector

Behavior change

In prior releases, signals connected to dashboard blocks logged to the Simulation Data Inspector regardless of whether signals were marked for logging. Starting in R2021a, signals connected to dashboard blocks that display an instantaneous value, such as the Gauge block and the Display block, no longer log to the Simulation Data Inspector unless you mark the signals for logging. Signals connected to Dashboard Scope blocks and variables and parameters connected to dashboard blocks still log to the Simulation Data Inspector.

MultithreadedSim model parameter no longer uses 'on'

Behavior change

The `MultithreadedSim` model parameter no longer uses the 'on' value. Instead, use the 'auto' value.

Starting in R2021a, when you open a model created in a previous release, Simulink interprets the 'on' setting to be the same as 'auto'.

Component-Based Modeling

Simulink Code Importer: Import C code as reusable Simulink libraries

Starting in R2021a, you can use the Simulink Code Importer tool to build a Simulink library from your custom C/C++ code library.

The Simulink Code Importer brings custom code into a Simulink library of C Caller blocks as callable functions, after analyzing the custom code for functions and their dependencies, including global variables and types used by the functions.

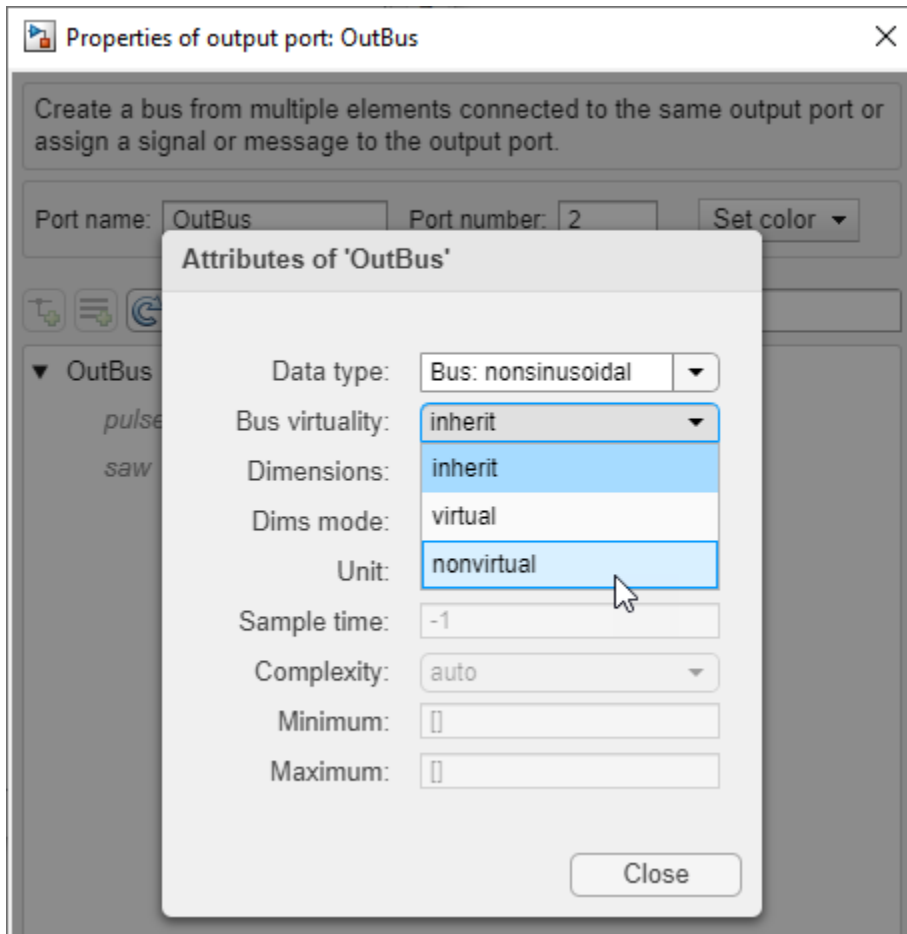
To use the Simulink Code Importer, go to the **Modeling** tab on the Simulink toolstrip. From **Design**, choose **Code Importer**. A series of screens then leads you through the process of creating a Simulink library from your custom code.

Alternatively, you can create an object of the `Simulink.CodeImporter` class and run the Simulink Code Importer tool using a command-line interface.

See [Import Custom C/C++ Code Using the Simulink Code Importer](#).

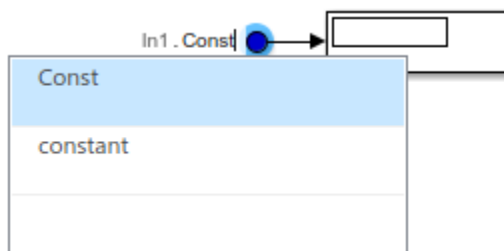
Bus ports support nonvirtual buses, arrays of buses, and case-insensitive searches

When an In Bus Element or Out Bus Element block specifies a `Simulink.Bus` object data type for a bus, you can now inherit or specify the virtuality of the bus with the new **Bus virtuality** parameter.



To define an array of buses at a subsystem or model interface, set **Bus virtuality** to `nonvirtual` and **Dimensions** to the number of buses required in the array.

When you type the name of a bus element in the label of an In Bus Element block, the list of suggestions now includes case-insensitive matches.



Messages: Use Message Merge block to combine multiple message lines into a single message line

The Message Merge block combines multiple message lines into a single message line. You can specify the number of message lines to merge.

Use this block when you want to merge multiple incoming messages and output to a single destination, such as a queue or a message storage block. For more information, see [Merge Message Lines Using a Message Merge Block](#).

Override, edit, and reset configuration parameters using configuration reference functions

In R2021a, you can use these new functions to programmatically interact with parameter overrides in a configuration reference. You can override parameters, determine which parameters are overridden, and restore parameters to their referenced values. Previously, you had to use the Configuration Reference dialog box to change parameter overrides.

Function	Description
<code>configset.reference.overrideParameter</code>	Override the parameter in the configuration reference and specify a new value.
<code>configset.reference.hasOverriddenParameters</code>	Determine if the configuration reference of the model has overridden parameters.
<code>configset.reference.getOverriddenParameters</code>	List the parameters that are overridden in the configuration reference of the model.
<code>configset.reference.isParameterOverridden</code>	Determine if the parameter is overridden in the configuration reference of the model.
<code>configset.reference.restoreAllOverriddenParameters</code>	Restore all overridden parameters to the values in the referenced configuration set for the model.
<code>configset.reference.restoreOverriddenParameter</code>	Restore the value of the specified parameter to the value in the referenced configuration set.

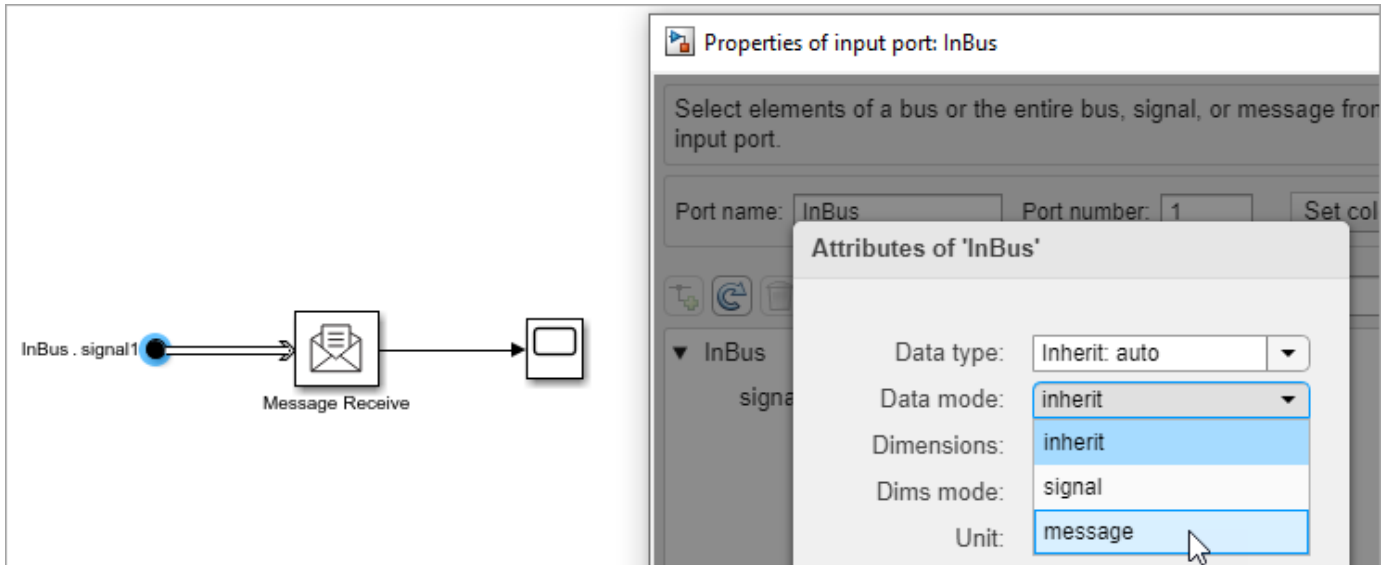
For more information on overriding configuration parameters, see [Change Parameter Value in a Configuration Reference](#).

Specify data mode for virtual bus elements at component interface

In R2021a, you can manually specify virtual bus elements' data mode at the component interface using In Bus Element and Out Bus Element blocks. For an example, see [Use Virtual Buses with Message Elements for Component Communication](#).

Select one of the following values for the **Data mode** parameter:

- `inherit` — The virtual bus element inherits its data mode from the model.
- `signal` — The virtual bus element is a signal.
- `message` — The virtual bus element is a message.



Get list of choices in Variant Subsystems using VariantChoices parameter

You can get a list of variant choices in a Variant Subsystem using the read-only parameter `VariantChoices`.

Here is an example:

```
varchoices = get_param(gcb, 'VariantChoices');
```

The parameter returns a 1-by-*N* struct array with fields:

Name: Name of the variant control

BlockName: Block path name

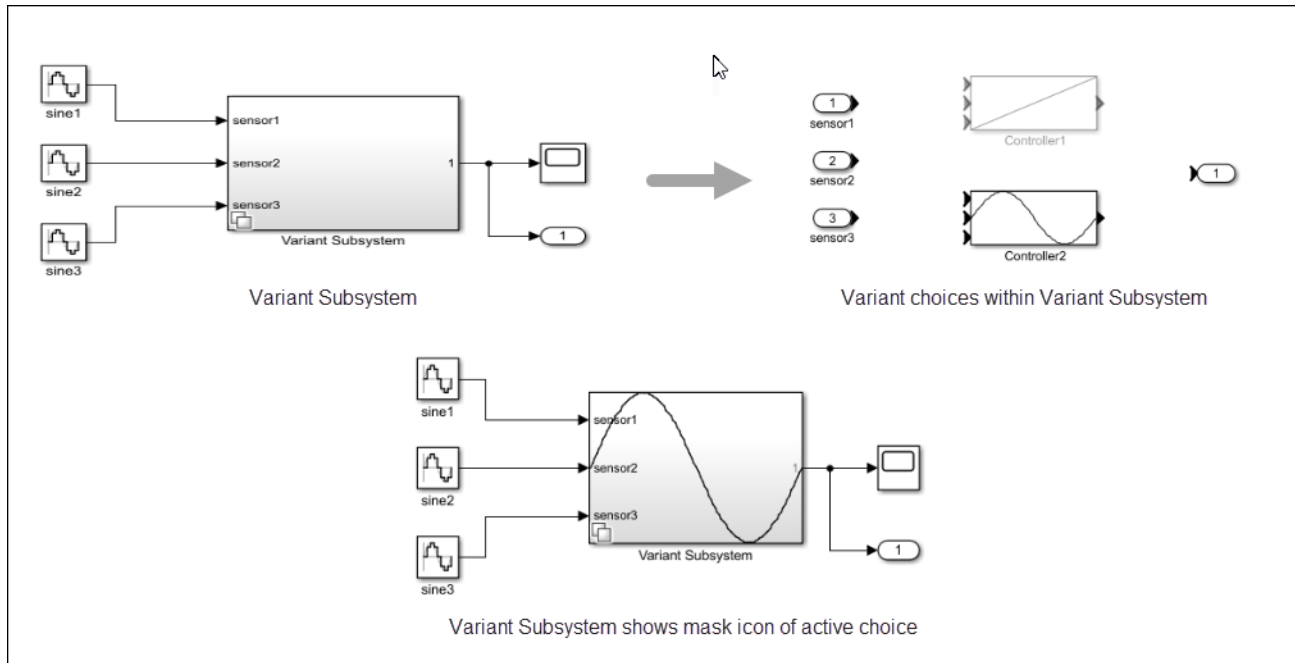
For example, consider a Variant Subsystem with the variant choices `Linear Controller` and `Nonlinear Controller` and the variant controls `VSS_MODE==1` and `VSS_MODE==2`. The `VariantChoices` parameter returns the following:

Fields	Name	BlockName
1	"VSS_MODE==1"	'sldemo_variant_subsystems/Controller/Linear Controller'
2	"VSS_MODE==2"	'sldemo_variant_subsystems/Controller/Nonlinear Controller'

Promote mask icon of active variant to Variant Subsystem automatically

Starting in R2021a, Simulink automatically displays the mask icon of the active variant block on the parent Variant Subsystem. Prior to R2021a, you had to provide icon drawing commands on the mask of the Variant Subsystem to display the mask icon of the underlying active variant block.

For example, consider this model with a Variant Subsystem. It has two variant choices, Controller1 and Controller2, that have mask icons on them. If you set the active variant choice as Controller2, its mask icon gets promoted to the Variant Subsystem.



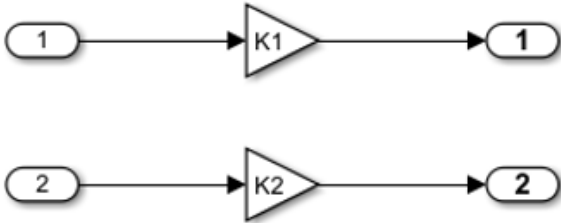
For an example, see Automatic Mask Icon Promotion in Variant Subsystems.

Reuse block parameters with different values by using variant parameters

Variant parameters provide you a way to design one model for many systems that differ in block parameter values. You can group different set of values based on the system to simulate. Each set of value is associated with a variant condition expression. During simulation, if a variant condition evaluates to `true`, all the values that are associated with that condition become active. If a variant condition evaluates to `false`, all the values that are associated with that condition become inactive. You can switch between the values of the variant parameters just by changing the value of the variant control variable. With variant parameters, you are not required to regenerate the code for different values of variant parameters. The generated code has preprocessor conditionals for all the active and inactive values of all the variant parameters in the model.

Consider this model.

The **Gain** parameters of the Gain blocks are variant parameters with the values set to K1 and K2. The variable K1 has two values 3.5 and 8.5. The variable K2 has two values 4.5 and 9.5. Simulink chooses active value of K1 and K2 based on the value of the variant control variable V. During simulation, if `V==1` evaluates to `true`, the value of K1 is set to 3.5 and the value of K2 is set to 4.5. If `V==2` evaluates to `true`, the value of K1 is set to 8.5 and the value of K2 is set to 9.5.



if	K1	K2
V == 1	3.5	8.5
V == 2	4.5	9.5

V is a variant control variable.

Gain parameters are variant parameters specified as K1 and K2. The value of K1 and K2 is based on the value of V.

For more information, see [Create a Simple Variant Parameter Model](#).

When you specify the variant activation time of the variant control variable V as 'code compile', all the active and inactive values of K1 and K2 are included as macros in the generated code. The values are enclosed in preprocessor conditionals #if and #elif.

```
#if V == 1
#define rtCP_Gain1_K1      (3.5)
#elif V == 2
#define rtCP_Gain1_K1      (8.5)
#endif

#if V == 1
#define rtCP_Gain_K2      (4.5)
#elif V == 2
#define rtCP_Gain_K2      (9.5)
#endif

mUDInlined_Y.Out1 = rtCP_Gain_K2 * mUDInlined_U.In2;
mUDInlined_Y.Out2 = rtCP_Gain1_K1 * mUDInlined_U.In1;
```

Limit scope of variant control variables for Inline Variant blocks and Variant Connector blocks

Previously, you could use the mask and model workspace variables of only Variant Subsystems as variant control variables. In R2021a, this functionality has been extended to support the Variant Sink, Variant Source, and Variant Connector (Simscape) blocks. Using the mask and model workspace variables as variant control variables limit the scope of the variables. Limiting the scope helps avoid name conflicts and establish clear ownership of the variable between the blocks. It also enables you to use same names for variables in different scopes.

For more information, see [Approaches to Control Active Variant Choice of a Variant Block](#).

Model reference simulation targets now support C++

When you set **Configuration Parameters > Simulation Target > Language** to C++, new and rebuilt model reference simulation targets use the C++ programming language.

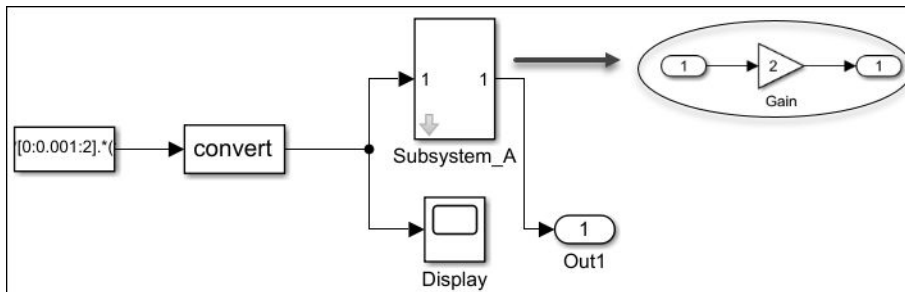
For more information, see [Manage Simulation Targets for Referenced Models](#).

Allow commenting out from and commenting through blocks in model, block, and mask initialization

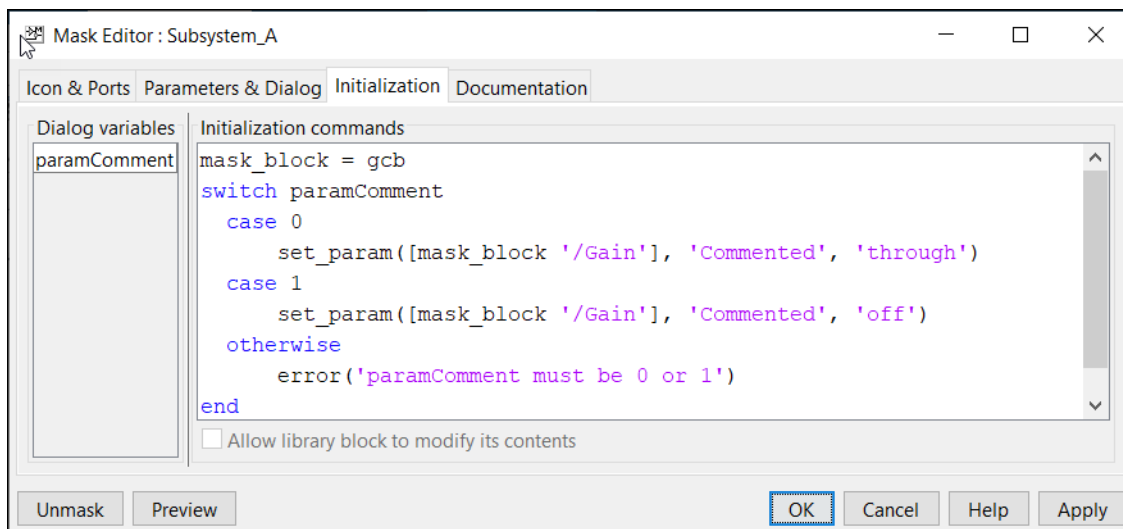
Starting in R2021a, you can comment out and comment through blocks from these additional functions and commands:

- Initialization function in a model (model InitFcn) or a block (block InitFcn)
- Mask initialization commands for masked Subsystem blocks

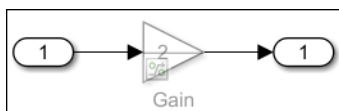
For example, consider this model. It has a masked Subsystem block, `Subsystem_A` with a mask parameter **paramComment**.



You can comment through the child block **Gain** using this mask initialization code for `Subsystem_A`. Set the value of the parameter **paramComment** to 0.



The **Gain** block is excluded from simulation.



Block and mask initialization do not support:

- Commenting out a block from its own callback code
- Commenting out blocks other than the child blocks of a subsystem

Support for variant conditions on function-call signals through Variant Source and Variant Subsystem blocks

In R2021a, you can analyze variant choices of function-call signals on Variant Source and Variant Subsystem blocks when the **Variant activation time** parameter is set to `update diagram`, `analyze all choices` or `code compile`. Previously, you could analyze the variation of function-call signals only when the **Variant activation time** was set to `update diagram`.

To analyze the variant choice in a Variant Source block, select the **Output function call** parameter available in the block parameters dialog box. For a Variant Subsystem block, select the **Output function call** parameter from the outport dialog box.

Transfer existing test harnesses to subsystem block diagram while converting subsystem to subsystem reference

You can now transfer existing test harnesses to the subsystem block diagram while converting the subsystem to subsystem reference. To transfer the test harnesses, in the **Convert to Subsystem Reference** dialog box, select the **Transfer test harness(es) to subsystem block diagram** check box.

Promote block parameters to system mask in subsystem reference

You can now promote block parameters to the system mask of a subsystem block diagram.

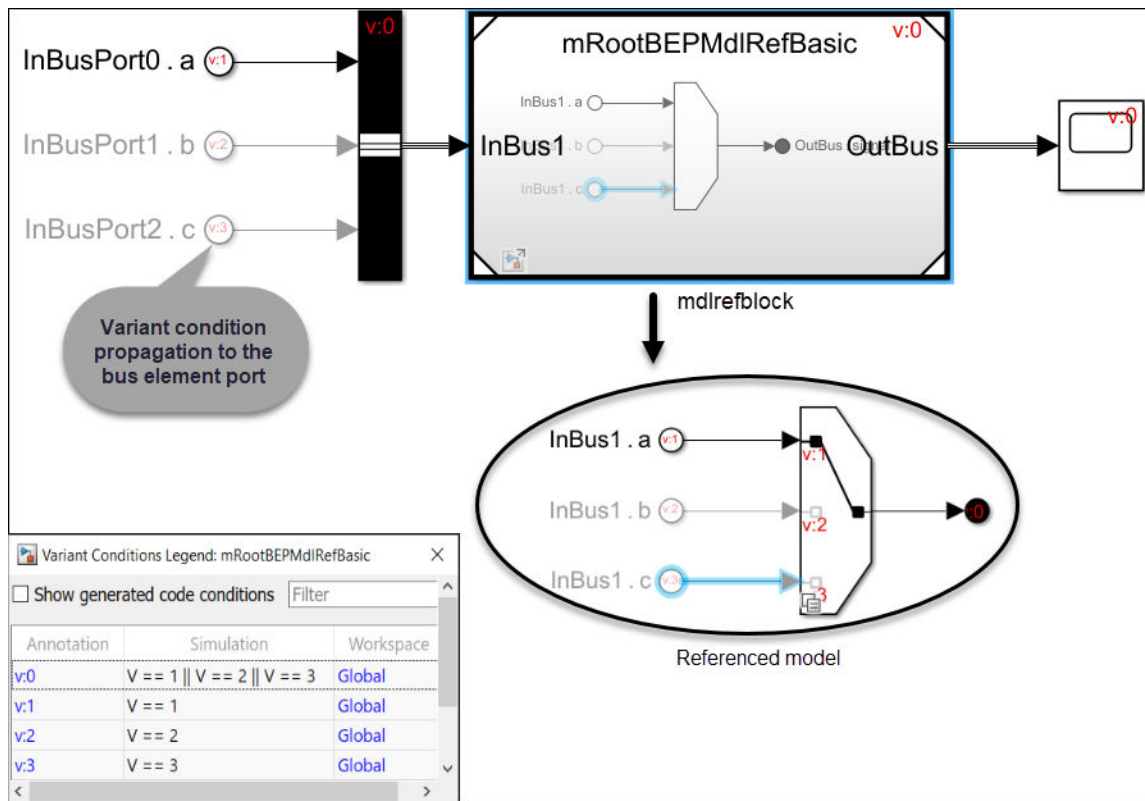
Replace continuous block with discrete versions inside Variant Subsystem

In R2021a, you can discretize a continuous block or a subsystem in a model using the **Model Discretizer** and store multiple discretization candidates inside a Variant Subsystem. Previously, you could store multiple discrete versions in a library as different block choices for the Configurable Subsystem. A Variant Subsystem offers enhanced capabilities compared to Configurable Subsystems. For more information, see [Convert Configurable Subsystem to Variant Subsystem](#).

Support for variant condition propagation to root-level In Bus Element, Out Bus Element blocks and Simulink.Bus objects

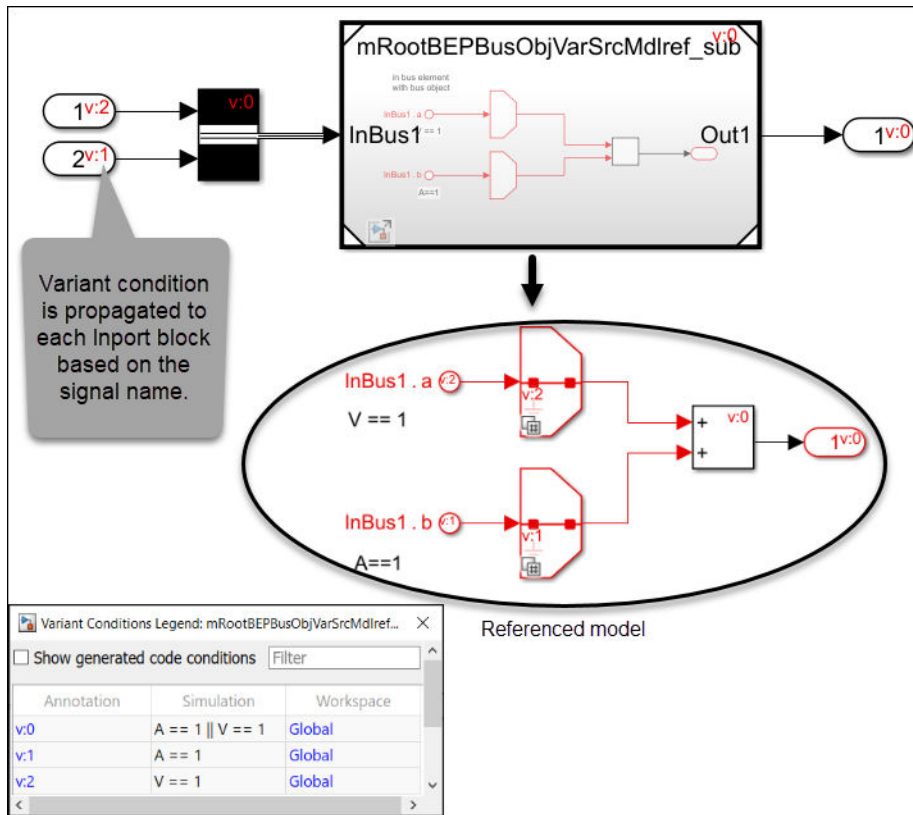
- You can now use In Bus Element and Out Bus Element blocks at root-level when variant blocks are present in a Simulink model. Previously, models with a variant condition on such blocks would produce an error during simulation.

Consider this model with a Model block and three In Bus Element blocks with signals a, b, and c, respectively. The referenced model has a Variant Source block with variant conditions on each signal. The variant conditions get propagated from the Variant Source block to the bus element blocks in the parent model.



- Simulink.Bus objects that are part of Inport, Outport, In Bus Element, and Out Bus Element blocks now support variant condition propagation across model reference boundaries based on signal names. Previously, a logical OR of all variant conditions was set on each block.

Consider this model with a Model block and two Inport blocks. In the referenced model, variant conditions are applied on each input bus signal.



Variant Configuration Analysis tool now supports the Variant Connector block

The Variant Configuration Analysis tool now displays Variant Connector blocks when the Variant option is selected from the **View Blocks** list in the Variant Analysis window.

Functionality being removed or changed

Variants option will be removed from `find_system`, `find_mdrefs`, and `Simulink.FindOptions Warns`

The Variants option will be removed from the `find_system` and `find_mdrefs` functions, and from the `Simulink.FindOptions` class in a future release. In R2021a, scripts that use the Variants option continue to work but throw a warning.

To find variant blocks in a model that are active during simulation or code generation, you can compile the model and use the `find_system` function with `MatchFilter` options. For more information on `MatchFilter` options, see `find_system`.

Configuration parameter `Check runtime output of execution context` being removed

The model configuration parameter `Check runtime output of execution context` (`CheckExecutionContextRuntimeOutputMsg`) is being removed in R2021a.

Simulink will ignore this parameter in models saved in releases prior to R2021a and will not check for potential output differences from pre-2004 releases.

Changes to the default behavior of `find_system` with `MatchFilter` option for variant blocks

Behavior change

- Starting in R2021a, the `find_system` function with the `MatchFilter` option applies the filters on the active and inactive variant choices by default.

For example, consider a model with a Variant Subsystem that has two variant choices, Linear Controller and Non-Linear Controller. The filter function `findAddBlocks` finds all the Add blocks in the model.

```
function match = findAddBlocks(handle)
match = strcmp(get_param(handle, 'Type'), 'block') &&...
    strcmp(get_param(handle, 'BlockType'), 'Sum');
end
```

This command returns both the active and inactive Add blocks in the model.

```
add_blocks=find_system('sldemo_variant_subsystems','MatchFilter',...
@findAddBlocks)
```

```
add_blocks =
```

```
    2x1 cell array
```

```
    {'sldemo_variant_subsystems/Controller/Linear Controller/Add' }
    {'sldemo_variant_subsystems/Controller/Nonlinear Controller/Add'}
```

- `find_system` does not support the use of `MatchFilter` along with the `Variants` option.

This command produces an error:

```
find_system(bdroot,'MatchFilter',@Simulink.match.activeVariants,...
    'Variants','ActiveVariants');
```

Convert Configurable Subsystem blocks to Variant Subsystem blocks

Warns

The Configurable Subsystem block will be removed in a future release. Configurable Subsystem blocks in existing models must be converted to Variant Subsystem blocks. Variant Subsystems offer more capabilities than Configurable Subsystems, including these advantages:

- Model blocks and Subsystem blocks can be mixed as variant systems.
- Flexible input or output ports support the same or a different number of input and output ports.
- Variant Subsystems are not dependent on Simulink libraries.
- Variant Subsystems have advanced code generation capabilities.

To convert a Configurable Subsystem block to a Variant Subsystem block, run the **Upgrade Advisor**.

During conversion, Simulink performs these operations:

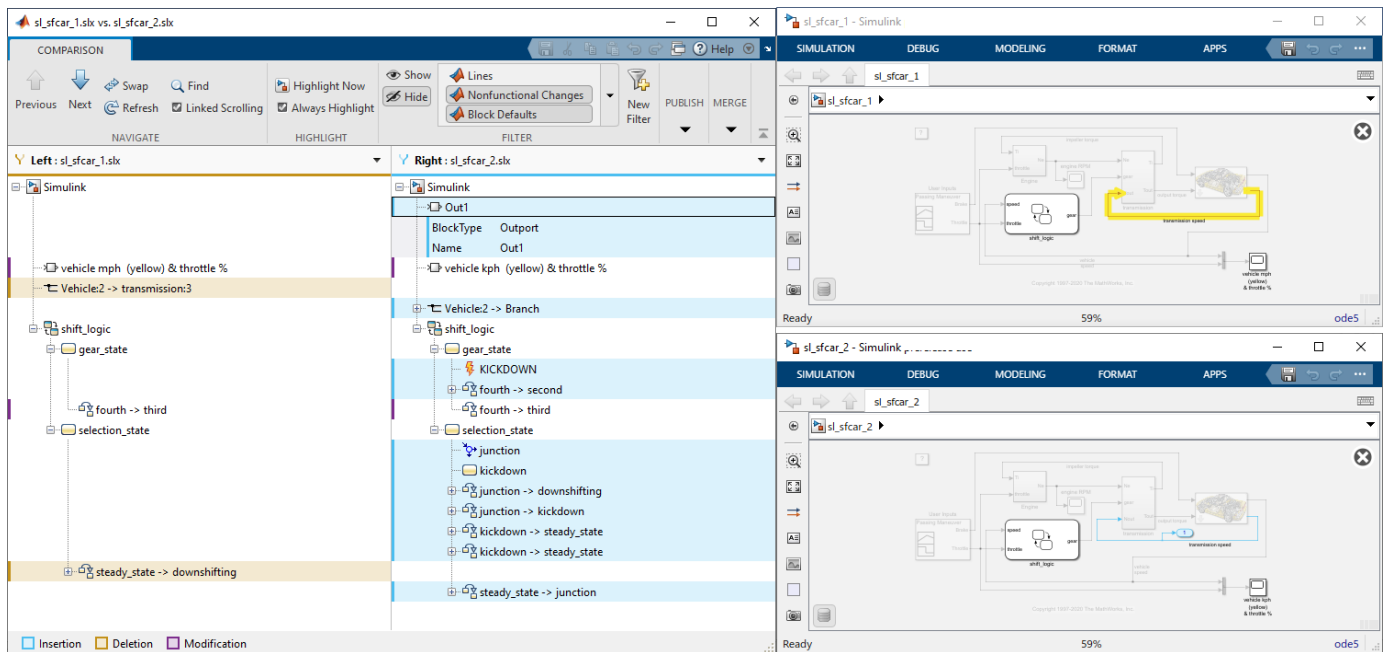
- Replaces the Subsystem block with a Variant Subsystem block, preserving ports and connections.
- Adds the original subsystem as a variant choice in the Variant Subsystem block.

- Preserves links to libraries. For linked subsystems, Simulink adds the linked subsystem as a variant choice.

Project and File Management

Model Comparison: Compare and merge models using rich highlighting

In R2021a, Simulink comparison and merge tools clearly highlight changes, conflicts, and merge choices for all model elements in the Simulink Editor. It is now easier to understand model changes and merge choices in the context of other, unchanged elements. Changes are highlighted with strong contrasting colors. Unchanged model elements are faded.



For examples with instructions, see [Compare and Merge Simulink Models](#) and [Resolve Conflicts with Simulink Three-Way Merge](#).

Model Comparison: Improved model comparison report

Starting in R2021a, comparison reports offer enhanced support for comparing large model files. You can more easily observe model changes by zooming on and searching pictures in HTML, Word, and PDF reports. To create printable reports, see [Export, Print, and Save Model Comparison Results](#).

Version History

To revert to the previous behavior and decrease the report file size on disk, you can reset the files from SVG to PNG format using `slxmlcomp.options`.

Project API: List all referenced projects of the current project

You can now use `listAllProjectReferences` to programmatically list all projects in the reference hierarchy of a specified project.

Project API: List impacted project files

You can now use `listImpactedFiles` to programmatically list all project files impacted by changes to a specified file.

Upgrade Advisor Check: Turn off file compression to optimize storage under source control

Starting in R2021a, Simulink provides a check to optimize SLX model files for storage under Git source control. This check is available in the model Upgrade Advisor and in the Upgrade Project tool. For more details, see [Check if SLX file compression is off](#).

Dependency Analyzer: Find required add-ons

Starting in R2021a, the Dependency Analyzer detects and lists required add-ons, including apps and toolboxes, for the whole project or for selected files. The Dependency Analyzer might not detect required support packages. For more details, see [Find Required Products and Add-Ons](#).

Functionality Being Removed or Changed

UpdateHistory will be removed

Warns

The `UpdateHistory` parameter will be removed in a future release. Scripts that use the `UpdateHistory` parameter continue to work.

Starting in R2021a, the **Model history** field and the **Prompt to update model history** option in the Model Properties dialog box have been removed. It is still possible to query the `ModifiedHistory` parameter, and enable the prompt to update behavior by setting the `UpdateHistory` parameter.

To prepare, resave models that use `UpdateHistory` in R2021a. This action removes the `UpdateHistory` parameter and fixes the warning thrown when you open the model.

Data Management

Share data objects with Simulink block library users by using a linked data dictionary

Previously, if the author of a block library defined data types — such as bus and enumeration types — to be used on block interfaces, a user of the block library had to recreate objects in the base workspace or in a data dictionary attached to the model. The user created these objects manually by following published guidelines or by using scripts that could automatically generate the objects. In R2021a, library authors can make data types available to users by storing them in a data dictionary that is attached to the library. Library users automatically gain access to the data contained in the dictionary when they drag a block from the library into their model.

For more information, see [Link Data Dictionary to Custom Libraries](#).

Model argument enhancements

In R2021a, these enhancements make it easier to use model arguments to provide instance-specific values for parameters in a model reference hierarchy.

- Configure a `Simulink.Parameter` object as a model argument without providing a default value. If you have specified `Data Type` and `Dimensions` for the model argument, you can compile and simulate the top model without specifying a value at the point of definition as long as you provide an explicit value somewhere in the parent model reference hierarchy.
- Visualize the compiled default value for a model argument. In a Model block that refers to a reusable model, if you do not specify an instance-specific value for a model argument, the argument uses the last value specified below it in the model hierarchy. In this case, the Model Data Editor displays `<from below>` for the argument value. In R2021a, after you update the model diagram, the Model Data Editor also displays the compiled value for the argument. To navigate to this default value from below, click the button with three vertical dots next to the compiled value and select `Navigate to default value from below`. The referenced model where the value was last specified opens in a new tab in the model canvas, and the Model Data Editor highlights the row that contains the block parameter.

For more information, see [Parameterize Instances of a Reusable Referenced Model](#).

Signal Editor user interface changes

Starting in R2021a, the Signal Editor user interface has these changes:

- Zoom now honors snap to grid and grid ticks.
- Move data now honors snap to grid and grid ticks.
- The Signal Editor user interface now supports importing and exporting from and to the base workspace and MAT files.
 - To import, use **Open > Import**. When importing MAT files, in the **Import** dialog box window, use the **Browse** button. When importing files from the base workspace, the Signal Editor automatically recognizes data formatted in Simulink loading and logging signal formats.
 - To export, use **Save > Export**.

Register Simulink.io.FileType objects from a different name space

Register `Simulink.io.FileType` objects from a different name space by implementing the `Simulink.io.PluggableNamespace` class.

To examine issues with the registration of these objects, use the `Simulink.io.getFileTypeDiagnostics` function.

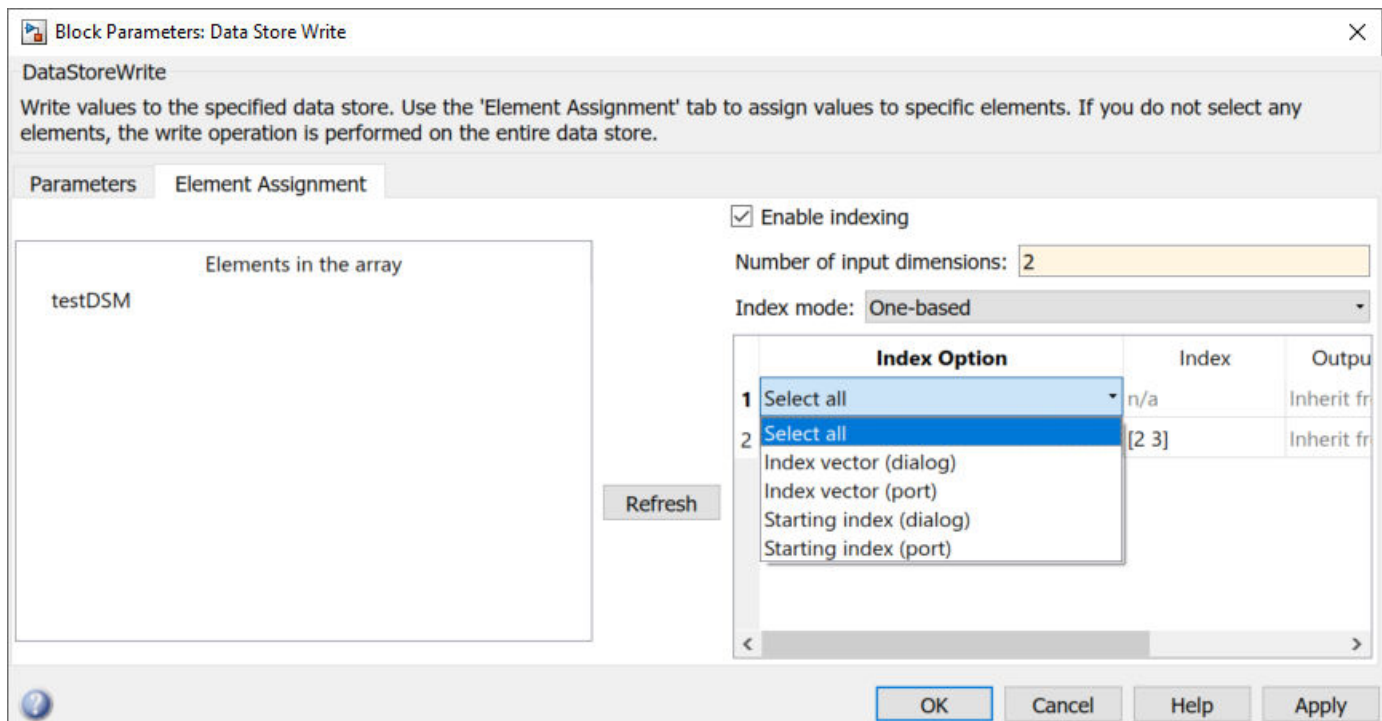
Block Enhancements

Data Stores: Access Data Store Memory elements dynamically via index input ports with Data Store Read/Write blocks

Data Store Read and Data Store Write blocks can now use dynamically generated indices to read and write a subset of the data stored in a data store. Previously, you had to use Selector and Assignment blocks in conjunction with Data Store Read and Data Store Write blocks for this purpose.

Code generated from models containing Data Store Read and Data Store Write blocks that use dynamically generated indices to access data store elements is more efficient in terms of memory and performance compared to using these blocks with Selector and Assignment blocks.

You can specify a subset of data to access along each dimension of the data store using five types of index options, including specifying an index vector or a starting index to be read from an input port to the block.



Highlighting to indicate related blocks

Similar to Goto and From blocks, you can now select any of these blocks to highlight blocks related to them:

- Inport blocks that have duplicates
- In Bus Element blocks that have duplicates
- Data Store Read, Data Store Write, and Data Store Memory blocks
- Simulink Function and Function Caller blocks

- State Reader, State Writer, Initialize Function, Reset Function, Terminate Function, and the corresponding state owner blocks
- Connection Label (Simscape) blocks

For example, selecting an In Bus Element block highlights any other In Bus Element blocks that select the same element from the input bus.

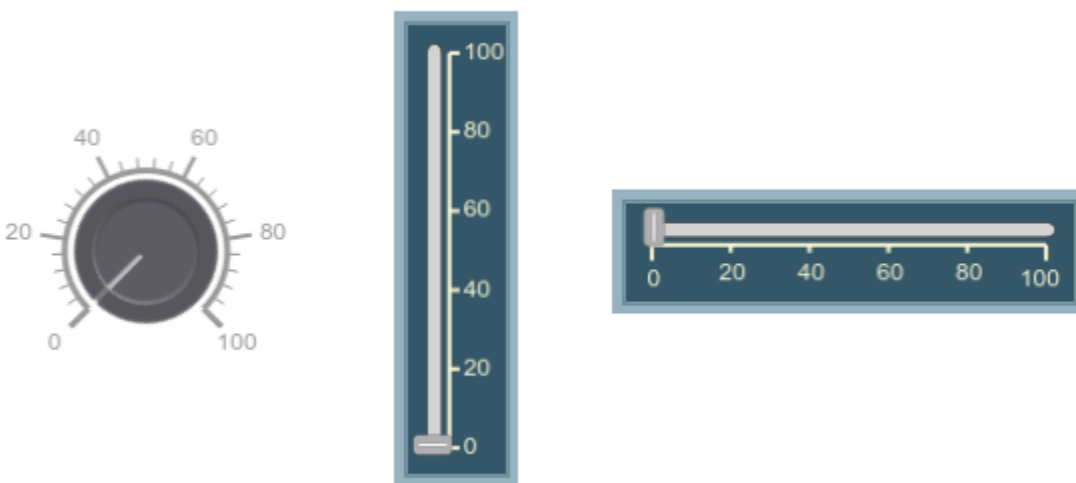


New Message Merge Block

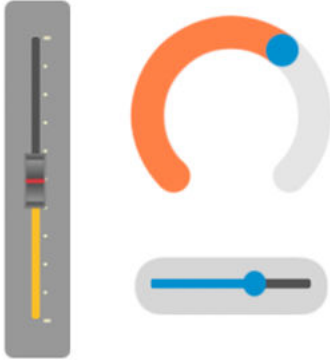
The Messages & Events library has a new block, Message Merge. The block combines multiple message lines into a single message line. For more information, see Merge Message Lines Using a Message Merge Block.

New customizable Knob, Vertical Slider, and Horizontal Slider blocks

Three new blocks in the Customizable Blocks sublibrary of the Dashboard library allow you to create intuitive and photo-realistic dashboards for interacting with your models. The Knob, Vertical Slider, and Horizontal Slider blocks provide the same configuration options as the Knob and Slider blocks in the Dashboard library, with additional options to customize the appearance. The blocks provide a similar look and feel to other blocks in the Customizable Blocks library, so you can use them as-is in your model. You can also customize each block with your own background image, foreground image, handle or slider image, and scale appearance.



For example, you can customize the Knob block to look like a thermostat, and you can customize the Vertical Slider block to look like a fader on an audio mixer.



Choose from a set of icons or specify a custom icon for the Lamp block

The Lamp block allows you to choose an icon that defines how the block appears in the model. Icon options that ship with the block include basic shapes and industry-standard icons for automotive and wireless applications. You can also upload a custom SVG file to use your own icon.

The Dashboard library includes sublibraries with a preconfigured Lamp block for each icon:

- Automotive Indicator Lamps
- Basic Shape Icons
- Wireless Icons

Dashboard gauge blocks support foreground, background, and font color

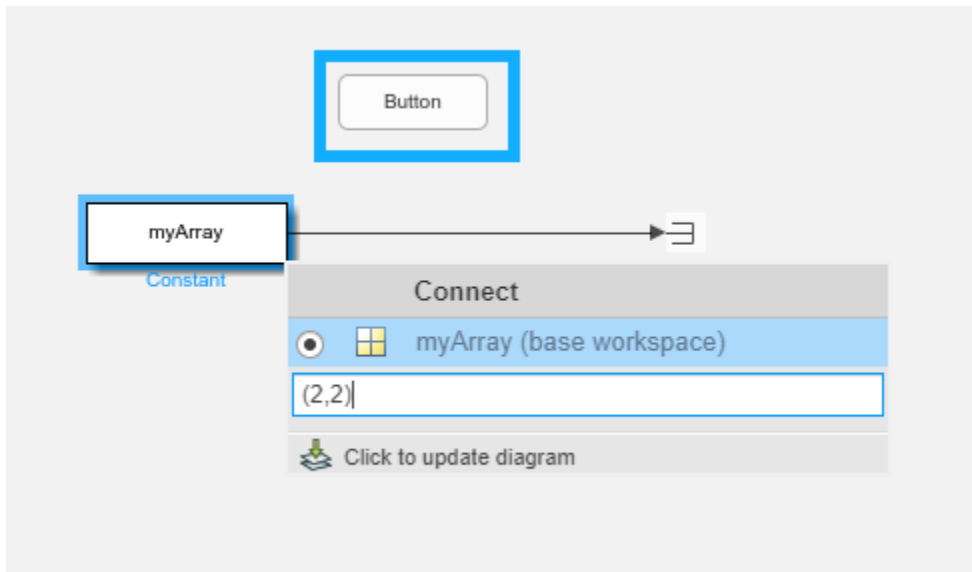
You can change the background, foreground, and font color for these blocks:

- Gauge
- Half Gauge
- Quarter Gauge
- Linear Gauge
- Circular Gauge
- Horizontal Gauge
- Vertical Gauge

Tune matrix and structure elements using dashboard blocks

You can connect a dashboard block to a scalar element of a matrix or structure that defines the value of a tunable block parameter. When you connect a dashboard block to a tunable parameter, you select one or more blocks and select the variable or parameter you want to connect. When you select a

variable or parameter with a nonscalar value to connect, you specify the element you want to tune using the text box.



Dashboard Scope block enhancements

The Dashboard Scope block has several enhancements. You can now:

- Choose to update the display by wrapping the plot or by scrolling.
- Normalize the y-axis.
- Show or hide horizontal and vertical grid lines.
- Control the visibility and position of the legend.
- Show or hide the plot border and data markers.
- Specify colors for the foreground, background, and font.

Change dashboard block orientation and scale direction

Several dashboard blocks support rotating the block using the **Rotate 90 clockwise** and **Rotate 90 counterclockwise** buttons on the **Format** tab:

- Half Gauge
- Quarter Gauge
- Linear Gauge
- Rocker Switch
- Slider Switch
- Toggle Switch
- Slider
- MultiStateImage

Several dashboard blocks have a new parameter that allows you to control the direction of the scale:

- Gauge
- Half Gauge
- Quarter Gauge

From Workspace block supports loading data for array of bus

You can use the From Workspace block to load data for an array of buses into a top model for normal or accelerator mode simulations and into a referenced model for normal mode simulations.

Sqrt and Dot Product blocks support half-precision data type

The Sqrt and Dot Product blocks now support the half-precision data type.

Trigonometric Function block disables input range checking for acos and asin functions

The Trigonometric Function block now disables input range checking for `acos` and `asin` functions with the **Remove protection against out-of-range input** check box. Disabling input checking prevents redundancy if the input is guaranteed to be in range.

Trigonometric Function block supports lookup table algorithms for multiple functions

The Trigonometric Function block now supports lookup table-based approximation for the functions: `sin`, `cos`, `sincos`, `cos + jsin`, and `atan2`. The Trigonometric Function has been updated accordingly with this function:

- For the **Approximation method**, you can now select a Lookup option. When you select Lookup for the **Approximation method**, these parameters are enabled:
 - **Angle unit**
 - **Number of data points**

1-D Lookup Table block supports full precision interpolation algorithm for linear interpolation

1-D Lookup Table block now supports full precision fixed-point algorithm for linear interpolation with the **Apply full precision fixed-point algorithm when possible** check box. This check box enables full precision fixed-point algorithm for linear interpolation lookup. This algorithm generally achieves better precision for hardware-efficient fixed-point rounding modes.

Switch, MinMax, and Multiport Switch blocks have new output data type option

The Switch and MinMax blocks have the new output data type option **Inherit: Same as first input**. The Multiport Switch block has the new output data type option **Inherit: Same as first data input**. Use these options to use the data type of the first data input port of the block.

Math Function block reciprocal function can now use Newton-Raphson method

The Math Function block reciprocal function can now use the Newton-Raphson algorithm method to calculate the reciprocal.

To use the Newton-Raphson algorithm method, from the **Algorithm method** parameter, select Newton-Raphson. To use the original reciprocal algorithm, select Exact.

Tunability of Rate Limiter block parameters

The Rate Limiter block now supports tunability of the **Rising slew rate** and **Falling slew rate** parameters for simulation and code generation.

Simulink.LookupTable and Simulink.Breakpoint classes are able to be subclassed

You can now subclass `Simulink.LookupTable` and `Simulink.Breakpoint` classes.

FMU block is now more customizable

The FMU block is now more customizable, enabling you to:


- Hide or display block input and output ports.
- Customize the FMU block interface for different model needs.
- Restore to default block layout.
- Set the start value for input or local variables.
- Edit the start values of local variables.
- Expose local variables as output signals.

Signal Editor block now supports array of buses

The Signal Editor block now supports array of buses.

Signal Builder block menu and toolbar options to export data to Signal Editor block

The Signal Builder block now exports signal data to Signal Editor block format with these options:

- From the Signal Builder block, select **File > Export > To Signal Editor block**.
- In the Signal Builder toolstrip, click  .

For more information, see Exporting Signal Group Data.

MATLAB System Block now supports message input and output

In R2021a, the MATLAB System block supports sending and receiving messages. You can use the new System object API `getInterfaceImpl` to specify input and output message ports and `stepImpl` to compute message payload.

See [Process Message Payload Using MATLAB System Block and `getInterfaceImpl`](#) for more information.

New upgrade advisor check to integrate with `sysobjupdate`

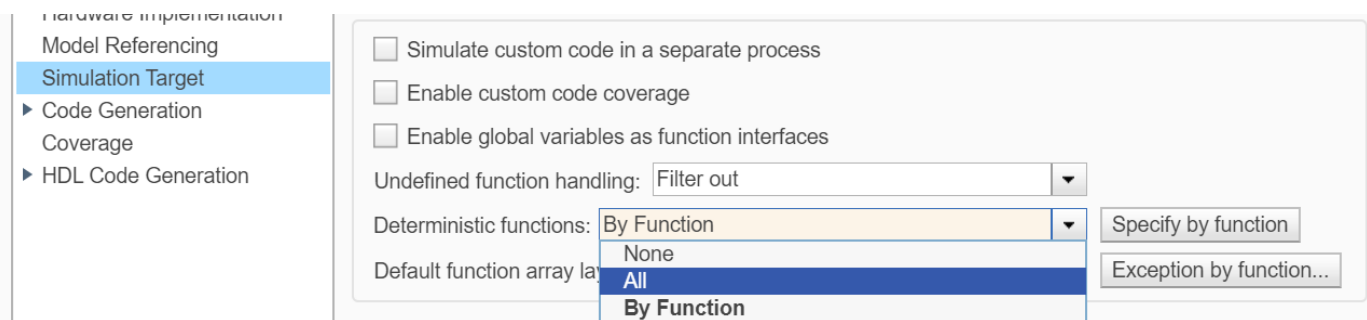
In R2021a, the Simulink Upgrade Advisor can scan all MATLAB System blocks and update user-authored System objects to the latest syntax using `sysobjupdate`. A link to the update results display in the Report section of the Upgrade Advisor.

C Function and C Caller blocks in For Each subsystem, with continuous time, and optimized in conditional input branch execution

C Function and C Caller blocks can now be used in For Each subsystems and in models with continuous sample time, and the blocks can be optimized for use in conditional input branch execution.

When a block is optimized for use in conditional input branch execution, it is executed only if it is in the active branch of a Switch or Multiport Switch block, both in simulation and in generated code.

For the C Function and C Caller blocks to be used with these features, any custom code functions called by the blocks must be deterministic, that is, they must always produce the same outputs for the same inputs. The new **Deterministic functions** parameter allows you tell Simulink which of your custom code functions are deterministic.



Control inherited block output data type for half-precision

You can now control the output data type of blocks that support the half-precision data type and that support the 'Inherit via Internal Rule' block output data type. To set the data type rule, go to **Configuration Parameters > Math and Data Types > Inherit floating-point output type smaller than single precision**. For more information, see [Inherit floating-point output type smaller than single precision](#).

Tunable mask enumeration in generated code

From 2021a onward, you can:

- Reference an external enumeration class for a mask popup parameter.
- Associate numeric values to a mask popup parameter using the external enumeration class.
- Associate a workspace variable with a mask popup parameter.
- Tune a popup parameter during simulation and in generated code.

For more information, see [Tune Mask Popup Parameters by Referencing an External Enumeration File](#)

Show or hide columns in custom table

Prior to R2021a, you could enable or disable a column in a custom table. From 2021a onward, you can show or hide a column using the **Visible** property.

Sparse Matrix Support for State-Space Block

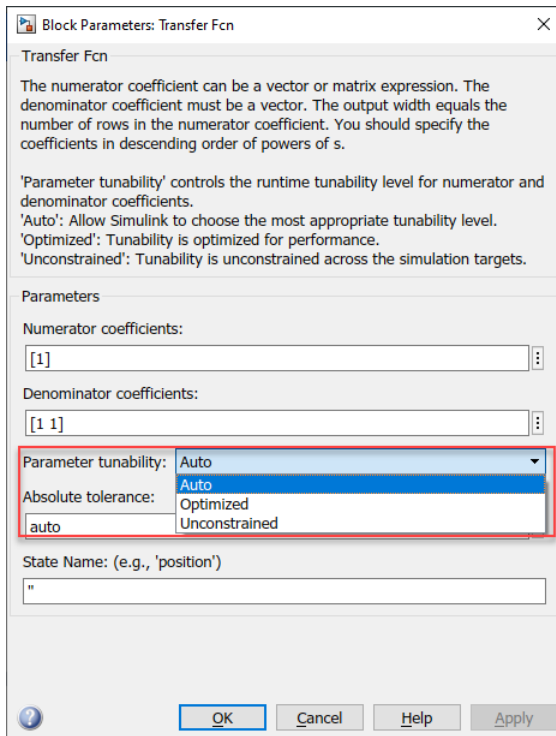
You can now specify the **A**, **B**, **C**, and **D** parameters of the **State-Space** block as sparse matrices. For more information on creating sparse matrices, see [sparse](#)

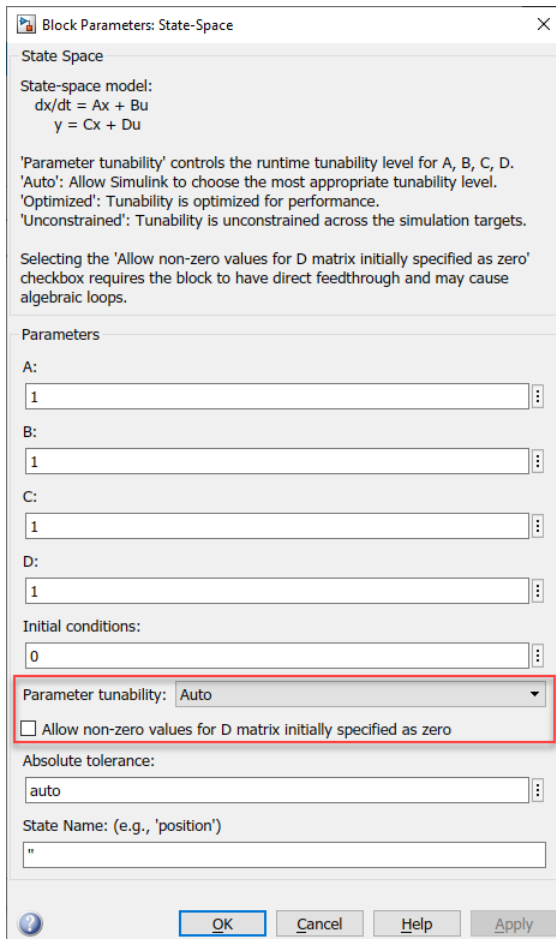
Parameter Tunability Enhancements for Continuous Blocks

You can now consistently tune several parameters of the State-Space and Transfer Function blocks in Accelerator and Rapid Accelerator simulation modes as well as the Simulink Compiler.

Set the **Parameter Tunability** setting in the State-Space and Transfer Fcn blocks to **Unconstrained** to allow full tunability of the parameter matrices or coefficients, respectively, in all simulation modes and in the Simulink Compiler.

When set to **Optimized**, Simulink optimizes the tunability of the block parameters for performance.





To tune the **D** matrix of the State-Space block when $D=0$, enable the **Allow non-zero values for D matrix initially specified as zero** parameter.

Note Enabling this parameter causes direct-feedthrough to be enabled for the State-Space block.

Functionality being removed or changed

The HDL Reciprocal block is not recommended

Still runs

The HDL Reciprocal block is not recommended. This block has been moved from the HDL Coder/Math Operations library to the Simulink/Simulink Extras/Additional Math library. Use the `reciprocal` function with the Newton-Raphson method in the Math Function block instead. Existing models continue to work.

Circular Gauge, Horizontal Gauge, and Vertical Gauge blocks move to Customizable Blocks library

Behavior change

In prior releases, the Dashboard library contained the Circular Gauge, Horizontal Gauge, and Vertical Gauge blocks. In R2021a, these blocks are now in the Customizable Blocks sublibrary within the

Dashboard library. The Customizable Blocks sublibrary also contains the new customizable Knob, Horizontal Slider, and Vertical Slider blocks.

Update scripts that added the Circular Gauge, Horizontal Gauge, or Vertical Gauge blocks to a model programmatically to add the blocks from the `simulink_hmi_customizable_blocks` library.

System Object Authoring

Behavior change

The following APIs will not be supported in a future release. To update your System object to the latest pass, run the `sysobjupdate` function:

- `isInputSizeLockedImpl` will be converted to `isInputSizeMutableImpl` (return value is negated).
- `isInputComplexityLockedImpl` will be converted to `isInputComplexityMutableImpl` (return value is negated).
- `isOutputComplexityLockedImpl` will be removed.
- `isOutputSizeLockedImpl` will be removed.
- `processInputSizeChangeImpl` will be converted to `processInputSpecificationChangeImpl`.

Connection to Hardware

Deploy deep learning applications on Android devices

The Simulink Support Package for Android Devices now enables you to generate code for deep learning applications using the Deep Learning Toolbox™ and deploy the application on Android devices. The deep learning applications continue to run on the Android device even when you disconnect the device from the host computer.

Create interactive display of Dashboard blocks on Android devices

These Dashboard blocks are enhanced to generate code on Android devices:

- Display
- Edit
- Push Button

Read proximity, gesture, ambient light, and color data using APDS9960 I2C sensor

The Simulink Support Package for Arduino Hardware now supports interfacing the APDS9960 I2C sensor with Arduino hardware. The APDS9960 Sensor block, which has been newly added to the support package detects proximity, provides directional data to decode gestures, and measures ambient light and color (RGB) data.

Support added for LCD and TFT display using Display Configuration app

The Simulink Support Package for Arduino Hardware now provides code generation support for:

- Dashboard text display on LCD and TFT display types
- Dashboard gauge display on the TFT display

You can use the Display Configuration app to configure the Simulink model to work with any graphical or character display of your choice. The app has been added in the **Simulink model > Configuration Parameters > Hardware Implementation > Target hardware resources** section.

Profile real-time execution of generated code on Arduino hardware

The Simulink Support Package for Arduino Hardware now provides real-time execution profiling for the Simulink models deployed on ARM architecture-based Arduino boards. You can now check if the generated code meets your real-time performance requirements to improve the use of available hardware resources.

Support added for Arduino Engineering Kit Rev2

The Simulink Support Package for Arduino Hardware now supports Arduino Engineering Kit Rev2. The kit provides a hands-on learning tool that demonstrates core engineering and programming

concepts. It also includes interfaces and physical components you need to build projects. You can also use this kit to create your own projects while using MATLAB and Simulink.

Support added for Arduino Nano 33 BLE Sense board

The Simulink Support Package for Arduino Hardware now supports deploying Simulink models on the Arduino Nano 33 BLE Sense boards. You can use all the blocks from the support package library with the Arduino Nano 33 BLE Sense board.

Measure rotation of motor interfaced with rotary encoder

The Simulink Support Package for Raspberry Pi Hardware now supports measuring incremental position and direction of a motor interfaced with a rotary encoder. The Encoder block, which has been newly added to the support package, provide this functionality.

New functionalities added to Raspberry Pi Resource Monitor app

The Raspberry Pi Resource Monitor App from the Simulink Support Package for Raspberry Pi Hardware has been improved to:

- Display peripherals used in a MATLAB or Simulink application deployed on the Raspberry Pi hardware
- Enable or disable peripherals
- Check for missing libraries and packages
- Display all processes currently running on the Raspberry Pi hardware

Support for MODBUS TCP/IP data communication protocol

The Simulink Support Package for Raspberry Pi Hardware now supports MODBUS[®] communication between master and slave devices connected over the TCP/IP network.

This table lists the new blocks supporting MODBUS data communication.

Block	Usage
Modbus TCP/IP Master Read	Master device requests data from the register(s) of the slave device
Modbus TCP/IP Master Write	Master device writes data to the register(s) of the slave device
Modbus TCP/IP Slave Read	Slave device requests data from its own registers
Modbus TCP/IP Slave Write	Slave device writes data to its own registers

New MODBUS properties have been added to all four blocks in the **Configuration Parameters > Hardware Implementation > Target hardware resources** section. These properties allow you to configure the mode of operation and modify master device and slave register settings.

Support for adding multiple SDL Display blocks to Simulink model deployed on Raspberry Pi

The Simulink Support Package for Raspberry Pi Hardware now supports adding multiple SDL Display blocks to the Simulink model deployed on your Raspberry Pi hardware. In the previous releases, you could add only one SDL Display block to the Simulink model.

Trigger downstream function-call subsystem

The Simulink Support Package for Raspberry Pi Hardware now supports triggering the downstream function-call subsystem from an interrupt service routine (ISR). The External Interrupt block, which has been newly added to the support package, provides this functionality. Invoking the function-call subsystem executes the output methods of the blocks within the subsystem in a sorted order.

MATLAB Function Blocks

Programmatically access MATLAB function reports

You can now access MATLAB function reports by calling these functions on MATLABFunctionConfiguration objects:

- `openReport` opens the MATLAB function report for the block.
- `closeReport` closes the MATLAB function report for the block.
- `getReport` returns a MATLABFunctionReport object for the block. You can query report information from this object by accessing its 'Functions' property, which is an array of `coder.Function` objects. See `coder.Function` Properties (MATLAB Coder).

For more information, see MATLAB Function Reports.

64-bit integer type support for parameters

Parameters in MATLAB Function blocks can now be set as 64-bit integer data. For more information, see Type Function Arguments.

Half-precision data type support for MATLAB Function blocks

MATLAB Function blocks can now use half-precision type data. A half-precision data type occupies 16 bits of memory, but its floating-point representation enables it to handle wider dynamic ranges than integer or fixed-point data types of the same size. For more information, see Floating-Point Numbers (Fixed-Point Designer).

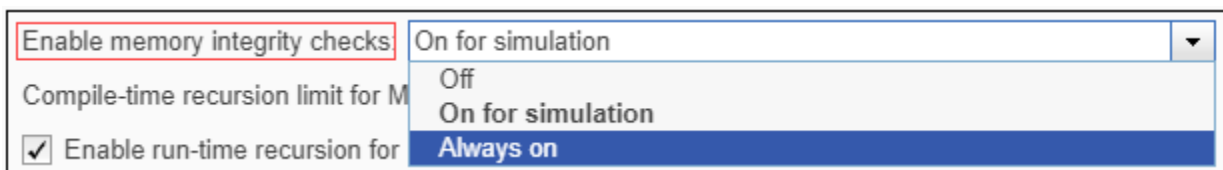
Error reporting from MATLAB Function blocks in rapid accelerator simulation mode

Starting in R2021a, a rapid accelerator simulation reports errors that might occur while simulating a MATLAB Function block. There is a new option to enable the diagnostics in rapid accelerator mode. If you have large and complex models, this error reporting can shorten the length of your debugging sessions.

To enable reporting of diagnostics from a MATLAB Function block during the rapid accelerator mode of simulation, at the MATLAB command prompt, enter:

```
set_param(ModelName, 'SimIntegrity', 'alwaysOn');
```

Alternatively, you can enable memory integrity checks for all simulation modes by selecting the **Always on** option. In the **Configuration Parameters** dialog box, select **Simulation Target > Advanced parameters** section and click on **Always on** from the drop-down option under **Enable memory integrity checks**.



Expanded code generation for categorical arrays

In R2021a, code generation supports more MATLAB toolbox functions when you use categorical arrays.

The supported functions for categorical arrays are:

- `issorted`
- `issortedrows`
- `sort`
- `sortrows`

For more information, see [Code Generation for Categorical Arrays \(Simulink\)](#).

Expanded code generation for tables and timetables

In R2021a, code generation supports more capabilities and MATLAB toolbox functions when you use tables and timetables.

The supported functions for tables are:

- `issortedrows`
- `join`
- `renamevars`
- `rows2vars`
- `sortrows`
- `splitvars`
- `stack`
- `unstack`
- `varfun`

The supported functions for timetables are:

- `issorted`
- `issortedrows`
- `join`
- `renamevars`
- `sortrows`
- `splitvars`
- `stack`
- `unstack`
- `varfun`

For more information, see [Code Generation for Tables \(Simulink\)](#) and [Code Generation for Timetables \(Simulink\)](#).

Code generation for more MATLAB functions

- `convhulln`
- `deLaunayn`
- `logm`
- `sylvester`
- `voronoin`

Code generation for more toolbox functions

In R2021a, you can generate code for many additional toolbox functions and objects. For a list of all functions and objects that are supported for code generation, see:

- [Functions and Objects Supported for C/C++ Code Generation \(Category List\)](#)
- [Functions and Objects Supported for C/C++ Code Generation \(Alphabetical List\)](#)

These are links to the release notes of some toolboxes that added code generation support in R2021a:

Computer Vision Toolbox

See “Extended Capability: Perform GPU and C/C++ code generation” (Computer Vision Toolbox)

Image Processing Toolbox

See “C Code Generation: Generate code from the `adaptHisteq` function using MATLAB Coder” (Image Processing Toolbox)

Signal Processing Toolbox

See “C/C++ Code Generation Support: Code generation for filtering, signal modeling, spectral analysis, and statistics” (Signal Processing Toolbox).

Statistics and Machine Learning Toolbox

- See “Generate C/C++ code for performing incremental learning using linear regression or binary classification model functions (requires MATLAB Coder)” (Statistics and Machine Learning Toolbox)
- See “Generate C/C++ code for prediction by using a machine learning model with heterogeneous data (requires MATLAB Coder)” (Statistics and Machine Learning Toolbox)

Wavelet Toolbox

See “C/C++ Code Generation: Automatically generate code for wavelet functions” (Wavelet Toolbox)

Modeling Guidelines

Use modeling guidelines to verify compliance of your models

In R2021a, these high-integrity modeling guidelines are added:

- hisl_0067: Protect against divide-by-zero calculations
- hisl_0071: Configuration Parameters > Hardware Implementation > Inconsistent hardware implementation settings

This table identifies modeling guidelines that were modified in R2021a.

Modeling Guideline	Rationale
hisf_0004: Protect against recursive function calls to improve code compliance	<p>The title of the guideline is updated from "Usage of recursive behavior" to "Protect against recursive function calls to improve code compliance".</p> <p>The description is updated to instruct you not to call functions recursively. Further explanation is provided in the guideline Notes.</p> <p>The guideline now specifies prerequisites hisf_0011, hisl_0311, and hisl_0060.</p>
hisl_0311: Configuration Parameters > Diagnostics > Stateflow	<p>Added these configuration parameters, which are to be set to <code>error</code>:</p> <ul style="list-style-type: none"> • Read-before-write to output in Moore chart • Absolute time temporal value shorter than sampling period • Self transition on leaf state • Execute-at-Initialization disabled in presence of input events • Use of machine-parented data instead of Data Store Memory
hisl_0016: Usage of blocks that compute relational operators	<p>Sign block is added to the list of blocks for which you must avoid comparisons by using the <code>==</code> or <code>~=</code> operator on floating-point data types.</p>
na_0010: Usage of vector and bus signals	<p>Removed NA-MAAB recommendations for application of sub IDs to your model. The working group voted to remove or not recommend the MAB v5.0 version of the guideline.</p>

Version History

This table identifies changes to the modelling guidelines that may impact compatibility, including removed guidelines or conditions.

Modeling Guideline	Corresponding Model Advisor Check	Rationale
hisl_0051: Configuration Parameters > Code Generation > Optimization > Loop unrolling threshold	Check safety-related optimization settings for Loop unrolling threshold	Removed guideline hisl_0051. The guideline suggests setting the loop rolling threshold to 2 or higher. A loop cannot exist unless the data type is nonscalar. Nonscalar types have a dimension of 2 or higher, therefore, the rule is irrelevant.
hisf_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA C:2012 compliance	N/A	Removed guideline hisf_0213. The guideline is replaced by hisl_0067: Protect against divide-by-zero calculations.
hisl_0005: Usage of Product blocks	N/A	Removed condition A from the guideline because divide-by-zero is covered by hisl_0067: Protect against divide-by-zero calculations.

R2020b

Version: 10.2

New Features

Bug Fixes

Version History




Simulink Editor

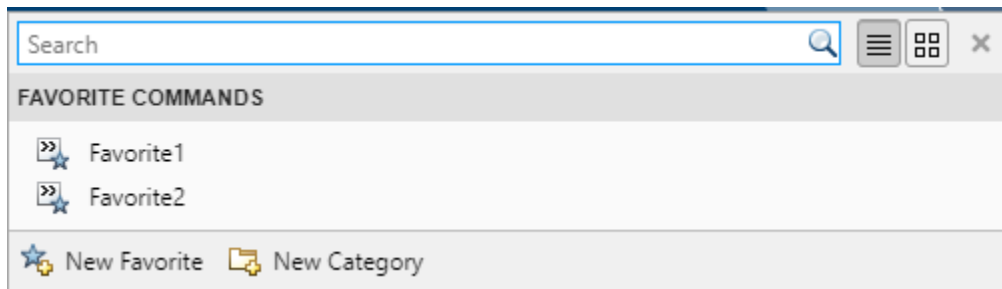
Simulink Online: Use Simulink through your web browser


Simulink Online was introduced in an R2020a web update. For more information, see [What is Simulink Online ?](#).

Enhancements to the Simulink quick access toolbar

Starting in R2020b, you can:

- Add toolstrip items to the Simulink quick access toolbar by right-clicking the item in the toolstrip and selecting **Add to Quick Access Toolbar**. To remove toolstrip items that you add, right-click the corresponding item and select **Remove from Quick Access Toolbar**.
- Search for actions by clicking the **Search Toolstrip** button  in the quick access toolbar. In the search box that appears, start typing the name or description of the toolstrip item, then select it from the menu.
- Drag items in the quick access toolbar to change their position.
- Show item labels in the quick access toolbar by right-clicking the item icon and selecting **Show Label**.
- Show or hide common items in the quick access toolbar by clicking the  button and selecting or clearing the corresponding check boxes. Common items now include **Select All**, **Search Toolstrip**, **Find**, **MATLAB Desktop**, **Exit MATLAB**, and **Favorite Commands**.
- Manage a group of your favorite MATLAB commands from the Simulink quick access toolbar. The **Favorites** button  opens a dialog box that allows you to add, edit, organize, run, and delete your favorite commands. The list of favorite commands in the Simulink quick access toolbar is independent of the list of favorite commands in the MATLAB quick access toolbar.




- Restore default settings for the quick access toolbar by right-clicking an item and selecting **Restore Defaults** or by clicking the  button and selecting **Restore Defaults**.

For more information, see [Access Frequently Used Features and Commands in Simulink](#).

Keyboard shortcuts to select and move objects in a diagram

Starting in R2020b, you can use the keyboard to select any combination of blocks and annotations in a diagram. Along with other keyboard shortcuts, you can now perform many editing tasks using only the keyboard.

Task	Keyboard Action
Select block or annotation	Arrow keys (M to toggle between moving selection and moving objects)
Select multiple adjacent blocks and annotations	Shift + arrow keys
Select multiple non-adjacent blocks and annotations	Windows, Linux — Ctrl + arrow keys to change the current object, Ctrl + spacebar to add the current object to the selection. macOS — command + arrow keys to change the current object, command + shift + spacebar or shift + spacebar to add the current object to the selection.
Move selected blocks and annotations	Arrow keys (M to toggle between moving selection and moving objects)

For additional keyboard shortcuts, see Keyboard Shortcuts and Mouse Actions for Simulink Modeling. You can now also open this page from a model: use **Shift** + **?**, or click the **Keyboard Shortcuts** button  on the navigation bar.

Drag or paste text into Simulink Editor canvas to create annotations

To create text annotations, you can now:

- Drag text from another application to the Simulink Editor canvas.
- Paste text from the clipboard. Right-click the canvas and select **Paste**, or use the **Ctrl** + **V** keyboard shortcut.

To accommodate long text annotations, Simulink now wraps text across multiple lines.

For more information on annotations, see Describe Models Using Notes and Annotations.

Optionally resize blocks to fit parameter values

Blocks such as Constant and Gain blocks display a parameter value only when it fits on the block icon.



After you edit the parameter value in the Simulink Editor, you now receive an interactive cue that you can click to resize the block. The size of the cue corresponds to the size the block must be to display the parameter value.

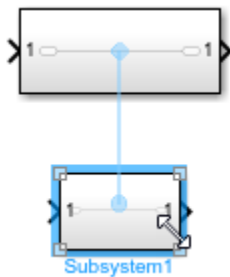


You can also use menu options to resize blocks so that they display the full parameter value.

- In the Simulink Toolstrip, on the **Format** tab, select **Fit to Content**.
- In the Simulink Editor, on the action bar that appears when you select a block and pause on the ellipsis, select **Fit to Content**.
- Right-click on the block and select **Arrange > Fit to Content**.

Align centers while resizing blocks and annotations

While resizing blocks and annotations, smart guides now help you align their centers.



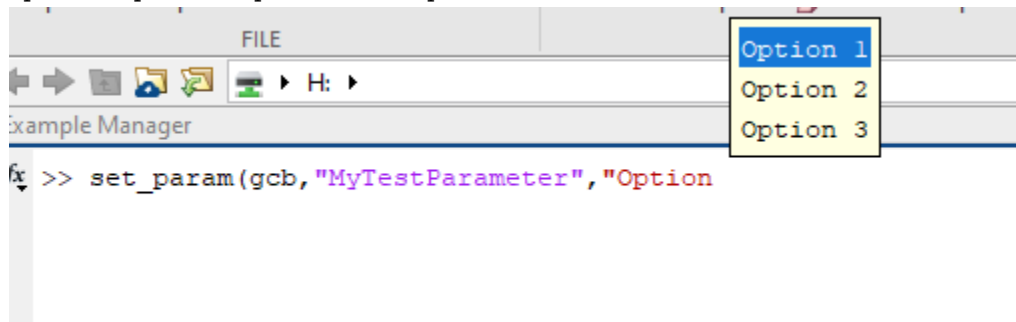
Use half and string data types in the mask datatypestr parameter

From R2020b, the mask datatypestr parameter supports half precision numeric object and string data types.

Display auto-complete options for mask parameter values while using set_param

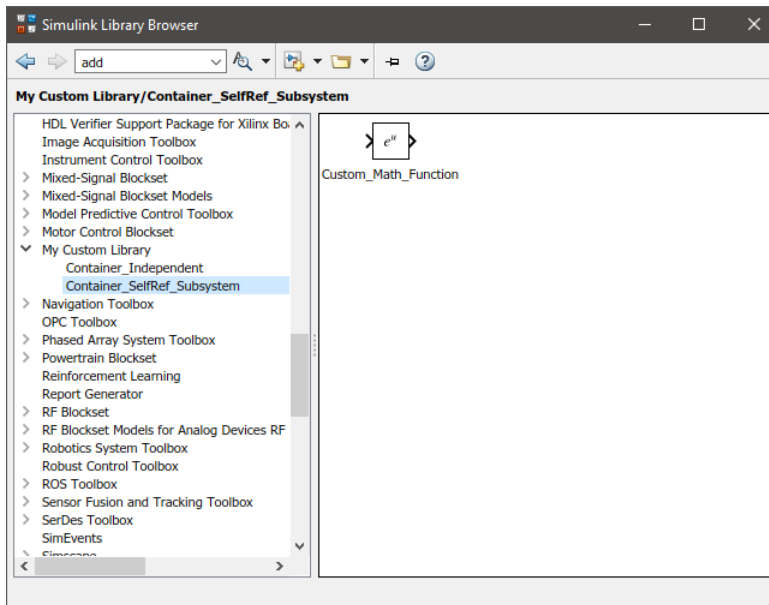
From R2020b, while using the `set_param` function, when you type a mask parameter name and press the Tab key, MATLAB displays auto-complete options for values of that parameter.

For example, consider a mask on a block 'MyBlock' and a popup parameter named 'MyTestParameter' on the mask. When you press the Tab key you will see three auto-complete options: Option1 , Option 2, and Option 3.



Refer between containers in the same library

In R2020b, you can refer to a container in a library from another container in the same library. For example, consider a custom library called `MyCustomLibrary` with two containers in it called `Container_Independent` and `Container_SelfRef_Subsystem`.



The `Container_SelfRef_subsystem` is referencing the content of `Container_Independent` using an `openFcn` callback.

Improved performance for the `add_block` function

The `add_block` function shows improved performance. For example, this code is about 8x faster than in the previous release:

```
function timingTest
    open_system(new_system('test_model'));
    add_block('built-in/SubSystem', 'test_model/Subsystem');

    tic
    for i=1:4000
        add_block('built-in/Product', sprintf('test_model/Subsystem/Block%d', i));
    end
    toc

    bdclose all
end
```

The approximate execution times are:

R2020a: 16.7 s

R2020b: 2.1 s

The code was timed on a Windows 10, Intel Xeon® CPU E5-1650 v4 @ 3.60GHz (12 CPUs), ~3.6GHz system by calling the function `timingTest`.

Function to check if Simulink is started

To check if Simulink is started, use the `isSimulinkStarted` function.

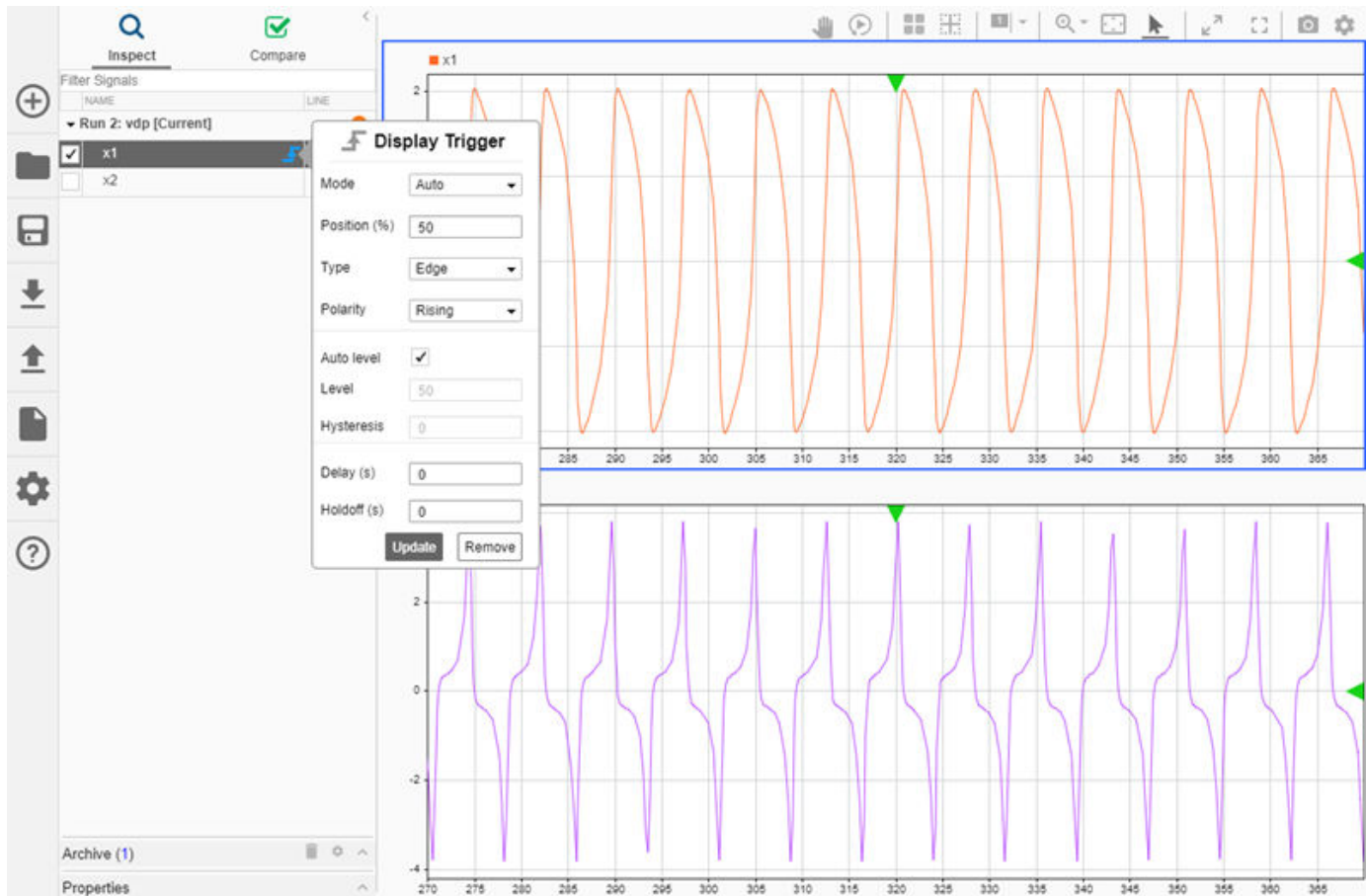
Simulation Analysis and Performance

FMU Export: MultiThreadCoSim renamed to MultithreadedSim

The `MultiThreadCoSim` parameter has been renamed to `MultithreadedSim`. Existing scripts continue to work. However, update all instances of `MultiThreadCoSim` to `MultithreadedSim`.

Streaming enhancements and triggers in the Simulation Data Inspector

Triggers and new time axis settings allow you to control the way streaming data is displayed in the Simulation Data Inspector. You can add a trigger to a signal to use characteristics of that signal to determine when to update the plot. You can only use one signal as a trigger, and that trigger controls when all subplots in the Simulation Data Inspector update, whether the trigger signal is plotted or not.



When you want to view streaming data without using a trigger, new settings give you better control over how and when the display updates. First, you can specify a value for the **Time span** setting in the **Time Plot Visualization Settings** that sets the range of the t -axis. When streaming data has samples for time values beyond the maximum time, the display scrolls or wraps, maintaining the specified time span.

If you want to pause the display to inspect a signal, you can use the **Freeze display** button to stop the plot from updating until you press the **Freeze display** button again. When you unfreeze the display, the plot updates to show the most recently streamed samples.

For more information about using triggers in the Simulation Data Inspector, see [Control Display of Streaming Data Using Triggers](#).

Simulation Data Inspector enhancements for importing Microsoft Excel files

The Simulation Data Inspector supports additional signal types and metadata for importing data from a Microsoft Excel file. With the enhancements, you can import Microsoft Excel files you use with Simulink Test for analysis in the Simulation Data Inspector.

In R2020b, you can import signals with fixed-point data types, aliased data types, and data types defined by a `Simulink.Bus` object from a Microsoft Excel file. You can also import function-call signals and simulation parameter values.

Microsoft Excel files you import into the Simulation Data Inspector can now include this signal metadata for use in comparisons:

- Relative tolerance
- Absolute tolerance
- Time tolerance
- Leading tolerance (only visible in Simulink Test)
- Lagging tolerance (only visible in Simulink Test)
- Synchronization method

For details, see [Microsoft Excel Import and Export Format](#).

Create a custom data reader to import data into the Simulation Data Inspector

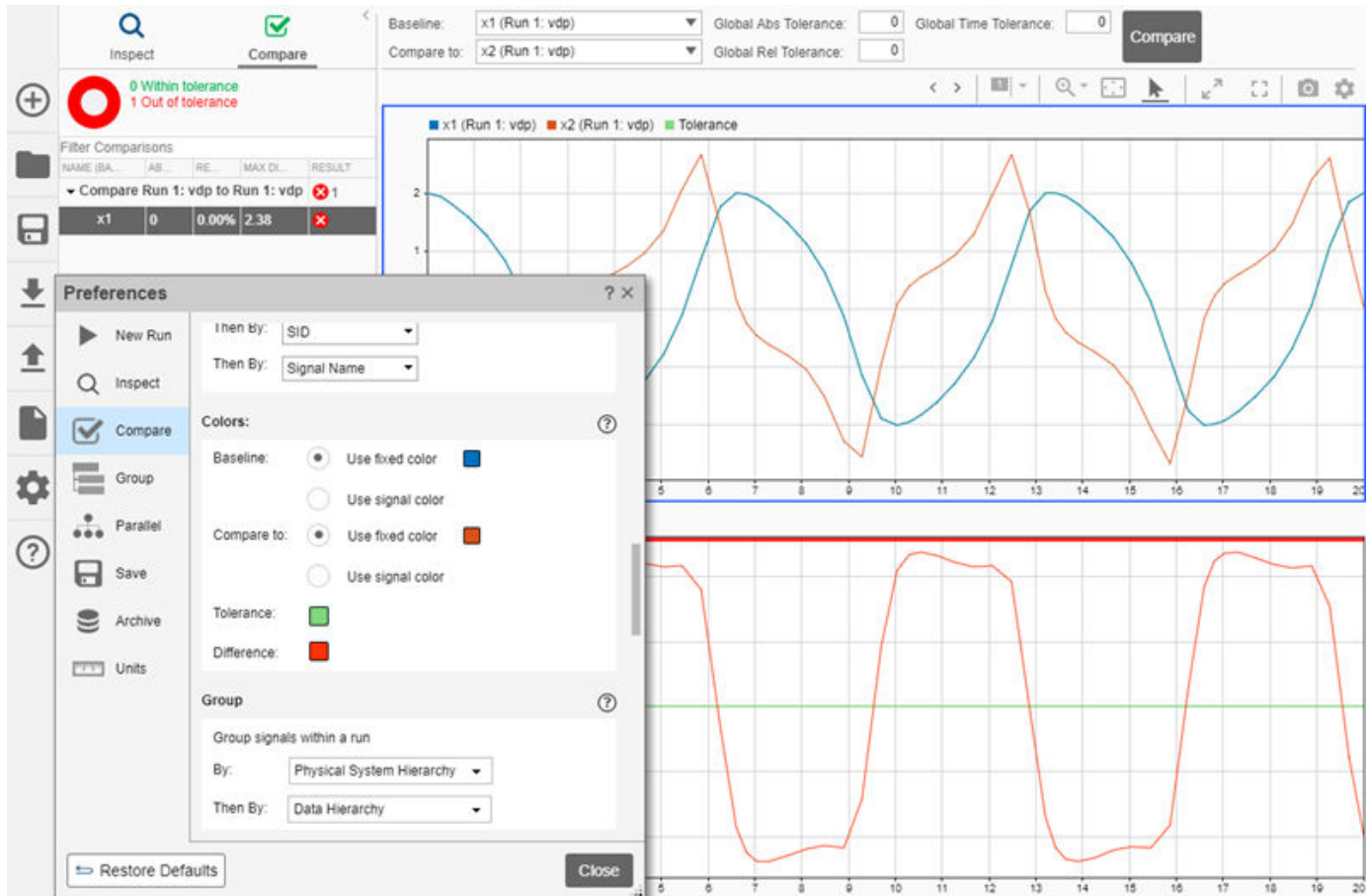
When you store data in a file or data format that the Simulation Data Inspector does not support, you can create a custom data reader to import the data into the Simulation Data Inspector using the `io.reader` base class. You can write a reader that imports data from the workspace or from a file, and you can register multiple readers for the same file extension. You can use the custom reader to import data using the Simulation Data Inspector UI or programmatically using the `Simulink.sdi.createRun` function. For details, see:

- [Import Workspace Variables Using a Custom Data Reader](#)
- [Import Data Using a Custom File Reader](#)

Select colors to use for comparison results in the Simulation Data Inspector

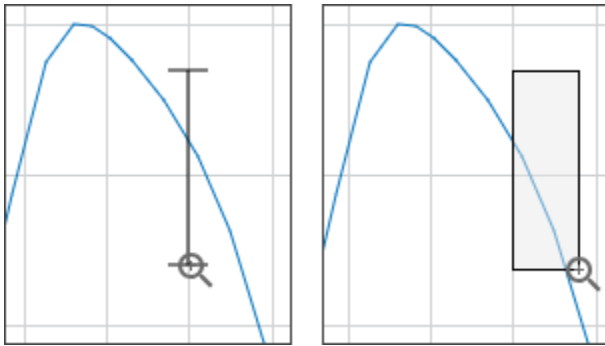
You can select colors to use for the baseline and compared signals in comparison results on the **Compare** tab of the Simulation Data Inspector. By default, the Simulation Data Inspector displays the baseline and compared signals using a fixed color and not the signal colors used for the signals on the

Inspect tab. You can choose a different fixed color to use for comparison results, or you can choose to use the signal color from the **Inspect** tab. You can also select colors for the tolerance and difference shown in the difference plot.



Simulation Data Inspector zoom enhancements

In R2020b, the marquee zoom functionality in the Simulation Data Inspector adapts according to the area you select with the cursor. For example, when you select a very narrow area, the zoom behavior is to zoom on the y-axis only. After the selection widens beyond the threshold indicated in the visual representation, the zoom behavior is to zoom on both the x-axis and the y-axis. The highlight around the selected region indicates the zoom behavior for the selection. Similarly, if you make a selection along the x-axis, the zoom behavior is to zoom on the x-axis only until the selection includes a tall enough selection of the y-axis.



You can still select a zoom action from the zoom menu on the Simulation Data Inspector toolbar to always zoom only in time or only along the y-axis.

Specify a units system to use to display data in the Simulation Data Inspector

In R2020b, signals in the Simulation Data Inspector have two properties for units: `Stored Units` and `Display Units`. The `Stored Units` for a signal represent the units of the data at the time the signal was logged. The `Display Units` property allows you to visualize all your data using a consistent system of measurement without changing the logged data values. The Simulation Data Inspector performs the conversion between the `Stored Units` for a signal and the `Display Units` that you specify.

To simplify the process of selecting `Display Units` for all signals, you can select a system of units to use for displaying signal data. When you select a units system, the Simulation Data Inspector updates the units for any signal with a `Display Units` property value that is not a valid unit for that system. A system of units may have more than one valid unit for a given measurement type. For example, in the SI unit system, both kg and g are acceptable units. To use a consistent unit for a type of measurement, you can specify a unit to override the `Display Units` property for all signals of that measurement type. For example, to display all distances using m, specify an override unit of m.

Simulink logging enhancements

When you log states data using the `Dataset` format, states data streams to the Simulation Data Inspector.

When you log data to a MAT file:

- You can log Stateflow data to the file.
- Dashboard blocks and the Simulation Data Inspector display logged data during simulation.
- Logging supports `int64` and `uint64` data types as built-in types rather than fixed-point types.
- You can log to file even when the `InspectSignalLogs` setting is enabled.

Enhanced Simulation Data Inspector interactive comparison report

When you create a report for Simulation Data Inspector comparison results, you can change the report style between interactive and printable in the generated report. The printable report style stacks the signal properties and plots under the comparison results summary, so you can see all the

report data in the printed report. In prior releases, you had to choose the report style before generating the report, and you had to generate another report to view the results in the other style.

The interactive report also has updated styling, and you can resize the report sections.

Read and write data using Simulation Data Inspector functions in deployed applications

You can use import functions for the Simulation Data Inspector to read and write data in applications deployed using MATLAB Compiler or Simulink Compiler. The import functions have built-in readers that import data from MAT, CSV, and Microsoft Excel files into `Simulink.sdi.Run` and `Simulink.sdi.Signal` objects. You can also use Simulation Data Inspector export functions to write the data to the application workspace, a MAT-file, or a Microsoft Excel file. These functions are supported for deployment:

- `Simulink.sdi.createRun`
- `Simulink.sdi.Run.create`
- `Simulink.sdi.addToRun`
- `add`
- export function for `Simulink.sdi.Signal` objects
- export function for `Simulink.sdi.Run` objects
- `Simulink.sdi.exportRun`

Intel C and C++ toolchain support for Windows

You can compile generated code by using Intel C and C++ compilers for Windows. For model referencing, accelerator mode, and rapid accelerator mode, R2020b supports:

- Intel Parallel Studio XE 2020 with Microsoft Visual Studio 2017, 2019
- Intel Parallel Studio XE 2019 with Microsoft Visual Studio 2015, 2017, 2019
- Intel Parallel Studio XE 2018 with Microsoft Visual Studio 2015, 2017, 2019

Support for Intel Parallel Studio XE 2017 is removed.

For more information, see Supported Compilers.

Functionality Being Removed or Changed

Default Simulation Data Inspector time axis behavior changed

Behavior change

In R2020b, the time axis range is set by the **Time span** setting, which is 100 by default. Streaming data is plotted on the range specified by the **Time span** and if the simulation continues beyond the maximum, the display updates with a scroll or wrap behavior, based on the **Update mode** setting.

In previous releases, the time span for time plots in the Simulation Data Inspector was automatically set to the stop time for the simulation streaming data. You can configure this behavior in the Simulation Data Inspector by setting the **Time span** to auto.

Restore Defaults now restores all settings to default values in the Simulation Data Inspector*Behavior change*

In R2020b, you access all settings for the Simulation Data Inspector in a single Preferences window, and the **Restore Defaults** button in the Preferences window restores all settings to default values.

In prior releases, the Simulation Data Inspector settings were spread over several dialogs, and each dialog had a **Restore Defaults** button that would restore defaults for only those settings.

Simulink will no longer support loading ModelDataLogs data as simulation input*Still runs*

In R2016a, the ability to log data using the ModelDataLogs format was removed. In a future release, the ability to load existing data stored in the ModelDataLogs format as input for a simulation will be removed. You can convert data stored using the ModelDataLogs format to use the Dataset format by using the `Simulink.ModelDataLogs.convertToDataset` function.

sbuild no longer generates model reference simulation targets by default*Behavior change*

Starting in R2020b, the `sbuild` function does not generate model reference simulation targets by default. Excluding the model reference simulation targets allows for faster code generation of model hierarchies.

You can continue to generate both the simulation and code generation targets with the default `buildSpec` value ('StandaloneCoderTarget') by setting `IncludeModelReferenceSimulationTargets` to `true`.

Component-Based Modeling

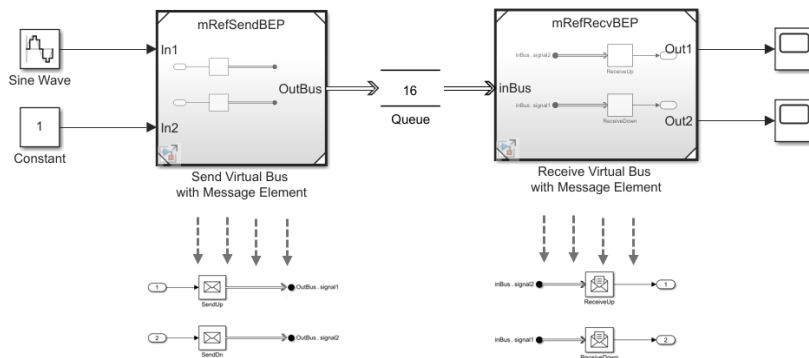
Schedule Partitions to Execute Based on Events Broadcast from Stateflow in the Schedule Editor

In R2020b, you can create and manage events in the Schedule Editor and bind the events to the partitions. Managing events in the Schedule Editor simplifies system creation and provides more scheduling flexibility. Events are triggers that allow dynamic scheduling of aperiodic partitions.

Using events with a test harness helps you avoid complex wiring between the test sequence block and the entire system. The generated test harness gets its own Schedule Editor which enables you to easily send events through the test harness. Through the test harness scheduler, you can test the model under different scenarios by triggering events at specific time

Support for virtual buses as component interfaces in message-based communication

In R2020b, you can use virtual buses with message elements to model asynchronous communication between model components. Use Bus Creator, Bus Selector, In Bus Element, and Out Bus Element blocks to send and receive virtual buses from your components' input and output ports. The Queue block is automatically duplicated for each message element of the virtual bus.



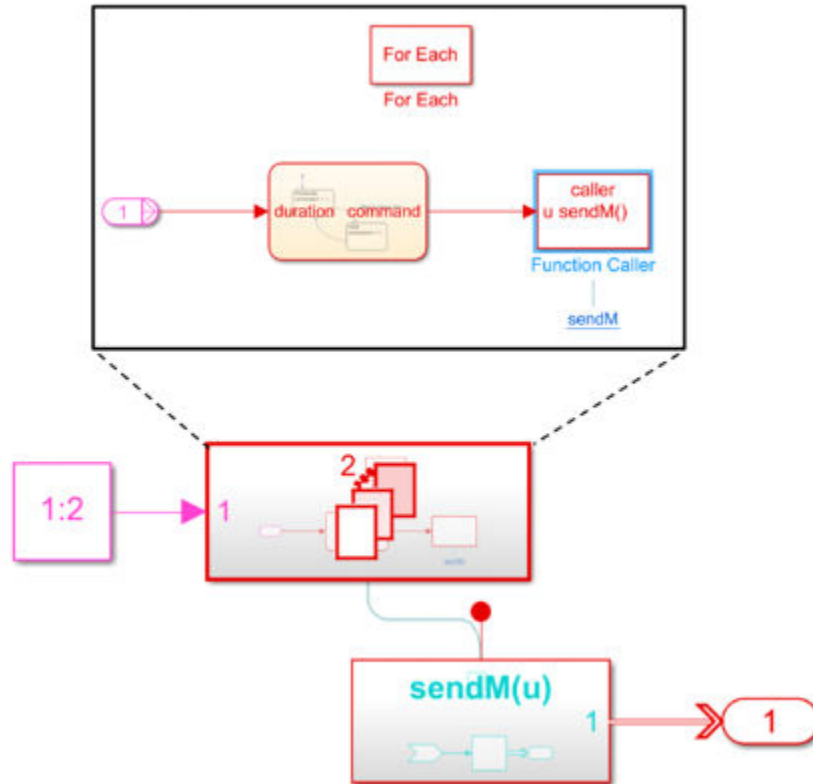
Use Sequence Viewer block to visualize virtual buses with message elements. Use Animation to track the virtual bus flow in your model during simulation.

For Each subsystem: Call a utility from repeated executions using the Function Caller block

You can now place a Function Caller block inside a For Each subsystem block to call a utility defined in a Simulink function outside the subsystem.

The utility is shared for all iterations and can be placed in the parent model.

By calling the utility, you can send and receive messages from within a For Each iteration.



For an example, see Temperature Control System Communicating with Messages.

Receive protected models with their dependencies in a project

You can now receive a protected model in a project archive (.mlproj). The project archive provides a way to share a project in a single file. You must open the project archive to create the project that contains the protected model, its dependencies, and its harness model..

You can simulate the harness model without defining missing values or objects because the project contains these protected model dependencies.

For more information, see Reference Protected Models from Third Parties.

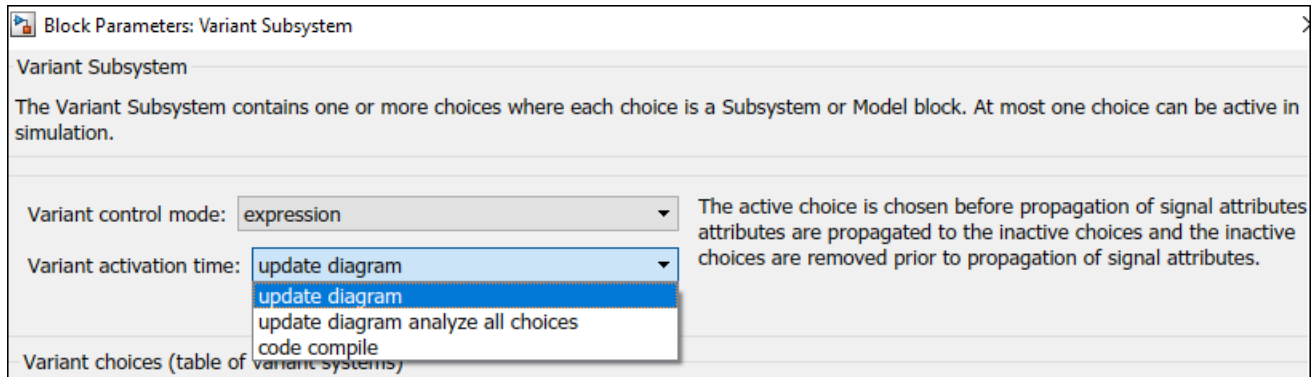
Restore and push edited and parameterized library blocks in the model using the improved Library Links Manager tool

In R2020b, you can use the **Library Links Manager** tool to restore and push edited and parameterized library blocks in a model. You can now push and restore multiple blocks at a time as well as nested disabled linked blocks. Prior to R2020b, the restore and push operations were done using the Library Links tool.

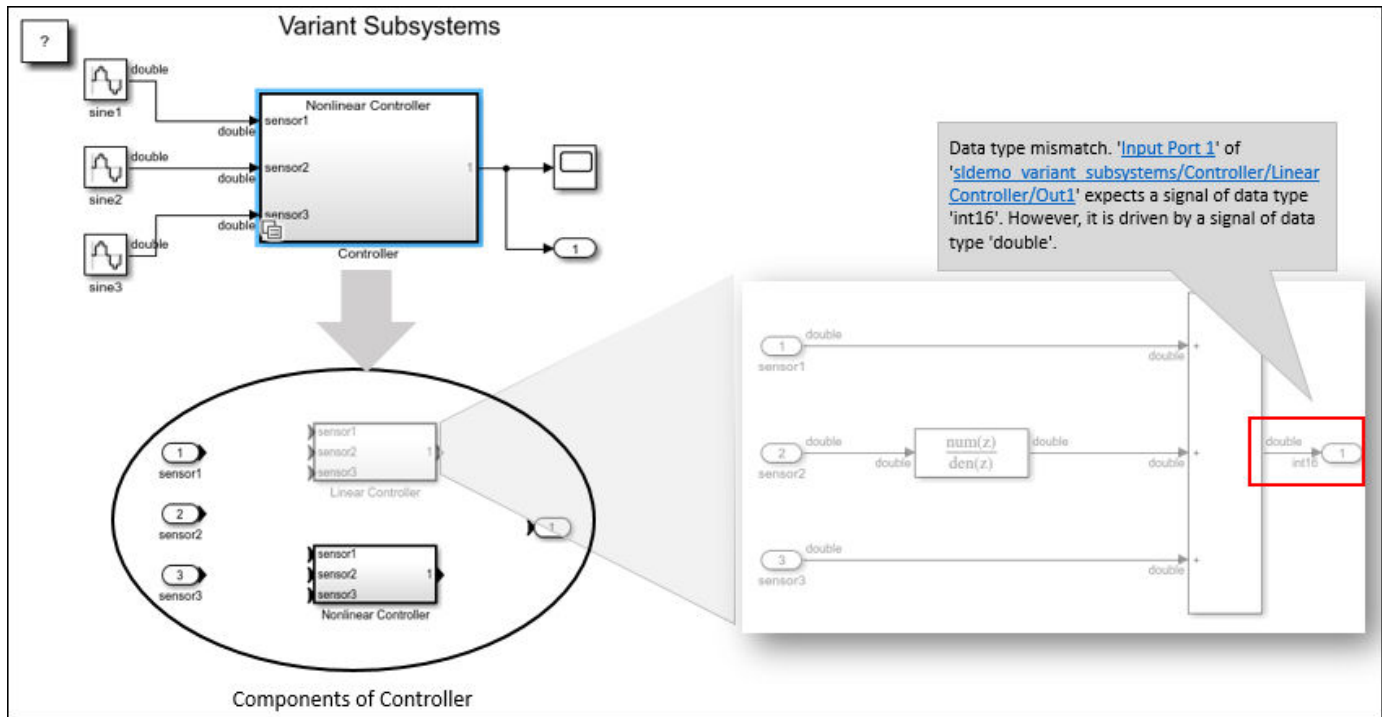
For more information on how to use the **Library Links Manager** to restore and push library blocks, see Restore Disabled Links.

Analyze all Variant choices using Variant activation time

In R2020b, the **All choices during update diagram and generate preprocessor conditionals** parameter is replaced by a new parameter, **Variant activation time**, which is available on the block parameters dialog box of all Variant blocks. With **Variant activation time**, you can analyze the variant choices during update diagram, which was only possible during code generation in previous releases. Analyzing variant choices during update diagram helps prevent incompatibilities such as datatype and dimension mismatches between the results of simulation and code generation. For more information, see Variant Sink.



For example, this model has potentially two variant choices: **Linear Controller** and **Nonlinear Controller**. **Linear Controller** is inactive and **Nonlinear Controller** is active. When you simulate this model with **Variant activation time** set to **update diagram**, the datatype mismatch at the **Out** port of the **Linear Controller** block is not detected because **Linear Controller** is an inactive choice. When you simulate this model with **Variant activation time** set to **update diagram analyze all choices** or **code compile**, both the choices are analyzed. In **update diagram analyze all choices**, the data type mismatch is detected during update diagram, whereas in **code compile**, the mismatch is detected during code generation.



The **Variant activation time** parameter is fully compatible with models created in previous releases of Simulink.

- When you open an existing model in the R2020b version of Simulink:

If All choices during update diagram and generate preprocessor conditionals is	Variant activation time is set to
off	update diagram
on	code compile

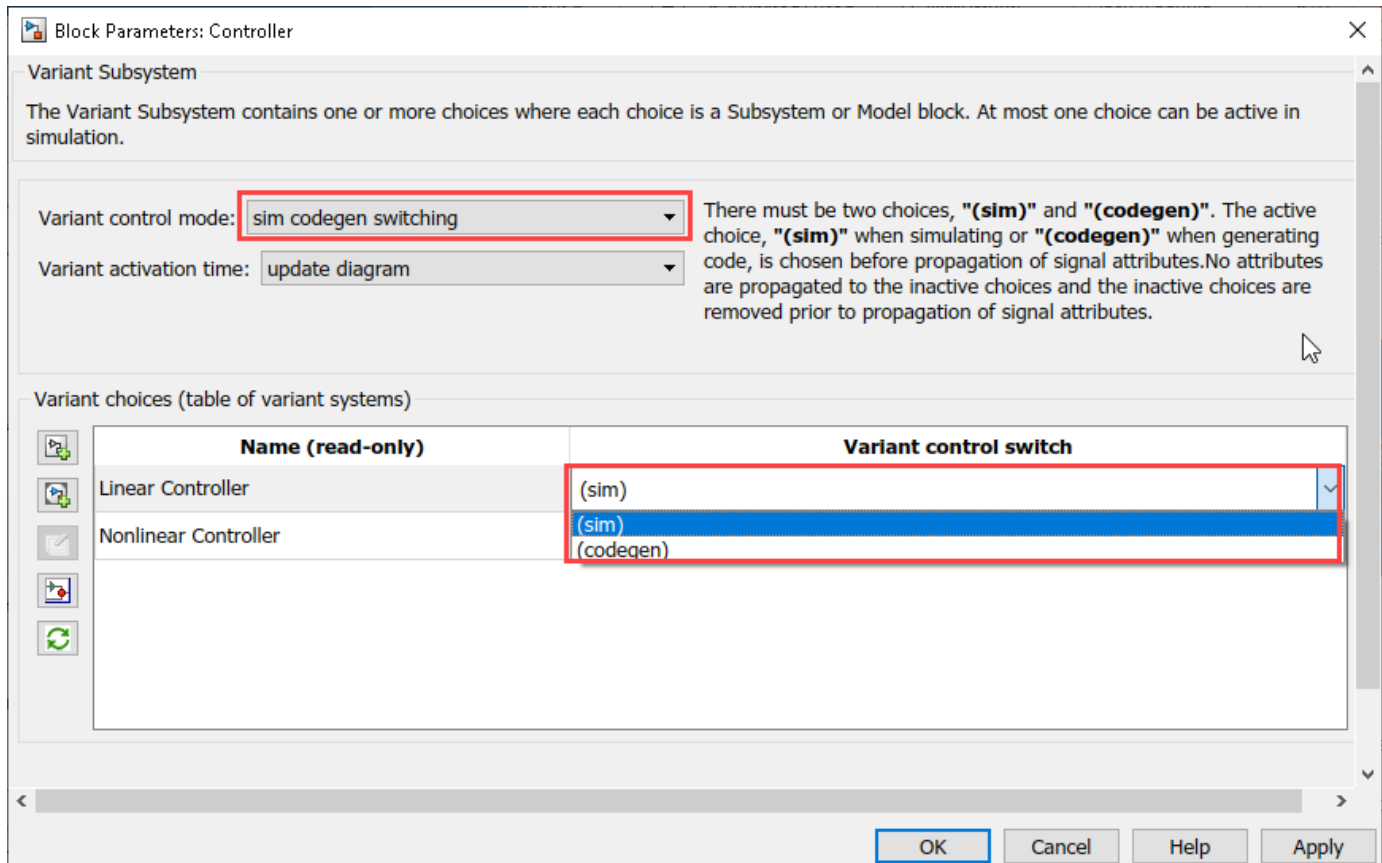
- When you create a model in R2020b and export it to earlier versions of Simulink:

If Variant activation time is set to	All choices during update diagram and generate preprocessor conditionals is set to
update diagram	off
update diagram analyze all choices	off
code compile	on

Automatic variant switching for simulation and code generation workflows

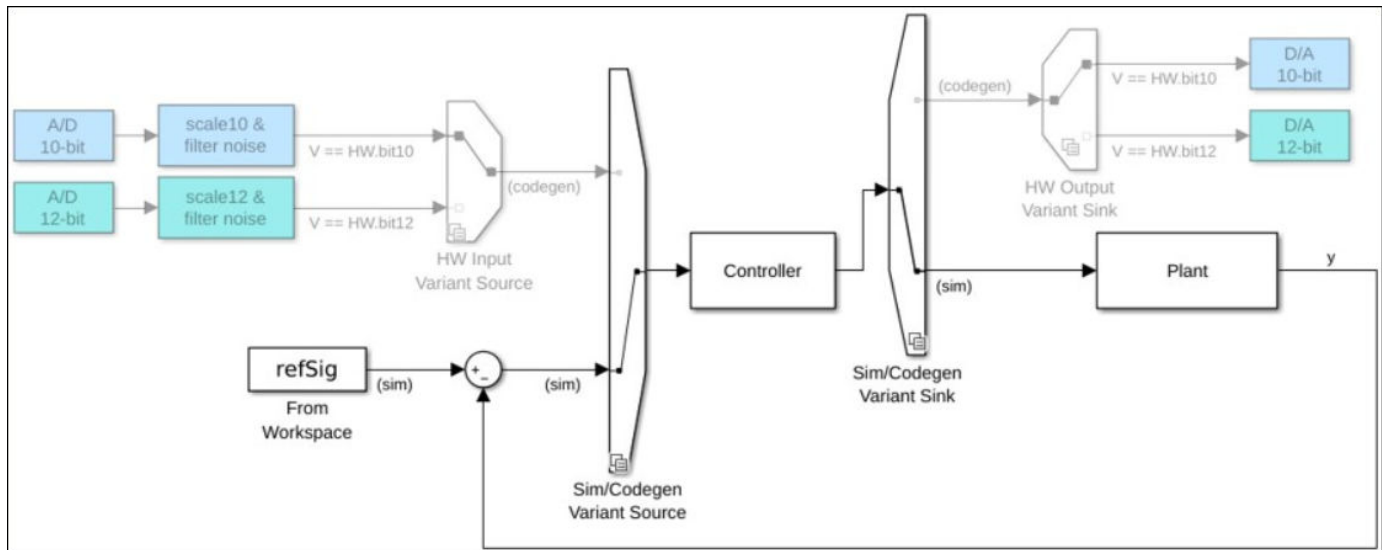
In R2020b, a new variant control mode enables automatic variant switching for simulation and code-generation workflows in variant blocks. This feature is convenient to switch between simulation and code-generation modes.

In the Block Parameter dialog box of the variant block, open the **Variant control mode** drop-down list and select **sim codegen switching**. This selection automatically enables the choices **sim** for simulation (normal, accelerator, rapid accelerator) and **codegen** for code generation workflows (Software-In-Loop simulation mode, Processor-In-Loop simulation mode, Generating code, External mode).



In the **Variant activation time** drop-down list, you can choose **update diagram** or **update diagram analyze all choices**. For data signals, **update diagram analyze all choices** ensures that the signal attributes (data types, dimensions, etc.) between both choices are consistent.

Here is a sample screen showing a variant block switched to **sim** choice.



Use true and false as conditions in Variant blocks

From R2020b onward, you can use the keywords `true` and `false` as conditions in Variant blocks.

Here is a sample screen showing a Variant Source block with Variant conditions specified as `true` and `false`. When you simulate this model, the first input port of the Variant Source block always evaluates to `true` and the second input port always evaluates to `false`.

Port	Variant control expression	Condition (read-only)
1	true	(N/A)
2	false	(N/A)

Suppress digital signature verification of protected model programmatically

Previously, to suppress digital signature verification of an individual protected model for the current MATLAB session, you used the error message in the **Diagnostic viewer** that appeared when you used the protected model. In R2020b, you can suppress signature verification for a protected model by using the `Simulink.ProtectedModel.suppressSignatureVerification` function before you use the model.

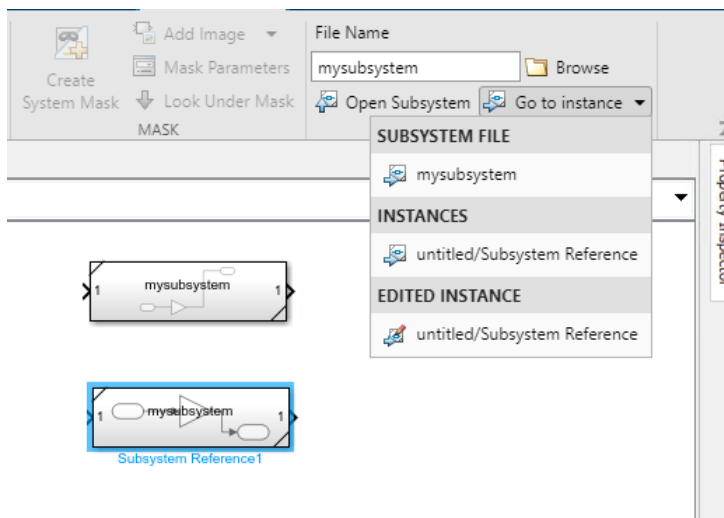
For more information, see Reference Protected Models from Third Parties.

Perform commonly used subsystem reference tasks using the toolstrip menu

In R2020b, the toolstrip menu is enhanced with options to interact with an instance of a subsystem reference instance without opening the instance. You can now:

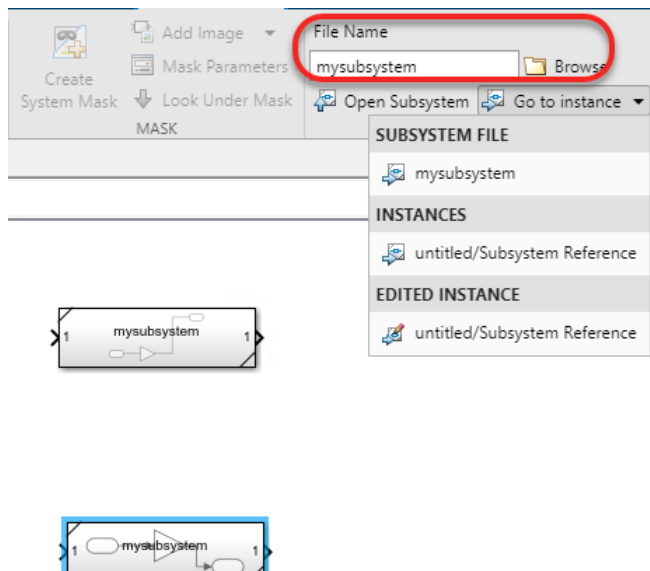
- Navigate to any open instance of a referenced subsystem.

To navigate to an instance, on the **Referenced Subsystem** tab, click the **Go to instance** drop-down menu and select the instance. In the drop-down menu, you can view the name of the subsystem file and the edited and non-edited instances of the file.



- Change the subsystem file referenced in a model.

To change the subsystem file referenced, go to the **Referenced Subsystem** tab. Enter the name of the subsystem file in the **File Name** field or click **Browse** to open a subsystem file from the local directory.



For more information on how to reference a subsystem in a model, see [Subsystem Reference](#).

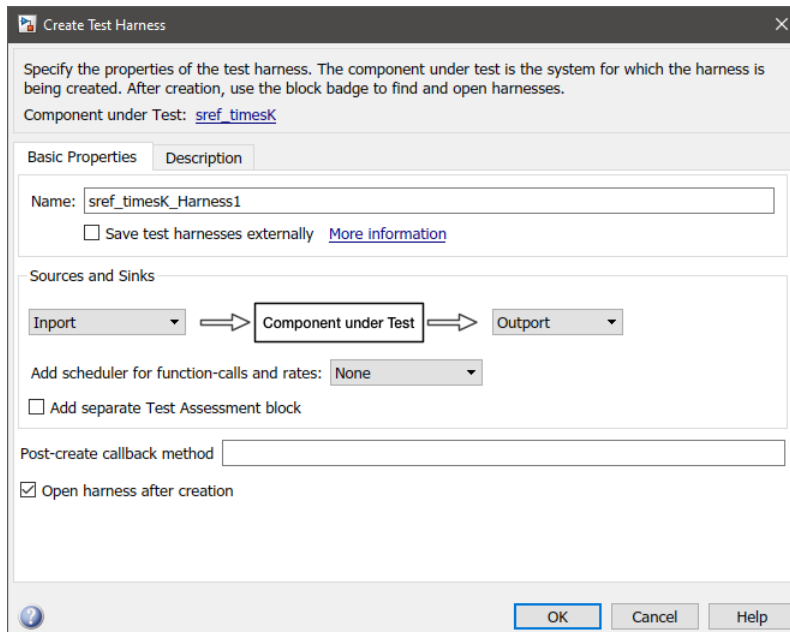
Simulate subsystem block diagrams using test harness from the subsystem context menu

In R2020b, you can create test harnesses from the subsystem context menu in the toolstrip. You can also set a default test harness for the subsystem from a set of available test harnesses, stop a test harness simulation, and a set stop time for the default test harness. Creating test harness requires a Simulink Test license.

To simulate with a test harness:

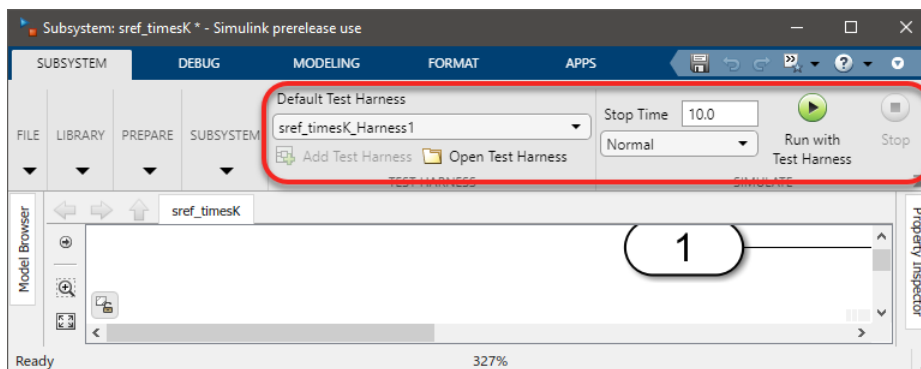
- 1 From the Simulink Editor, open a subsystem block diagram.
- 2 In the toolstrip, on the **Subsystem** tab, click **Add Test Harness**.

The **Create Test Harness** window appears.



- 3 Specify a name for the new test harness and click **OK**. This becomes the default test harness.
- 4 Click **Run with Test Harness** to simulate the block diagram.

You can also stop the simulation with the **Stop** button, set **Stop Time** for the default simulation, and update the block diagram with the **Update with Test Harness** button.



Schedule Editor Supports Partitioning with Variable-Step Solvers

The Schedule Editor now allows partitioning of subsystems and model blocks when using variable-step solver, with all the implicit rates put in one implicit task. This functionality enables you to create partitions in the model with continuous dynamics. You can switch the models with partitions from fixed-step solver to variable-step solver. All the rates that are not partitioned in the model with variable-step solver, are executed as a single task.

Parameter CompiledVariantInfo

You can use the parameter `CompiledVariantInfo` to check if a block is active during simulation and if it is part of generated code.

Here is an example.

```
cinfo = get_param(gcb, 'CompiledVariantInfo');
```

The parameter returns these values:

struct with fields:

```
IsActive: 'on'
```

```
IsInCodegen: 'on'
```

MatchFilter option with find_system

In R2020b, the MatchFilter option is added to the existing filtering mechanisms of find_system. MatchFilter option provides these advantages:

- Allows you to filter blocks with customized filter functions
- Avoids processing blocks when filters do not match
- Applies complex filters on block, lines, or annotations, to filter the results internally

This example shows how to use MatchFilter to find all non Inport and, Outport blocks in a model using your own defined filter function, nonInOutBlocks:

```
blks = find_system('ModelName', 'MatchFilter', @nonInOutBlocks)
```

```
function match = nonInOutBlocks(handle)
match = true;

if strcmp(get_param(handle, 'Type'), 'block')
    blockType = get_param(handle, 'BlockType');
    if strcmp(blockType, 'Inport') || ...
        strcmp(blockType, 'Outport')
        match = false;
    end
end
end
```

This example shows how to find active Gain blocks in a model using a built-in MatchFilter, Simulink.match.activeVariants, after compiling the model:

```
set_param('ModelName', 'SimulationCommand', 'update')
gainBlocks = find_system('ModelName', 'MatchFilter', @Simulink.match.activeVariants, 'Type', 'block', 'BlockType', 'Gain')
```

find_system enhancements with Variants

In R2020b, find_system with variants is enhanced to provide consistent results among all variant blocks.

You can also find active variants or code compile variant blocks using built-in MatchFilter options. To do so, compile the model and apply the appropriate MatchFilter options:

- Simulink.match.activeVariants
- Simulink.match.codeCompileVariants

This example shows how to use find_system to find active variants in a model:

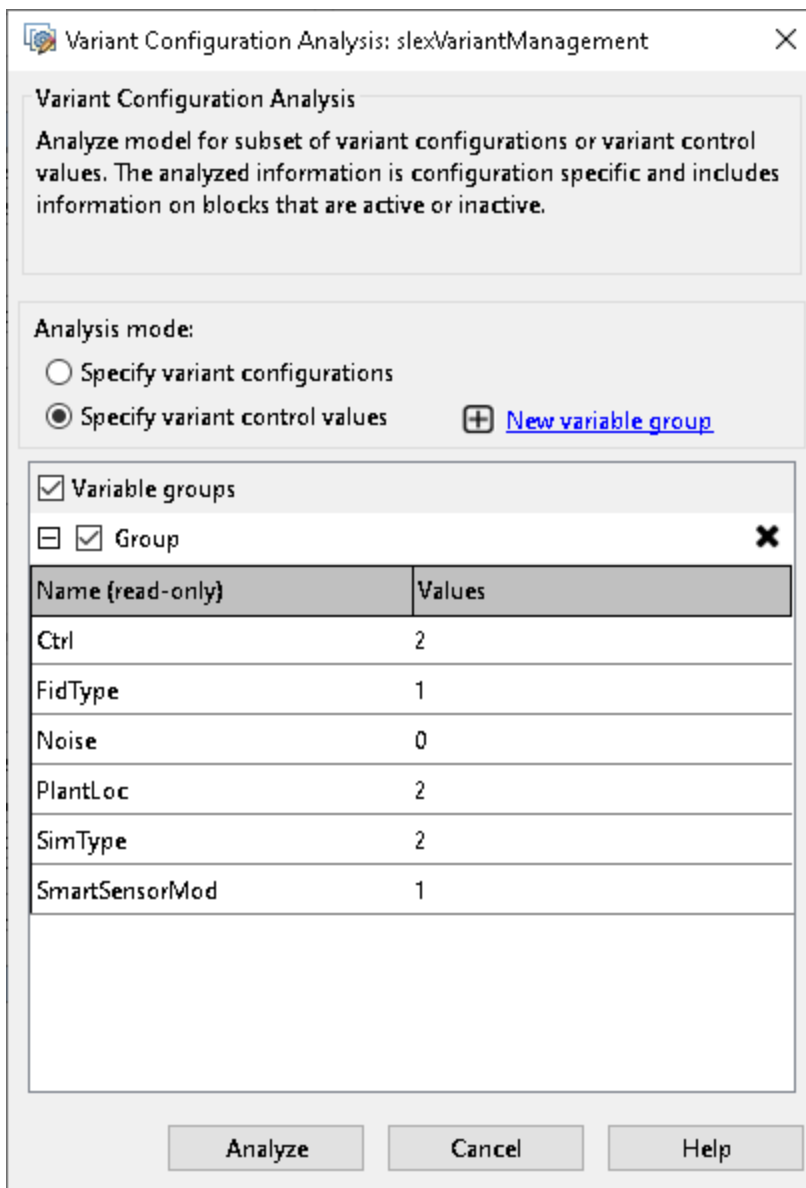
```
% Step1: Compile the model.
set_param(model, 'SimulationCommand', 'update');

% Step2: Call find_system using the built-in MatchFilter.
activeBlks = find_system(model, 'MatchFilter', @Simulink.match.activeVariants);
```

The existing Variants option works only for Variant Subsystem blocks with **Propagate conditions outside of variant subsystem** set to off.

Variant Configuration Analysis Tool: Specify variant control values

In R2020b, using the **Analysis mode** in the Variant Configuration Analysis tool, you can either select the required variant configurations or specify variant control values to perform the analysis.



If you select the **Specify variant configurations** option, you can select the required variant configurations from the list of associated variant configurations.

If you select the **Specify variant control values** option, you can create multiple variable groups that correspond to different configurations. Click **New variable group** to set the values for variant control variables. You can either specify a variant control value or select **Ignored** from the drop-down list. If you select **Ignored**, then that variant control variable is not considered while analyzing the model.

For more information on the Variant Configuration Analysis tool, see *Analyze Variant Configurations in Models Containing Variant Blocks*.

Before R2020b, in the Variant Configuration Analysis tool, you could only select the required variant configurations from the list of associated variant configurations to perform the analysis.

Variant Manager: Improved handling of variant control variables defined in InitFcn

Starting in R2020b, if you define variant control variables in the **default configurations** using Variant Manager, then it cannot be overridden by the values defined for variant control variables in the `InitFcn`. This allows you to configure the model for variations using the variant manager.

Before R2020b, the values defined for variant control variables in the `InitFcn` was overriding the variant control variables defined in the **default configurations**.

Variant Connector support in Variant Manager

In R2020b, the Variant Manager supports the new Variant Connector block. You can manage various variation points that are modeled using Variant Connector blocks in a system model. For more information on Variant Manager, see *Variant Manager Overview*.

Subsystem Reference support in Variant Reducer

In R2020b, Variant Reducer reduces variant models containing Subsystem Reference blocks. The referenced subsystems are reduced to simplified models based on the selected variant configurations. For more information on reducing a model, see *Reduce Models Containing Variant Blocks*.

Set simulation target language to build custom code

In R2020b, you can specify the programming language to build your custom code. You can set the language under **Configuration Parameters > Simulation Target > Language**. Language options include C or C++ programming languages. The following blocks offer this setting:

- MATLAB Function
- MATLAB System
- C Caller
- C Function
- Stateflow charts

To learn more, see *Language*.

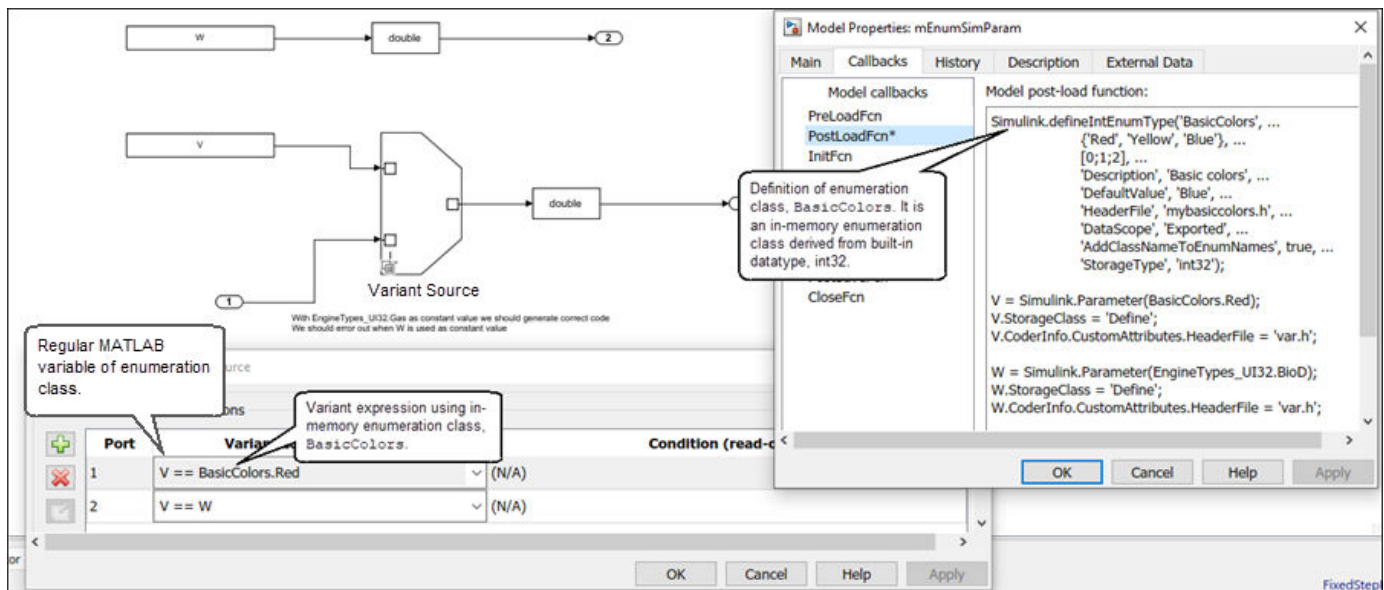
Enhanced support for different types of variables in Variant expressions

R2020b includes several enhancements to the type of variables supported in Variant expressions during simulation and code generation.

- In previous releases, the variable on the left side of the Variant expression had to be a `Simulink.Parameter`. From this release onwards, a regular MATLAB variable of enumeration class is also supported.
- In previous releases, the enumeration class on the right side of the Variant expression had to be a `classdef` enumeration class derived from `Simulink.IntEnumType`. From this release onwards, these integer-based enumeration classes are also supported:
 - `classdef` enumerations derived from built-in data types
 - In-memory enumerations derived from `Simulink.IntEnumType`
 - In-memory enumerations derived from built-in datatypes

Consider this model. The Variant control expression for the Port 1 of the **Variant Source** block is `V == BasicColors.Red`. `V` is a regular MATLAB variable of enumeration class, and `BasicColors` is an in-memory enumeration class derived from the built-in datatype `int32`. You can successfully simulate and generate code from this model.

For more information, see [Represent Variant Condition Values of Enumerated Type in Generated Code \(Embedded Coder\)](#).



Limit scope of Variant control variables from mask and model workspace of Variant subsystems

Prior to R2020b, you could use only the base workspace or data dictionary variables as Variant control variables. In R2020b, you can also use mask and model workspace variables as control variables during simulation. These options are supported only in Variant subsystems during

simulation. This approach lets you limit the scope of the variable, which was not possible in earlier releases. Limiting the scope helps avoid name conflicts and establish clear ownership of the variable between the blocks. It also enables allows you to use same names for variables in different scopes.

Variant control variable in mask workspace during simulation

Previously, setting different active choices for multiple instances of blocks that belonged to a locked library required you to write mask initialization scripts. Editing mask initialization scripts can be cumbersome if there are multiple levels of nested libraries or if the model contains many blocks. In R2020b, you can seamlessly set different active choices for multiple instances of custom library blocks.

You can use the mask parameter of a Variant subsystem as a Variant control variable. You can also define a MATLAB variable in the initialization script of the mask workspace and use it as a control variable. The variables that you define in the mask workspace have a limited scope. These variables can be used only by the given block and the underlying layers of that block, so you can use same name for the variables in different masks. Using the same name allows you to set different active choices for multiple instances of a locked library and reduces the number of variables in the base workspace.

For more information, see Approach 1: Use Mask Parameter as a Variant Control Variable and Approach 2: Use Mask Initialization Variable as a Variant Control Variable.

Variant control variable in model workspace during simulation

You can use the model workspace variable of Variant subsystem as a Variant control variable. Variables that are model arguments are not supported. For more information, see Approach 3: Use Model Workspace Variable as a Variant Control Variable.

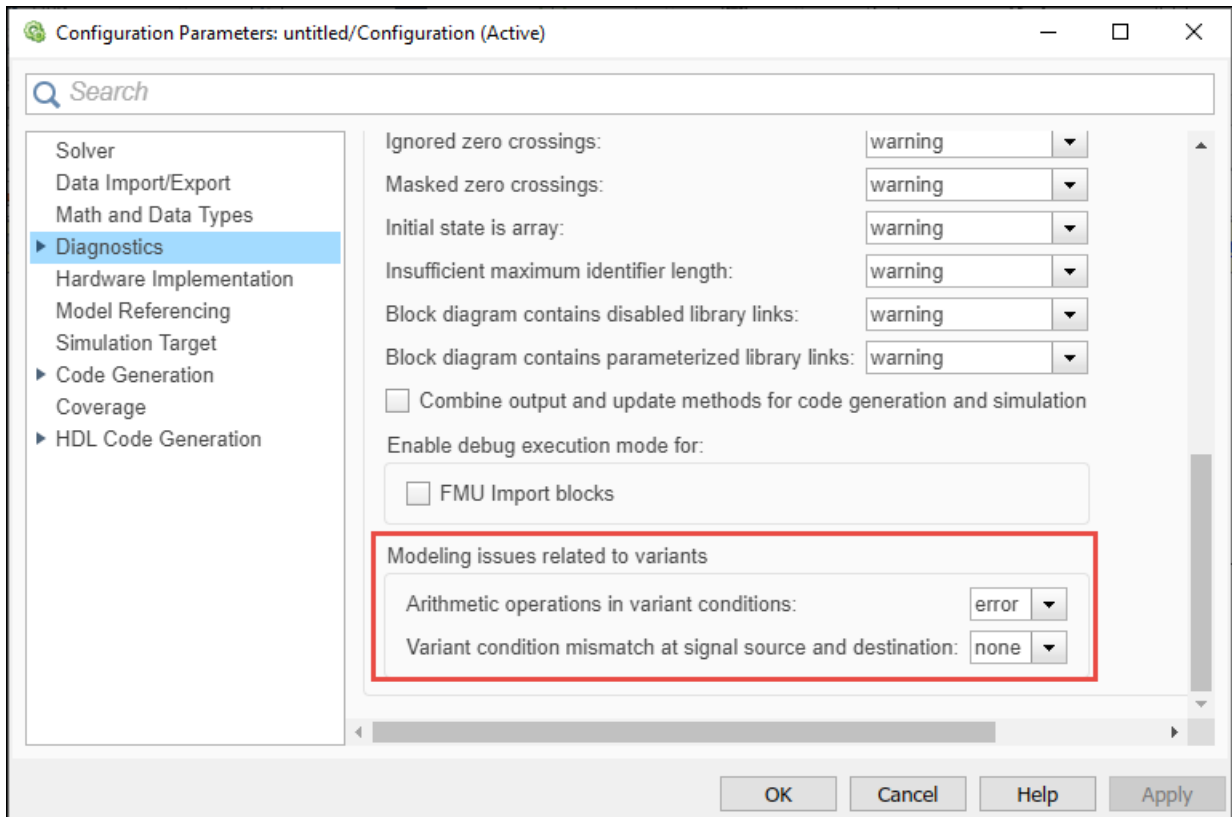
Handle Variant-related modeling issues using new diagnostic

The Configuration Parameters dialog box has a new section for handling variants related to modeling issues: **Modeling issues related to variants**. This section contains two diagnostics, **Arithmetic operations in variant conditions** and **Variant condition mismatch at signal source and destination**.

- **Arithmetic operations in variant conditions:** This diagnostic used to be in the **Advanced parameters** section of the Configuration Parameters dialog box. In R2020b, this diagnostic moves to the **Modeling issues related to variants** subsection of the **Advanced parameters** section. For more information, see Arithmetic operations in variant conditions.
- **Variant condition mismatch at signal source and destination:** When you generate code using Embedded Coder, you can use this new diagnostic to specify the action to take place if there are any variant-related modeling issues that may result in unused variables during code compilation. Unused variables are created in the generated code when there is discrepancy in the variant conditions that propagate between the source and the destination blocks.

If you set the value of this parameter to **warning** or **error**, Simulink displays a warning or an error when it sees a discrepancy. Detecting discrepancies increases the likelihood of generating C code that is compliant with the Rule 2.2 of the MISRA C:2012 guidelines. For more information, see Variant condition mismatch at signal source and destination.

To navigate to the **Modeling issues related to variants** section, select **Model Settings > Diagnostics**. Expand the **Advanced parameters** section and scroll to the bottom of the dialog box.



SIMD optimization for simulation targets

SIMD optimizations are applied where possible to referenced models that use a simulation target, or models that simulate in classic accelerator or rapid accelerator modes.

Using the `SimSIMDOptimization` model parameter, you can control when SIMD optimizations are used for simulation target simulations.

For example, to enable the optimizations,

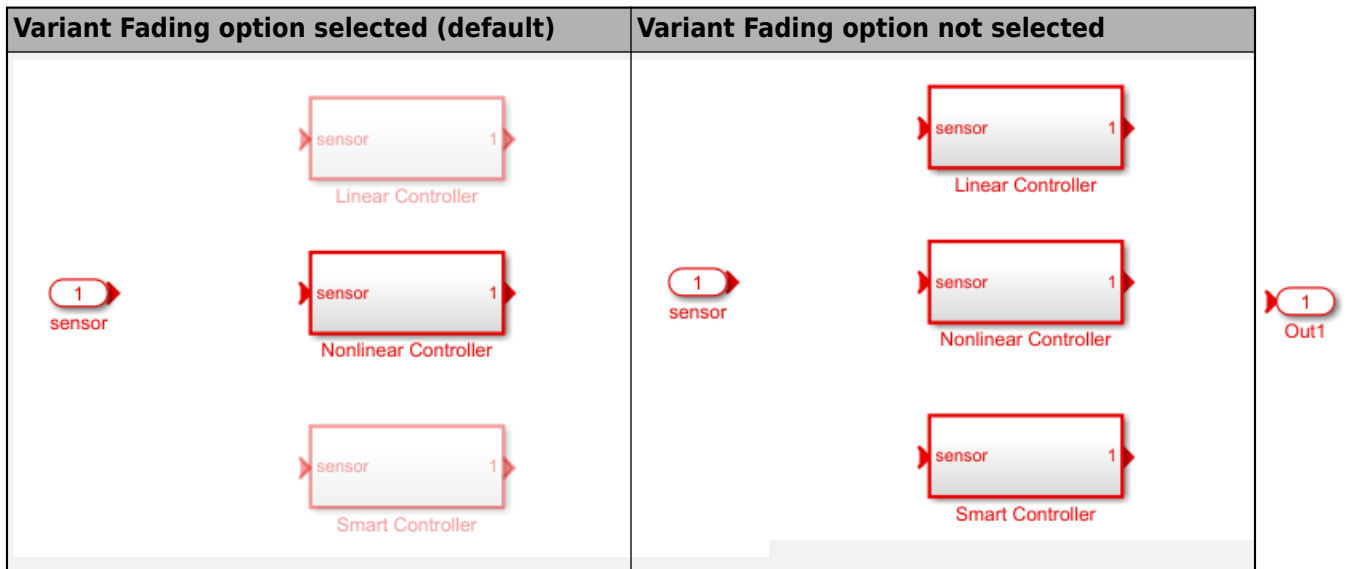
```
set_param(model, 'SimSIMDOptimization', 'Portable')
```

To disable the optimizations, set the parameter to `Off`. The default value for the parameter is `Portable`.

The optimizations are not supported when using LCC compilers.

Disable fading effect of inactive blocks using Variant Fading option

Simulink fades the color of inactive variant choices during update diagram. You can now choose to disable this fading effect by using the **Variant Fading** option. The **Variant Fading** option is available in the **Information Overlays** menu on the **Debug** tab of the Simulink Editor.



Improved version control for Simulink models

Prior to R2020b, when you saved a Simulink model, only the minor version number of the model was incremented.

In R2020b, the model version algorithm is improved to synchronize with every release of Simulink. The major model version is incremented by the number of releases passed since the model was last saved. The minor model version is reset to zero for every new release of Simulink and is incremented by one each time you save the model within the same release.

Consider that you create a model in R2019a release. The version number of the model is set as 1.0. Open the model in R2020b, make some changes, and save the model. The version number is now updated as 3.0. For more information, see [Manage Model Versions and Specify Model Properties](#).

Project and File Management

Automerge Tool: Customize source control tools to automerge models

You can now configure Git to automatically merge branches that contain changes in different subsystems in the same SLX file. For details, see [Customize External Source Control to Use MATLAB for Diff and Merge](#).

Project API: Export to previous version using `Simulink.exportToVersion`

You can now use `Simulink.exportToVersion` to export an entire project to open in a previous version of Simulink.

Dependency Analyzer: Export to archive, generate dependency report, and create a project from the dependency graph

Dependency analysis for projects and models is improved and includes new capabilities:

- Export files in the dependency graph to an archive.
- Save the dependency analysis results in a printable report.
- Create a project from a model dependency graph.
- Perform an incremental analysis for models. In prior releases, this was only possible for projects.
- Specify analysis scope for models. In prior releases, this was only possible for projects.
- Investigate where in your files the dependency to a product is introduced.
- Determine which files, if any, use functionalities shared among different MathWorks® products.

For more information, see [Dependency Analysis](#).

Compatibility Considerations

In R2020b, the Dependency Analyzer replaces the Simulink Manifest tool.

Project API: Change project definition file type and preserve source control history

You can now use `matlab.project.convertDefinitionFiles` to programmatically change the project definition file management from the type selected when you first created the project to a new type. `matlab.project.convertDefinitionFiles` preserves the source control history of your project. To avoid any merging issues, make sure to convert the definition file type only once for your project.

Model Templates: Edit Simulink Model Templates

You can now open model files templates for editing from the **Current Folder** and the Simulink start page. For more details, see [Edit a Template](#).

Source Control: Improved workflow to set up Git source control

The workflow to set up Git is improved and simplified.

- You do not need to install command-line Git to fully use Git with MATLAB. You can now merge branches using the built-in Git integration.
- For newly created projects using Git and for projects that switched to Git from another source control, MATLAB automatically generates a `.gitattributes` file and populates it with a list of common binary files. You do not need to manually register binary files to prevent corruption.

For more information, see [Set Up Git Source Control](#).

Text Comparison: Compare text files in MATLAB Online

Starting in R2020b, you can compare text files in MATLAB Online using the `visdiff` function. To compare text files, in the Command Window, type:

```
visdiff('filename1.m','filename2.m');
```

Clear Simulink History Programmatically

You can now programmatically clear the list of favorites and the list of recent models and projects in Simulink. For more details, see `Simulink.history.clear`.

Model File Preferences: Notify when opening a model in an unopened project

You can now disable the prompt that notifies to open the project containing the selected model. For more details, see [Notify when opening a model in an unopened project](#).

polyspacePackNGo Function : Generate and package Polyspace option files from a Simulink model

In R2020b, you can package Polyspace® option files along with code generated from a Simulink model, and then analyze the code on a different machine in a distributed workflow. After packaging the generated code, create and archive options files required for a Polyspace analysis by using the `polyspacePackNGo` function.

See also:

- `polyspacePackNGo`
- [Run Polyspace Analysis on Generated Code by Using Packaged Options Files](#)

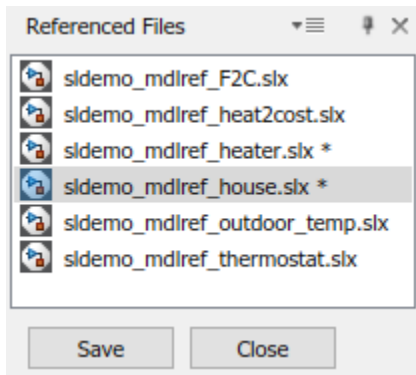
In a distributed workflow, a Simulink user generates code from a model and sends the code to another development environment. In this environment, a Polyspace user analyzes the generated code by using design ranges and other model-specific information. Previously, in this distributed workflow, you configured the Polyspace analysis options manually. Starting in R2020b, you do not have to manually create the option files when analyzing generated code by using Polyspace in a distributed workflow.

Product: .

Save and close referenced subsystems and models in a model hierarchy

You can now use the **Referenced Files** pane to save or close a referenced subsystem or model. This pane appears as a tab on the left side of the Simulink Editor when you navigate into a referenced model. You can also access this pane through the toolstrip.

- On the **Modeling** tab, in the **Design** gallery, click **Referenced Files**.
- If a referenced file in the model hierarchy has been modified, on the **Simulation** tab, click the **Save All** button arrow, then click **Save Referenced Files > Manage Referenced Files**.



For more information, see **Referenced Files Pane**.

Simulink cache functions for querying info and unpacking contents

You can now use functions to programmatically interact with Simulink cache files.

- `slxcinfo` — Create a MATLAB table that describes the contents of the specified Simulink cache file.
- `slxcunpack` — Unpack the simulation and code generation artifacts from the specified Simulink cache file for the current platform and MATLAB release outside of the build process.

For more information on Simulink cache files, see [Share Simulink Cache Files for Faster Simulation](#).

Functionality Being Removed or Changed

`slupdate` has been removed

`slupdate` has been removed. Use `upgradeadvisor` instead.

`dependencies.fileDependencyAnalysis` output arguments `manifestfile` and `depfile` will be removed

Still runs

`dependencies.fileDependencyAnalysis` output arguments `manifestfile` and `depfile` will be removed in a future release.

Starting in R2020b, you can only generate the manifest file programmatically. You can open manifest files created prior to R2020b using the Dependency Analyzer. To interactively generate a dependency

graph and report, perform a dependency analysis. For more details, see Analyze Model Dependencies.

Starting in R2020b, you can no longer generate a user dependencies file (.smd file). The function only returns a .smd file if it already exists. To manually add or exclude dependencies, create a project from your model. For more details, see Create a Project from a Model and Add Files to the Project.

SVN cleanup no longer removes unversioned or ignored files

Behavior change

Starting in R2020a Update 5, SVN cleanup only removes stale locks and unfinished transactions. It does not remove unversioned or ignored files.

You can manually remove unversioned and ignored files.

- 1** In the **Files** view, in the **All** tab, click the **SVN** header to sort files by their SVN status.
- 2** Select the **Not Under Source Control** files.
- 3** Right-click and select **Delete**.

For more information, see Get SVN File Locks.

Data Management

Export signal data in custom formats from the Signal Editor

You can now export signal data in custom formats from the Signal Editor user interface with the **Export** dialog window. For more information, see [Export Signals to Custom Registered File Types](#).

In R2020a, you could export signal data in custom formats into the Signal Editor user interface using only the `Simulink.io.FileType` class and its `export` method.

Default value for model arguments

Previously, each Model block had to specify an instance-specific value for model arguments defined in the referenced model. Without a specified value, the model failed to compile. In R2020b, if you do not specify an instance-specific value for a model argument, the argument uses the last value specified below it in the model hierarchy. For example, if you do not specify any instance-specific values, model arguments use the value of the arguments at the point of definition.

A new diagnostic configuration parameter, **No explicit final value for model arguments**, allows you to select the diagnostic action to take when the topmost Model block that can set the value for a model argument uses the inherited value instead of providing an explicit value.

For more information, see [Parameterize Instances of a Reusable Referenced Model](#).

Property dialog boxes for data objects use separate tabs for design and code generation properties

The property dialog boxes for Simulink data objects now separate design properties from code generation properties by using **Design** and **Code Generation** tabs. These include the property dialog boxes for `Simulink.LookupTable`, `Simulink.Breakpoint`, `Simulink.Parameter`, `Simulink.Signal`, `Simulink.NumericType`, `Simulink.AliasType`, and `Simulink.Bus` data objects.

Functionality being removed or changed

Simulink Interface Changes for Data Configuration

Behavior change

In R2020b, the Code Mappings editor becomes the primary location to configure model data elements for code generation.

- In the Model Data Editor, the Code view has been removed, and the editor does not display a **Code** section in the Property Inspector.
- For `Simulink.Signal` objects, you can no longer configure code generation properties in Model Explorer or in the Signal Properties dialog box. To configure these elements for code generation, use the Code Mappings editor or code mappings API.
- For data objects other than `Simulink.Signal` objects, where previously you could configure code generation properties in Model Explorer or in the property dialog box, links or buttons take you to the Code Mappings editor instead.

- In Model Explorer, in the **Contents** pane, click the **Configure** link in the **Storage Class** column.
- In the Model Explorer **Dialog** pane and in the property dialog box, on the **Code Generation** tab, click **Configure in Coder App**.

Simulink.BusElement objects no longer support sample time specification

Errors

The `SampleTime` property of `Simulink.BusElement` objects is no longer supported.

`BusElement` objects that specify a sample time cause an error during compile. To remove the sample time specification from a `BusElement` object, set its `SampleTime` to `-1`.

`Simulink.Bus.cellToObject` continues to accept cell arrays that specify sample time for bus elements. `Simulink.Bus.objectToCell`, `Simulink.Bus.save`, and `Simulink.Bus.createObject` continue to return cell arrays or arrays that include the sample time when it is noninherited. When the sample time is inherited (`-1`), they omit it.

To specify the sample time for an element of a bus, use the `SampleTime` block parameter of corresponding blocks. For example, you can use `In Bus Element`, `Out Bus Element`, and `Signal Specification` blocks to specify sample time.

Block Enhancements

Global variables in C Caller block

Starting in R2020b, you can expose global variables as a custom code function's interface when you map them to a C Caller block. To use global variables as ports or global arguments, turn on Enable custom code globals as function interface from **Model Settings > Configuration Parameters > Simulation Target**.

When a global variable in custom code is set as a **Global** scope on a C Caller block, the global variable does not serve as an interface to the custom code function and does not appear as a port on the C Caller block.

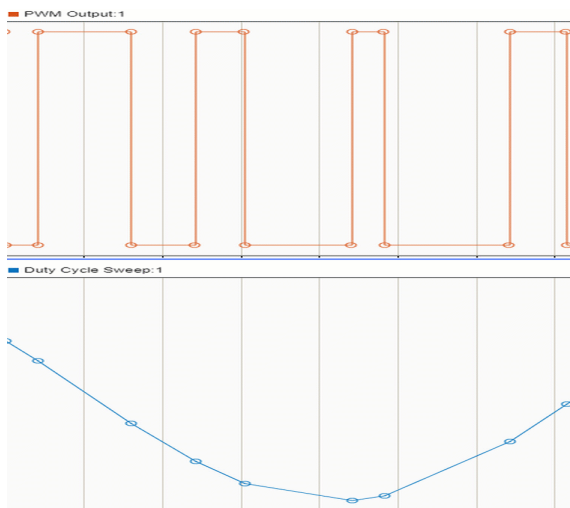
To learn more about Global arguments and scope, see Integrate C Code Using C Caller Blocks. To work with global variables through the command-line, create a FunctionArgument using the FunctionPortSpecification. Then, use the getGlobalArg to obtain a handle to the properties of each global variable.

New PWM block

Use the PWM block in continuous or discrete modes in order to simulate ideal pulse width modulated signals.



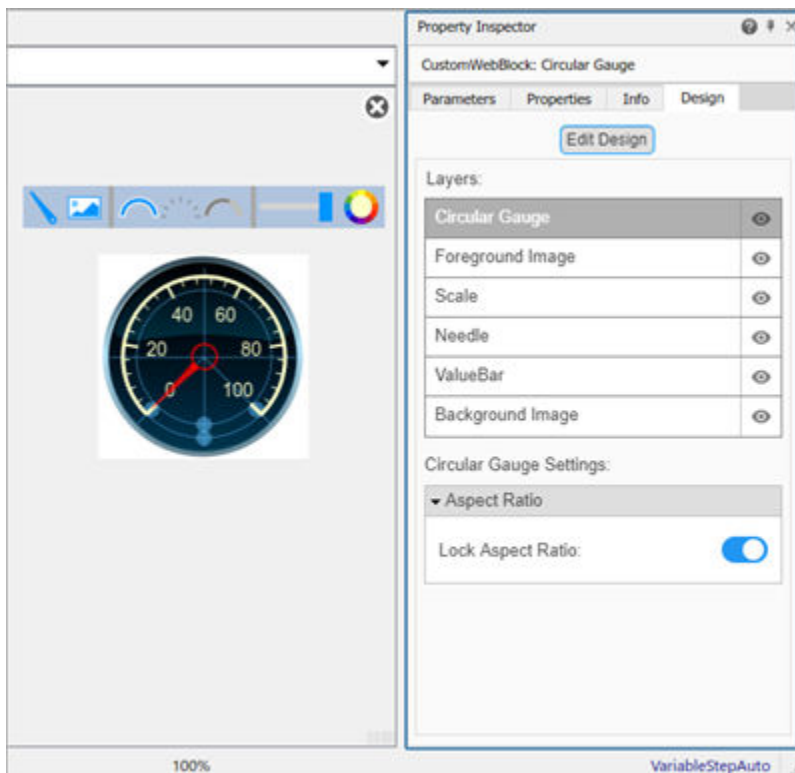
The PWM block modulates the width of the output pulse signal according to the input duty cycle.



For an example using this block, see PWM Control of a Boost Converter.

Property Inspector and toolstrip enhancements for custom dashboard blocks

You can use the Property Inspector to edit and design custom dashboard blocks. You can enter design mode from the **Design** tab of the Property Inspector or from the block tab in the toolstrip for a selected customizable block. To edit the design for a custom block, select the layer you want to edit in the Property Inspector. Properties of the layer you can edit are shown below the layer selection area in collapsible sections. You can use the toolbar above the block, the interactive design tools within the block itself, and the Property Inspector to customize the block.



Name property added for panels

To name a panel in your model, select the panel and specify the name in the **Name** field in the Property Inspector.

Simulink toolstrip support for dashboard blocks

The **Block** tab in the Simulink toolstrip turns into a block-specific tab when you select a block in your model from the Simulink Dashboard library or from the Flight Instruments library in the Aerospace Blockset™ Flight Control Analysis Library. From the toolstrip, you can connect, disconnect, and modify connections for the selected block. You can jump to the model element connected to the selected block, and you can add the selected block to a panel.

Add a foreground image to the Horizontal Gauge, Vertical Gauge, and Circular Gauge blocks

Using the **Design** tab in the Property Inspector, you can add a foreground image to a Horizontal Gauge, Vertical Gauge, or Circular Gauge block in your model.

Background Color, Foreground Color, and Opacity properties added for several dashboard blocks

You can specify a background color, a foreground color, and opacity for these dashboard blocks:

- Check Box
- Combo Box
- Edit
- Push Button
- Radio Button

IC block supports bus signals

The IC block now supports bus signals.

1-D, 2-D, and n-D Lookup Table blocks have new input ports

Starting in R2020b, the 1-D Lookup Table, 2-D Lookup Table, and n-D Lookup Table blocks have input ports for the table and breakpoint parameters. This capability enables you to use signals tunable at runtime to specify dynamic values for the table and breakpoint parameters. In previous releases, you specified these values directly in the block parameter.

FMU block changes

The FMU block **Input** and **Output** tabs have been renamed to **Input Bus** and **Output Bus**, respectively.

Simulate models containing string blocks in rapid accelerator mode from the MATLAB command line

In R2020a, support was added to enable rapid accelerator mode simulation. Simulation from the MATLAB command line is now also supported for rapid accelerator mode. For example:

```
set_param(gcs, 'SimulationMode', 'Rapid');  
sim(gcs);
```

MATLAB System block supports data types with slope bias scaling

The MATLAB System block now supports fixed-point data types with slope and bias scaling. In previous releases, the MATLAB System block supported only fixed-point data types with zero bias. For more information, see MATLAB System Block.

Simulink.LookupTable property dialog box updated

The standalone `Simulink.LookupTable` property dialog box and the version in the Model Explorer have been updated:

- When the **Breakpoints specification** property is set to `Explicit`, the **Value** property is replaced with a tabular interface and MATLAB expression text box at the bottom of the property dialog box.
- Dimension slices of multiple dimensions are now viewable in the tabular interface.
- The tabular interface lets you use standard table editing actions to edit and create table and breakpoint data.

For more information, see [View Simulink.LookupTable Object Data Using the Property Dialog Box Tabular Interface](#).

The new MATLAB expression text box is available only if the **Breakpoints specification** property is set to `Explicit`. Otherwise, the property dialog box is unchanged.

New SimStruct functions for string support

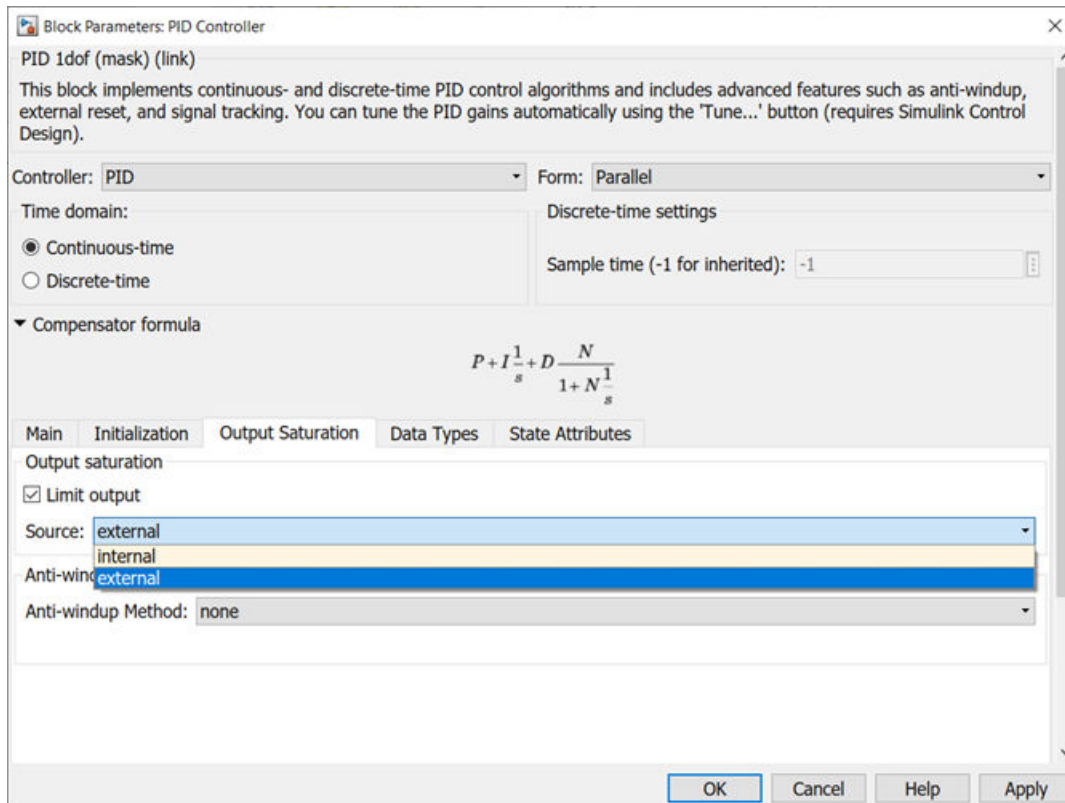
To create S-function blocks that work with strings with dynamic sizes during simulation, use these new SimStruct functions:

- `ssRegisterStringDataType` — Register string data type.
- `ssReadInputString` — Read C string from input.
- `ssIsStringDataType` — Determine if data type is string.
- `ssGetStringDataTypeMaxLength` — Get maximum length of string data type.
- `ssGetInputStringLength` — Get length of string from input port.
- `ssWriteOutputString` — Write C string to output port.

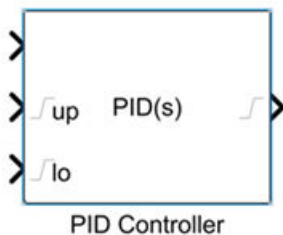
PID Controller Simulink Blocks: Option to provide output saturation limits as external inputs

PID Controller Simulink blocks now let you specify upper and lower output saturation limits as external inputs. Doing so allows you to implement saturation limits determined by logic or other calculations in the Simulink model and passed to the block. Previously, the output saturation limits had to be specified in the block parameters.

To enable external inputs for the output saturation limits, in the block parameters, on the **Output Saturation** tab, enable **Limit output**, and then in the **Source** list, select **external**.

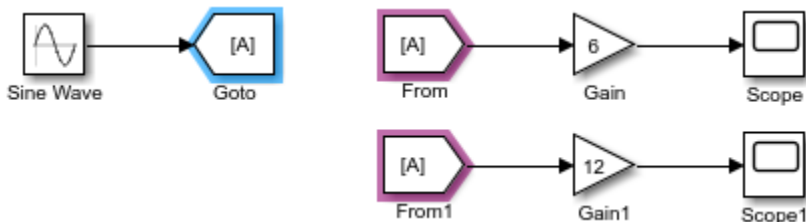


When you click **OK** or **Apply**, the new inputs appear on the block in the Simulink model.



Highlighting for related Goto and From blocks

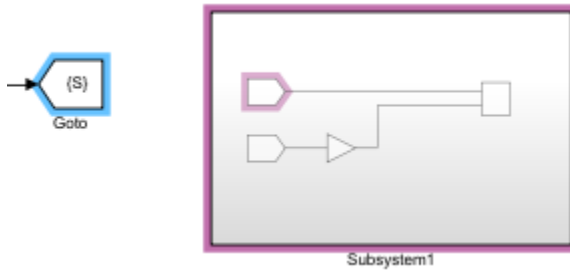
Selecting a Goto block highlights the corresponding From blocks.



Similarly, selecting a From block highlights the corresponding Goto block.



When a subsystem contains a corresponding Goto or From block, the Subsystem block is highlighted.



Specify name of subsystem created by `Simulink.BlockDiagram.createSubsystem`

You can now specify the name of the Subsystem block being created by the `Simulink.BlockDiagram.createSubsystem` function by using the comma-separated pair consisting of 'Name' and a character vector or string scalar.

By default, if a name is taken, the function uses the next available increment of the name. For example, if Name is set to 'Controller' and the diagram already contains a block named 'Controller', the function names the new subsystem 'Controller1'.

To return an error if the specified name is taken, use the comma-separated pair consisting of 'MakeNameUnique' and 'off'.

C Function block improvements

In R2020b, the C Function block:

- Accepts multi-dimensional array symbols.
- Accepts bus and enum as parameters.
- Supports row-major code generation.
- Supports InputOutput scope. An InputOutput scope enables you to create input and output ports with the same name. Reusing buffer on these ports can optimize memory use depending on the signal size and block layout. See [Call and Integrate External C Algorithms into Simulink](#).
- Supports a subset of C math library functions.

abs	ceil	floor	log	sinh
acos	cos	fmod	log10	sqrt
asin	cosh	labs	pow	tan
atan	exp	ldexp	sin	tanh
atan2	fabs			

For more information on multi-dimensional array support for the C Function block, see [Access Elements of a Matrix Using Output Code in a C Function Block](#) and [Use External Functions with Matrix Input in a C Function Block](#). To learn more on the C functions supported by the C Function block, see [Call C Library Functions From C Function Block](#).

Limit the range of data to process in a For Each subsystem

- "Number of iterations" parameter added to the For Each block dialog, providing the flexibility to perform the for-each operation on only a sub-range of input signals or mask parameters.
- Parameter allows symbolic specification and can be adjusted in generated code. See [Implement Dimension Variants for Array Sizes in Generated Code \(Embedded Coder\)](#).

Model blocks refresh automatically

If a referenced model is loaded, the Model blocks that reference it refresh automatically to show changes such as new or removed ports. For more information, see [Refresh Model Blocks](#).

Updated block parameter names for buses

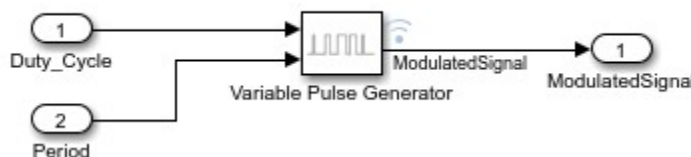
Starting in R2020b, the following block parameters have updated names in their respective Block Properties dialog boxes.

Block	Parameter Name in R2020a and Earlier	Parameter Name in R2020b and Later
Bus Creator	Signals in the bus	Elements in the bus
	Require input signal names to match signals above	Require names of inputs to match names above
	Override bus signal names from inputs	Use names from inputs instead of from bus object
Bus Selector	Signals in the bus	Elements in the bus
	Selected signals	Selected elements
Bus Assignment	Signals in the bus	Elements in the bus
	Signals that are being assigned	Elements that are being assigned

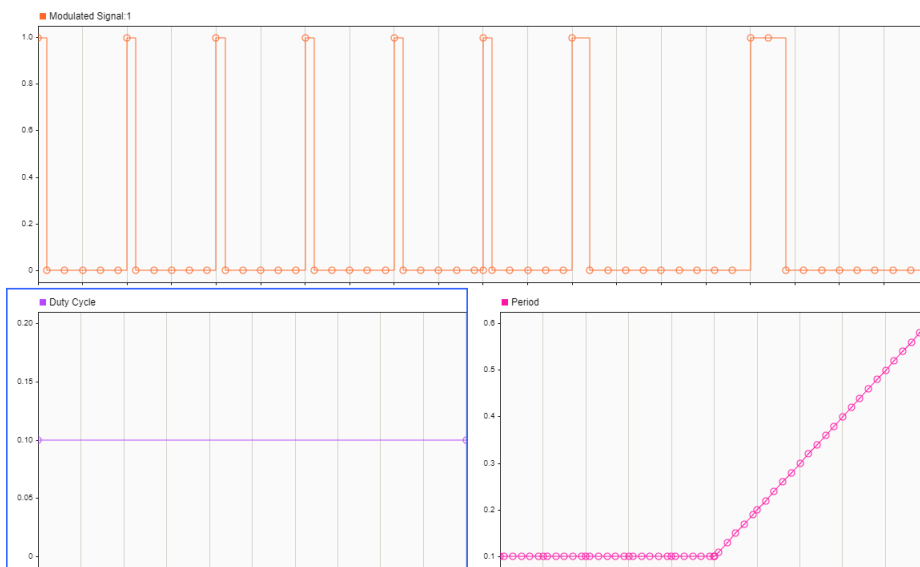
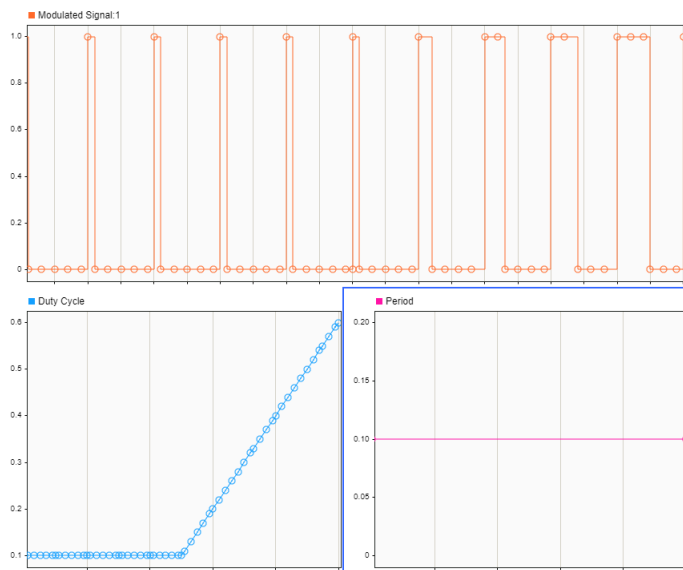
The programmatic use for these parameters remains the same.

New Variable Pulse Generator Block

Use the Variable Pulse Generator block in continuous or discrete sampling modes in order to simulate ideal pulse modulated signals.



You can modulate the output pulse by varying the duty cycle as well as the period inputs.



For an example using this block, see Voltage Controlled Oscillator.

Functionality Being Removed or Changed

Circular Gauge block replaces the Custom Gauge block

Behavior change

In R2020b, the Circular Gauge block replaces the Custom Gauge block. The Circular Gauge block has the same styling as the Custom Gauge block and allows you to customize the appearance of the block. The Circular Gauge block brings usability enhancements for interactive editing and the capability to add a foreground image.

When you open a model from a previous release that contains the Custom Gauge block, the Custom Gauge block is automatically replaced with a Circular Gauge block with a matching configuration.

PID Controller Simulink Blocks: ReferenceBlock parameter returns different path

Behavior change

Starting in R2020b, for PID Controller Simulink blocks, the `get_param` function returns a different value for the `ReferenceBlock` parameter. The `ReferenceBlock` parameter gives the path of the library block to which a block links. Previous and new paths for each block are as follows.

Block	Previous Path	New Path
PID Controller	'simulink/Continuous/PID Controller'	'pid_lib/PID Controller'
PID Controller (2DOF)	'simulink/Continuous/PID Controller (2DOF)'	'pid_lib/PID Controller (2DOF)'

The Discrete PID Controller and Discrete PID Controller (2DOF) blocks link to the same path as the corresponding continuous blocks because the blocks are identical, except for the default value of the `Time domain` parameter.

This change does not affect any other functionality or workflows. You can still use the previous path with the `set_param` function.

Product block in Matrix mode change in output type

Behavior change

The Product block might cause different numerical outputs from previous releases when **Multiplication** mode is set to `Matrix(*)` and either of these conditions is true:

- If any input port data type is of double data type, but the output port data type is not set to double.
- If there is no double data type for the input and output and these conditions are true:
 - Any input is of data type single
 - The output data type is not set as single

Connection to Hardware

Communicate with Android devices in Normal mode simulation using Connected I/O

Simulink Support Package for Android Devices now supports Connected I/O in Normal mode simulation. When you simulate a Simulink model in Normal mode with Connected I/O, the model communicates with the IO peripherals on the hardware, enabling you to verify the model design before deploying it on the hardware.

The Connected I/O feature is enabled on these Android sensor blocks.

- Accelerometer
- Gyroscope
- Light Sensor
- Ambient Temperature Sensor
- Pressure Sensor
- Humidity Sensor
- Location Sensor
- Orientation
- Magnetometer

Create an interactive display of Dashboard blocks on Android devices

These Dashboard blocks are enhanced to generate code on Android devices.

- Quarter Gauge
- Lamp
- Knob
- Slider
- Slider Switch
- Toggle Switch
- Rocker Switch

Code generation allows you to deploy these blocks as a standalone executable on the device. You can interactively tune your Android application using these blocks without any active connection between the device and computer.

Measure frequency and duty cycle of an external signal

Simulink Support Package for Arduino Hardware now supports measuring the frequency and duty cycle of an external signal connected to an input pin on the Arduino hardware. A new block, Input Capture, has been added to the support package to provide this timer-based functionality.

Measure linear acceleration, angular rate, and magnetic field using LSM9DS1 IMU sensor

Simulink Support Package for Arduino Hardware now supports measuring linear acceleration, angular rate, and magnetic field using a LSM9DS1 IMU sensor. This sensor comprises of an accelerometer, gyroscope, and magnetometer. A new block, LSM9DS1 IMU Sensor, has been added to the support package to measure linear acceleration, angular rate, and magnetic field, respectively.

Support for MODBUS RS485 serial data communication protocol

Simulink Support Package for Arduino Hardware now supports MODBUS serial data communication between Master and Slave devices connected over the same RS485 network.

This table lists the new blocks supporting Modbus serial data communication.

Blocks	Usage
Master Read	Master device requests data from the register(s) of the Slave device
Master Write	Master device writes data to the register(s) of the Slave device
Slave Read	Slave device requests data from its own registers
Slave Write	Slave device writes data to its own registers

New properties have been added in all four blocks in the **Configuration Parameters > Hardware Implementation > Target hardware resources** section. This table lists the new properties.

Property	Usage
Modbus properties	Configure the mode of operation, Master device settings, and Slave register settings
RS485 properties	Configure the baud rate for communication and control pin connection to the RS485 network

Support added for Nano 33 IoT board

Simulink Support Package for Arduino Hardware now supports deploying Simulink models on the Nano 33 IoT boards. You can use all the blocks from the support package library with the Nano 33 IoT board.

Publish data to WebSocket server using Arduino hardware

Simulink Support Package for Arduino Hardware now supports publishing data to a WebSocket server using Arduino hardware. A new block, WebSocket Publish, has been added to the support package to publish data in the JavaScript Object Notation (JSON) format.

Read data from WebSocket server using Arduino hardware

Simulink Support Package for Arduino Hardware now supports reading data from a WebSocket server using Arduino hardware. A new block, WebSocket Subscribe, has been added to the support package that outputs received data along with status indicating the validity of the data. The data must be in the JavaScript Object Notation (JSON) format.

Obtain Arduino library locations

You can now locate the folder in which an Arduino library resides using these functions.

Function Name	Function Description
<code>arduino.supportpkg.getAVRLibraryRoot</code>	Returns or displays the root directory where the Arduino AVR libraries are installed
<code>arduino.supportpkg.getAVRRoot</code>	Returns or displays the root directory where the Arduino AVR files are installed
<code>arduino.supportpkg.getBoardInfo</code>	Returns or displays the properties of the target selected in the a model
<code>arduino.supportpkg.getIDERoot</code>	Returns or displays the root directory where Arduino IDE is installed
<code>arduino.supportpkg.getLibraryRoot</code>	Returns or displays the root directory where Arduino library files are installed
<code>arduino.supportpkg.getSAMDLibraryRoot</code>	Returns or displays the root directory where Arduino SAMD libraries are installed
<code>arduino.supportpkg.getSAMDRoot</code>	Returns or displays the root directory where Arduino SAMD files are installed
<code>arduino.supportpkg.getSAMLLibraryRoot</code>	Returns or displays the root directory where Arduino SAM libraries are installed
<code>arduino.supportpkg.getSAMRoot</code>	Returns or displays the root directory where Arduino SAM files are installed

Support for I2C serial data communication

Simulink Support Package for LEGO® MINDSTORMS® EV3 Hardware now supports I2C serial data communication between the EV3 brick and I2C-based third-party sensors. A new block, I2C Register Read, has been added to the support package to enable the EV3 brick to request data from the I2C Slave device or Slave device register when connected over the same I2C network.

Support for parallel compilation of generated code and driver source code

Simulink Support Package for LEGO MINDSTORMS EV3 Hardware now supports the parallel compilation of generated code and driver source code. A new option, Disable parallel build, is available in the **Configuration Parameters > Hardware Implementation > Target hardware resources > Build options** section. You can either enable or disable this option. When you disable this option, the support package sequentially compiles the generated code and driver source code.

Communicate with Raspberry Pi hardware in Normal mode simulation using Connected I/O

Simulink Support Package for Raspberry Pi Hardware now supports Connected I/O during Normal mode simulation for these peripherals.

- SDL Video Display

- V4L2 Video Capture
- Serial Read
- Serial Write
- Websocket Publish
- Websocket Subscribe

When you simulate a Simulink model in Normal mode with Connected I/O, the model communicates with the I/O peripherals on the hardware, enabling you to verify the model design before deploying it on the hardware.

Base rate synchronization between ALSA Audio blocks and Simulink models

Simulink Support Package for Raspberry Pi Hardware now provides a base rate synchronization solution between the ALSA Audio Capture and ALSA Audio Playback blocks and Simulink models. A new property, **Base rate trigger**, has been added to the **Configuration Parameters > Hardware Implementation > Operating system/scheduler** section common in both the blocks. You can select these options in the **Base rate trigger** property depending on the type of base rate trigger.

- Operating system timer
- ALSA Audio Capture
- ALSA Audio Playback

MATLAB Function Blocks

Access fields of a structure array in a MATLAB Function block

In R2020b, you can access field data by indexing into a structure array in the MATLAB code you put inside a MATLAB Function block. See [Access Data in Structure Array \(MATLAB\)](#).

You can access fields of fixed-size structure arrays in MATLAB Function blocks. You can also use structure arrays that are made variable-size by using the `horzcat` function in MATLAB Function blocks.

Expansion of the structure array on the left side of the assignment is not allowed in MATLAB Function blocks.

Expanded code generation for datetime and duration arrays

In R2020b, code generation supports more MATLAB toolbox functions when you use `datetime` and `duration` arrays.

The supported functions for `datetime` arrays are:

- `colon`
- `datevec`
- `hms`
- `hour`
- `interp1`
- `max`
- `mean`
- `min`
- `minute`
- `posixtime`
- `repmat`
- `ymd`

The supported functions for `duration` arrays are:

- `ceil`
- `colon`
- `floor`
- `interp1`
- `intersect`
- `ismember`
- `issorted`
- `issortedrows`
- `linspace`

- `max`
- `mean`
- `median`
- `min`
- `mode`
- `repmat`
- `setdiff`
- `setxor`
- `sort`
- `sortrows`
- `std`
- `sum`
- `union`
- `unique`

For more information, see [Code Generation for Datetime Arrays \(Simulink\)](#) and [Code Generation for Duration Arrays \(Simulink\)](#).

Expanded code generation for tables and timetables

In R2020b, code generation supports more capabilities and MATLAB toolbox functions when you use tables and timetables.

Supported table and timetable capabilities are:

- Table and timetable variable names do not have to be valid MATLAB identifiers. The names must be composed of ASCII characters, but can include any ASCII characters (such as commas, dashes, and space characters).
- Timetables that have `datetime` vectors as row times.

The supported functions for tables are:

- `addvars`
- `convertvars`
- `intersect`
- `ismember`
- `movevars`
- `removevars`
- `setdiff`
- `setxor`
- `union`
- `unique`

The supported functions for timetables are:

- `addvars`
- `cat`
- `convertvars`
- `horzcat`
- `intersect`
- `ismember`
- `movevars`
- `removevars`
- `retime`
- `setdiff`
- `setxor`
- `synchronize`
- `union`
- `unique`

For more information, see [Code Generation for Tables \(Simulink\)](#) and [Code Generation for Timetables \(Simulink\)](#).

Expanded code generation for categorical arrays

In R2020b, code generation supports more MATLAB toolbox functions when you use categorical arrays.

The supported functions for categorical arrays are:

- `addcats`
- `cat`
- `countcats`
- `eq`
- `ge`
- `gt`
- `horzcat`
- `histcounts`
- `intersect`
- `ismember`
- `le`
- `lt`
- `max`
- `mervecats`
- `min`
- `ne`
- `removecats`

- `renamecats`
- `reordercats`
- `setcats`
- `setdiff`
- `setxor`
- `union`
- `unique`
- `vertcat`

For more information, see [Code Generation for Categorical Arrays \(Simulink\)](#).

Code generation for more MATLAB functions

- `besselh`
- `besselk`
- `bessely`
- `getenv`
- `setenv`
- `matches`
- `pcg`
- `ordeig`
- `im2gray`
- `cmap2gray`

Code generation for more toolbox functions

5G Toolbox

- `generate`
- `networkTrafficFTP`
- `networkTrafficOnOff`
- `networkTrafficVoIP`
- `nrOFDMDemodulate`
- `nrOFDMInfo`
- `nrOFDMModulate`
- `nrPRACHOFDMInfo`
- `nrPRACHOFDMModulate`
- `nrResourceGrid`
- `nrTBSS`
- `nrULSCHDemultiplex`
- `nrULSCHMultiplex`
- `pcapngWriter`

- pcapWriter
- write
- writeCustomBlock
- writeGlobalHeader
- writeInterfaceDescriptionBlock

Audio Toolbox

- acousticFluctuation
- audioDelta
- audioFeatureExtractor
- cepstralCoefficients
- vggish
- yamnet

Antenna Toolbox

- comm.Ray

Communications Toolbox

- arrayConfig
- bleAngleEstimate
- bleAngleEstimateConfig
- blePCAPWriter
- bluetoothFrequencyHop
- comm.ChannelFilter
- comm.Ray
- comm.RayTracingChannel
- getElementPosition
- getNumElements
- getPayloadLength
- getPhyConfigProperties
- nextHop
- pcapngWriter
- pcapWriter
- write of blePCAPWriter
- write of pcapWriter and pcapngWriter
- writeCustomBlock
- writeGlobalHeader
- writeInterfaceDescriptionBlock

Computer Vision Toolbox

- `estimateGeometricTransform2D`
- `estimateGeometricTransform3D`
- `insertObjectMask`
- `pabin`
- `ransac`

DSP System Toolbox

- `getFrequencyVector`

Fixed-Point Designer

- `nnz`

Image Processing Toolbox

- `poly2label`
- `rgbwide2ycbcr`
- `ycbcr2rgbwide`

Navigation Toolbox

- `dynamicCapsuleList` and all its object functions
- `dynamicCapsuleList3D` and its object functions `addEgo`, `addObstacle`, and `checkCollision`
- `gnssSensor`
- `plan` of `plannerAStarGrid`
- `plannerAStarGrid`
- `referencePathFrenet` and all its object functions
- `stateSpaceSE3`
- `trajectoryGeneratorFrenet` and its object function `connect`
- `validatorOccupancyMap3D`
- `wheelEncoderAckermann`
- `wheelEncoderBicycle`
- `wheelEncoderDifferentialDrive`
- `wheelEncoderUnicycle`

Optimization Toolbox

- `fsolve`
- `lsqcurvefit`
- `lsqnonlin`

Robotics System Toolbox

- `analyticalInverseKinematics`

- gazebogenmsg
- generateIKFunction
- interpolate
- manipulatorRRT
- packageGazeboPlugin
- plan
- shorten

Sensor Fusion and Tracking Toolbox

- copy
- geoTrajectory
- initsingerekf
- singer
- singerjac
- singermeas
- singermeasjac
- singerProcessNoise
- stateinfo
- trackerGridRFS

Signal Processing Toolbox

See “C/C++ Code Generation Support: Generate code for feature extraction, signal measurements, and vibration analysis” (Signal Processing Toolbox).

Wavelet Toolbox

- ewt
- iswt
- iswt2
- scaleSpectrum
- swt
- swt2
- timeSpectrum
- wcoherence
- wdenoise2

WLAN Toolbox

- generate
- getNumPostFECPaddingBits
- interprethESIGBCommonBits
- interprethESIGBUserBits

- networkTrafficFTP
- networkTrafficOnOff
- networkTrafficVoIP
- pcapngWriter
- pcapWriter
- scramblerRange
- wlanInterpretScramblerState
- wlanNonHTDataBitRecover
- wlanNonHTOFDMDemodulate
- write
- writeCustomBlock
- writeGlobalHeader
- writeInterfaceDescriptionBlock

Modeling Guidelines

High-Integrity System Modeling guidelines to verify compliance with DO, IEC, EN, and ISO safety standards

This table identifies modified modeling guidelines. For a complete list of High-Integrity System Modeling guidelines, including their applicable Model Advisor checks, see Model Advisor Checks for High-Integrity Modeling Guidelines.

High-Integrity Modeling Guideline	Rationale
hisl_0013: Usage of data store blocks	<p>These conditions are removed:</p> <ul style="list-style-type: none"> • Avoid data store reads and writes that occur across model and atomic subsystem boundaries • Avoid using data stores to write and read data at different rates, because different rates can result in inconsistent exchanges of data. To provide deterministic data coupling in multirate systems, use Rate Transition blocks before Data Store Write blocks, or after Data Store Read blocks. <p>The Notes section is enhanced to include information about:</p> <ul style="list-style-type: none"> • Deterministic data transfer between different rates and tasks • Using Simulink Design Verifier to more accurately detect data store memory access violations <p>This change does not affect Model Advisor check Check safety-related diagnostic settings for data store memory (Simulink Check), which verifies that the Data Store Memory configuration parameters are set to <code>error</code> as defined by the guideline.</p>

S-Functions

S-Function Builder changes

In 2020b, S-Function Builder has an improved UI to streamline workflow. Changes include:

- Embedded code editor for viewing function signatures and editing code.
- Improved layout for managing ports, parameters, libraries, and settings.
- Toolstrip view for easier discoverability of most-used actions, such as **Insert Ports**, **Save**, and **Build**.

See [Build S-Functions Automatically Using S-Function Builder](#) for more information. For examples of how to use S-Function Builder, see [Model a State-Space System Using S-Function Builder](#) and [Use a Bus Signal with S-Function Builder to Create an S-Function](#).

R2020a

Version: 10.1

New Features

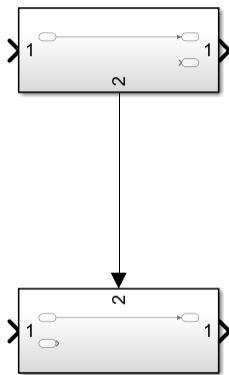
Bug Fixes

Version History

Simulink Editor

Port enhancements

- **Flexible port placement** — You can now put ports on any side in any order on these blocks:
 - Model block
 - Subsystem block
 - Subsystem Reference block
 - Stateflow Chart
 - Stateflow Truth Table
 - Stateflow State Transition Table

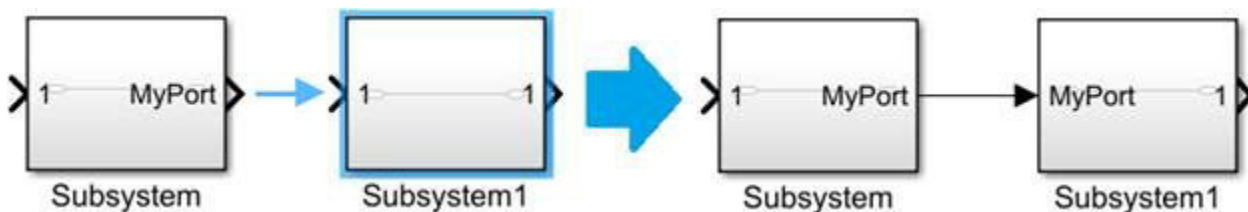


You can move ports by clicking and dragging the port. For Subsystem blocks, the port index will automatically be renumbered after each move.

- **Port hyperlinks in error messages** — Simulink now adds a hyperlink to the error and warning messages for most ports. Click the hyperlink to navigate to the port.
- **Port index as port label for port block with default names** — By default, Simulink now displays the port number instead of the block name when the port block has the default name.
- **Display port number** — You can now display the port number of port when you click a block. To enable this feature, in the **Debug** tab, click **Information Overlays** and select **Port Number**.

Port name propagation

Creating a connection between subsystem ports and/or port blocks now propagates a nondefault port name to the default named port. This applies to Simulink, System Composer™, and AUTOSAR subdomains. Default port names include **In**, **InBus**, and **Conn**.



Simplified line segment moving

In previous releases, when editing lines, line segments were routed to avoid obstacles such as other blocks and annotations. In R2020a, line segments do not avoid obstacles during routing to give you more control over line editing.

Improved in-canvas text editing

When editing a block or signal name, use **Enter** to commit the text and **Shift+Enter** to insert a line break.

Put the block name on any side of the block

Use the new block parameter `NameLocation` to put the name on any of the four sides of a block. You can continue to use the old parameter `NamePlacement`. Use `set_param` and `get_param` with `NameLocation` and the options `'right'`, `'left'`, `'top'`, and `'bottom'` to set the value.

```
set_param(gcb, 'NameLocation', 'right');
```

You can also move the name interactively. Select the block and drag the name around the block to the desired location.

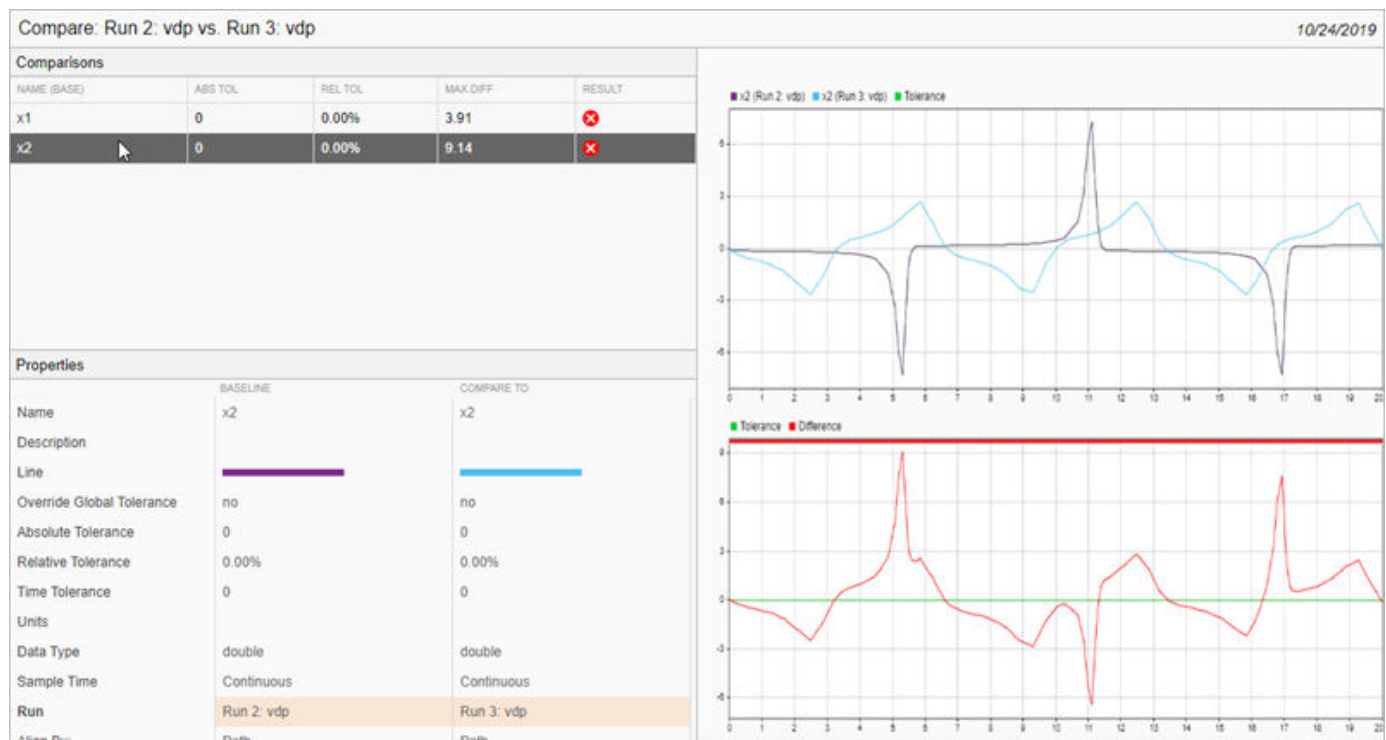
For more information, see `set_param` and `Format a Model`.

Simulation Analysis and Performance

Save Simulation Data Inspector comparison results in a shareable, interactive HTML report

You can create an interactive, web-based report to save and share the results of Simulation Data Inspector comparisons. The report is a self-contained HTML file, so viewing the report does not require any product license.

You can interact with the report similarly to how you interact with the results in the **Compare** tab of the Simulation Data Inspector. The plot area on the right displays the results for the signal you select in the **Comparisons** table, and you can scroll through the **Properties** information for the selected signal.



For more information, see [Create an Interactive Comparison Report](#).

Display message payload as port value labels

You can display message data as port value labels on the block diagram during or after a simulation.

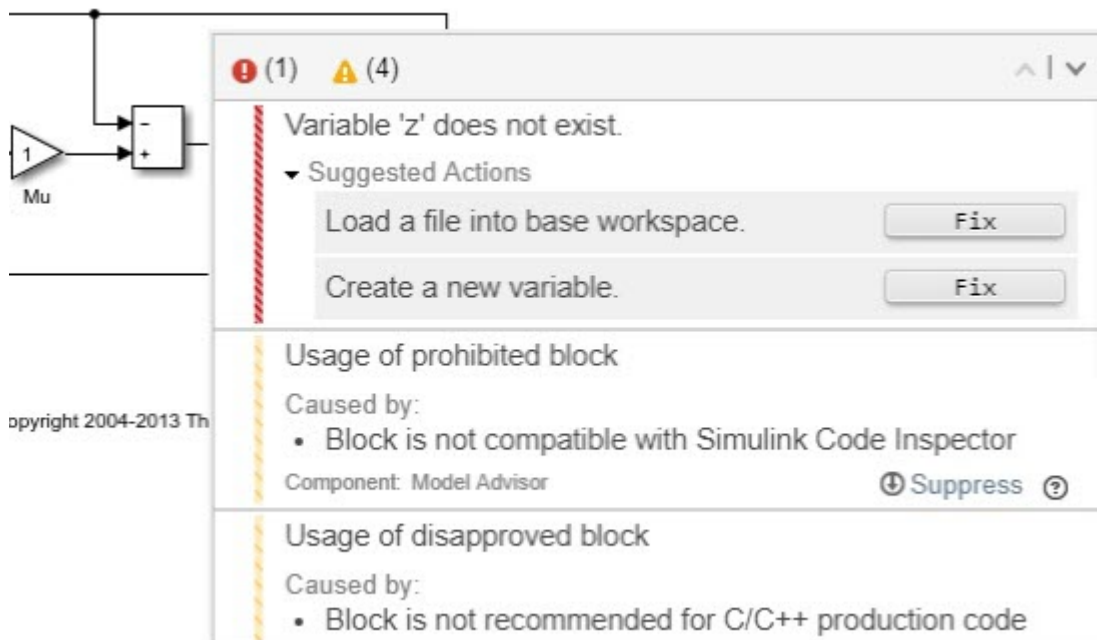
To display message data, right-click the message line emerging from a block and select **Show Value Label of Selected Port**, and click the data to be displayed.

For more information, see [Animate and Understand Sending and Receiving Messages](#).

Filter and scan errors and warnings during edit-time in diagnostic viewer.

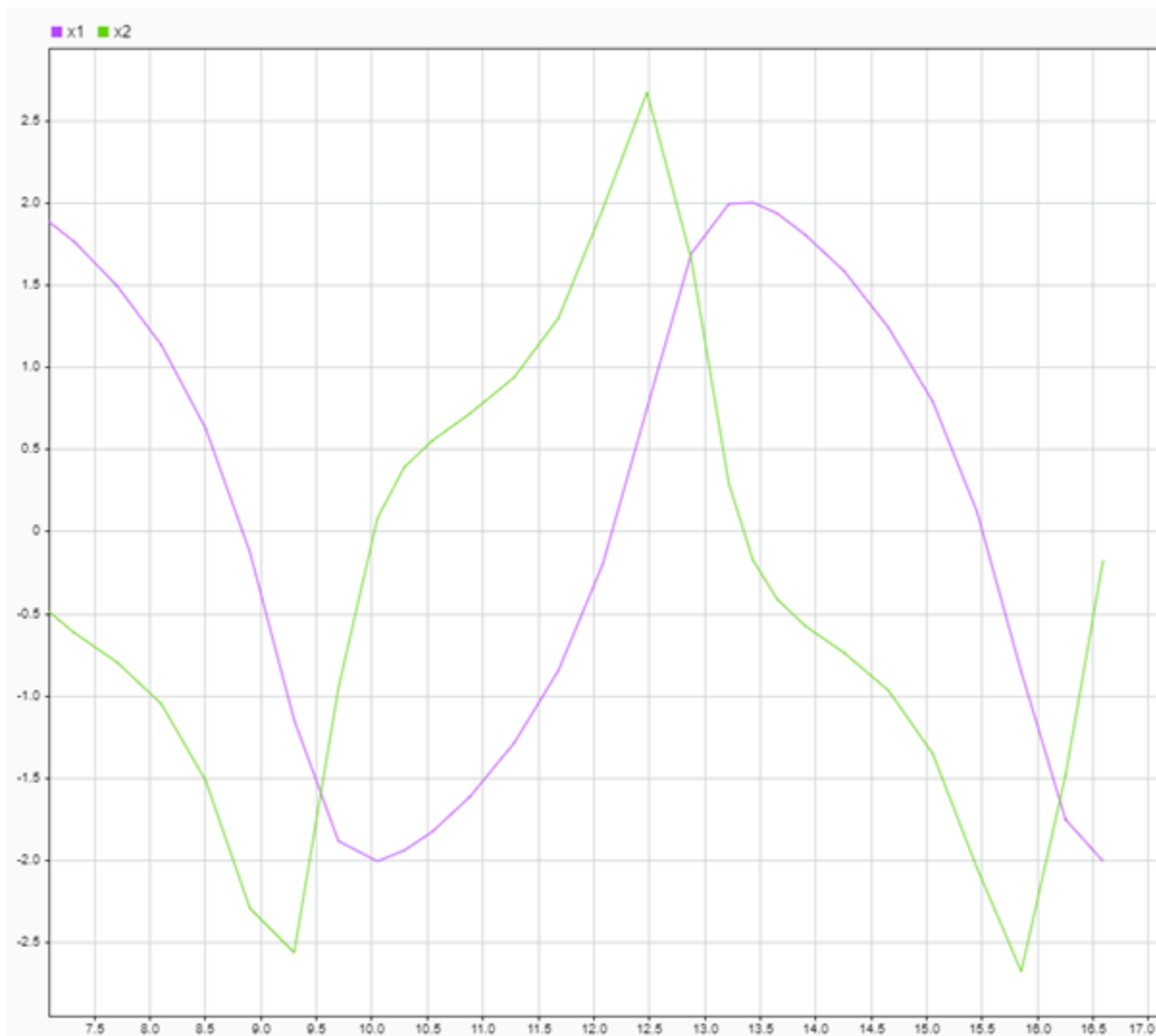
The edit-time diagnostic widget now includes the following features:

- Diagnostic errors and warnings are categorized and shown in tabs.
- All edit-time errors and warnings for a block are shown in a vertical, scrollable list.
- The diagnostic widget is displayed in a transient window.



View streamed data on a scrolling display in the Simulation Data Inspector

You can configure time plots in the Simulation Data Inspector to scroll when plotted data overruns the time span of the plot. Set the **Update mode** to **Scroll** in the **Time Plot Settings**.



Simulation Data Inspector programmatic interface enhancements

R2020a includes several enhancements to the Simulation Data Inspector programmatic interface, including new functions that facilitate accessing logged data:

- `Simulink.sdi.Run.getLatest` returns the most recently created run in the Simulation Data Inspector repository.
- `Simulink.sdi.getCurrentSimulationRun` returns the `Simulink.sdi.Run` object that corresponds to the in-progress or most recently completed simulation of the specified model.
- `Simulink.sdi.getCurrentComparison` returns the `Simulink.sdi.DiffRunResult` or `Simulink.sdi.DiffSignalResult` object with results for the most recently performed comparison.
- `getSignalsByName` returns one or more signals with the specified name within a `Simulink.sdi.Run` object.
- `getSignalIDsByName` returns one or more signal IDs for signals with the specified name within a `Simulink.sdi.Run` object.

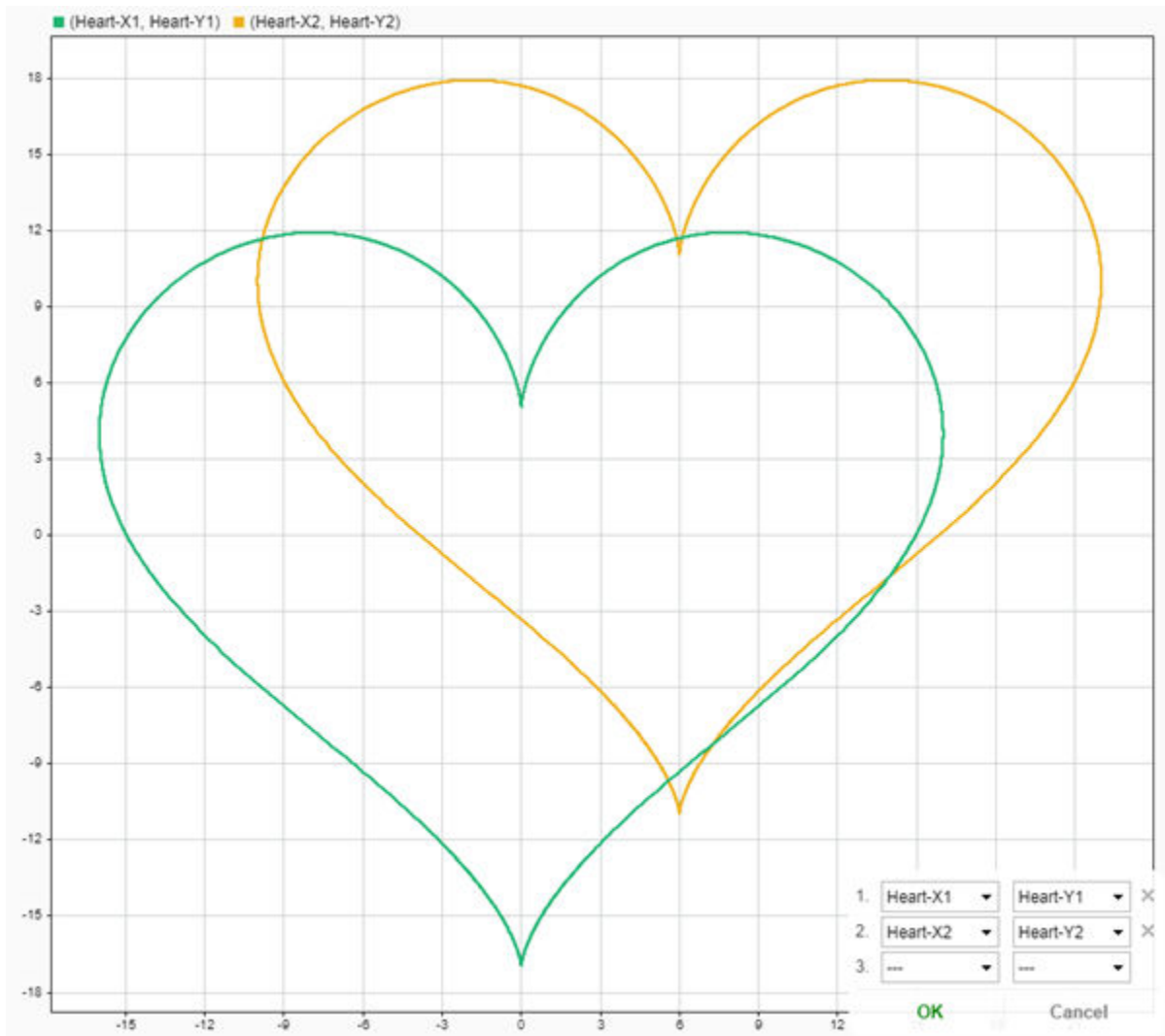
The `Simulink.sdi.Run` and `Simulink.sdi.Signal` objects are also enhanced to provide more information:

- `Simulink.sdi.Run` objects now have a `RunIndex` property that indicates the index of the run within the Simulation Data Inspector.
- `Simulink.sdi.Signal` objects have these additional properties:
 - `BlockName` — Name of the block that produced the signal
 - `OverrideGlobalTol` — Whether signal tolerance values override global tolerance values for comparisons
 - `Children` — Vector of `Simulink.sdi.Signal` objects for signals at the next level of hierarchy in nonscalar signals, such as matrix, vector, or bus signals

You can also programmatically configure the **Input Processing** property for logged signals using the `Simulink.sdi.setSignalInputProcessingMode` function or check the property value using the `Simulink.sdi.getSignalInputProcessingMode` function.

View and inspect multiple series on an XY visualization in the Simulation Data Inspector

You can plot and analyze multiple series on XY visualizations in the Simulation Data Inspector. When you select signals to plot on the XY visualization, a window appears in the lower-right of the plot, where you can specify x-y signal pairs for each series you want to plot.

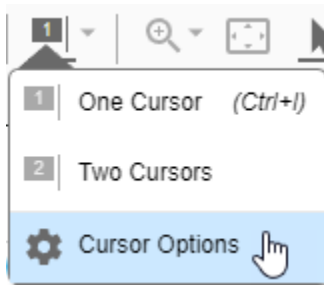


The **XY Settings** apply to all plotted series. For example, when you select the **Trend line** option, a trend line appears for each series. When you add cursors to the XY plot, each series gets its own cursor, and all cursors on the plot remain synchronized in time.

For more information, see Visualize Simulation Data on an XY Plot.

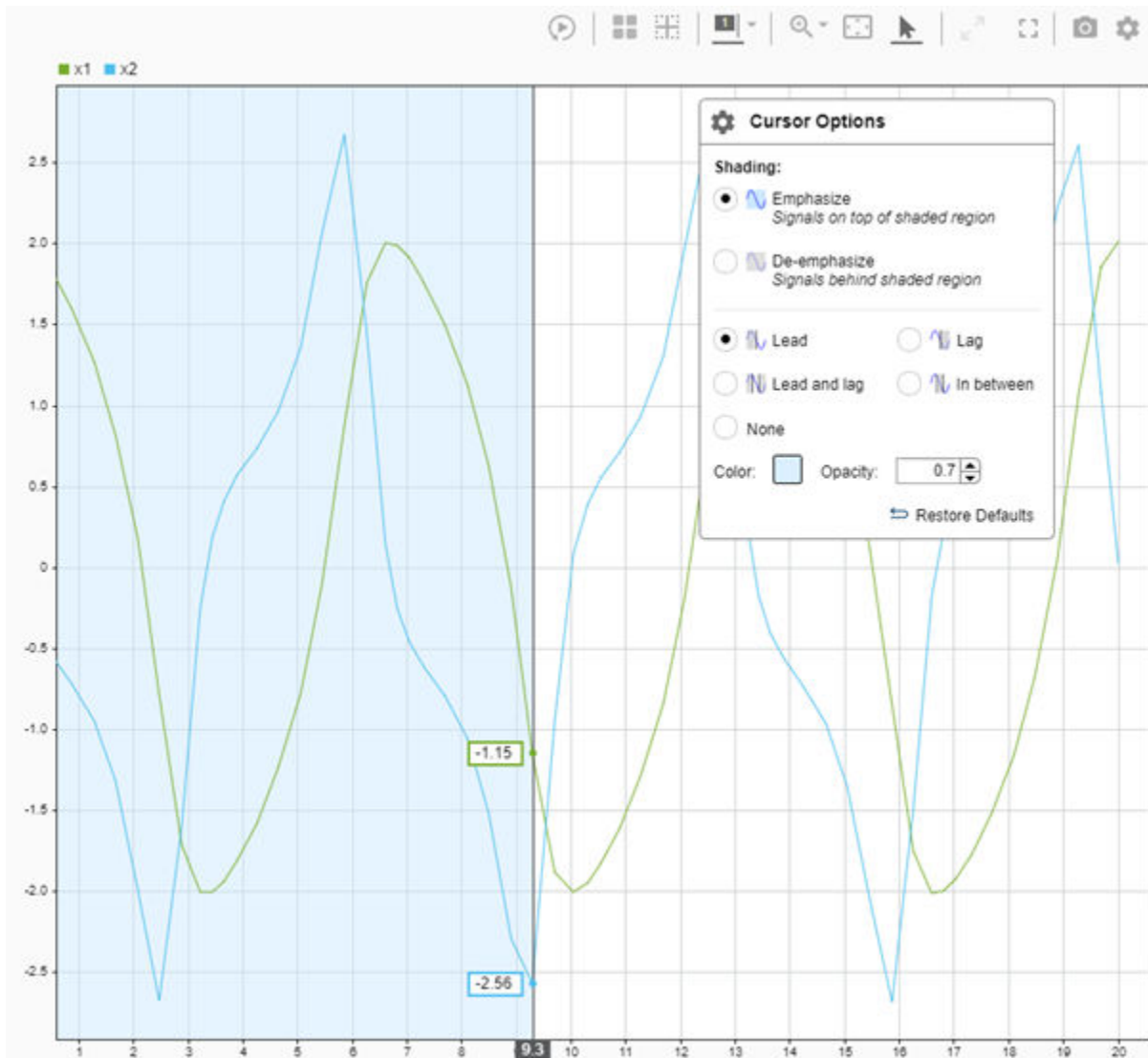
Shade signal regions using cursors in the Simulation Data Inspector

You can use shading to draw attention to signal regions in plots in the Simulation Data Inspector using new cursor options. To configure the behavior and appearance of the shading, open the **Cursor Options** from the cursors menu.

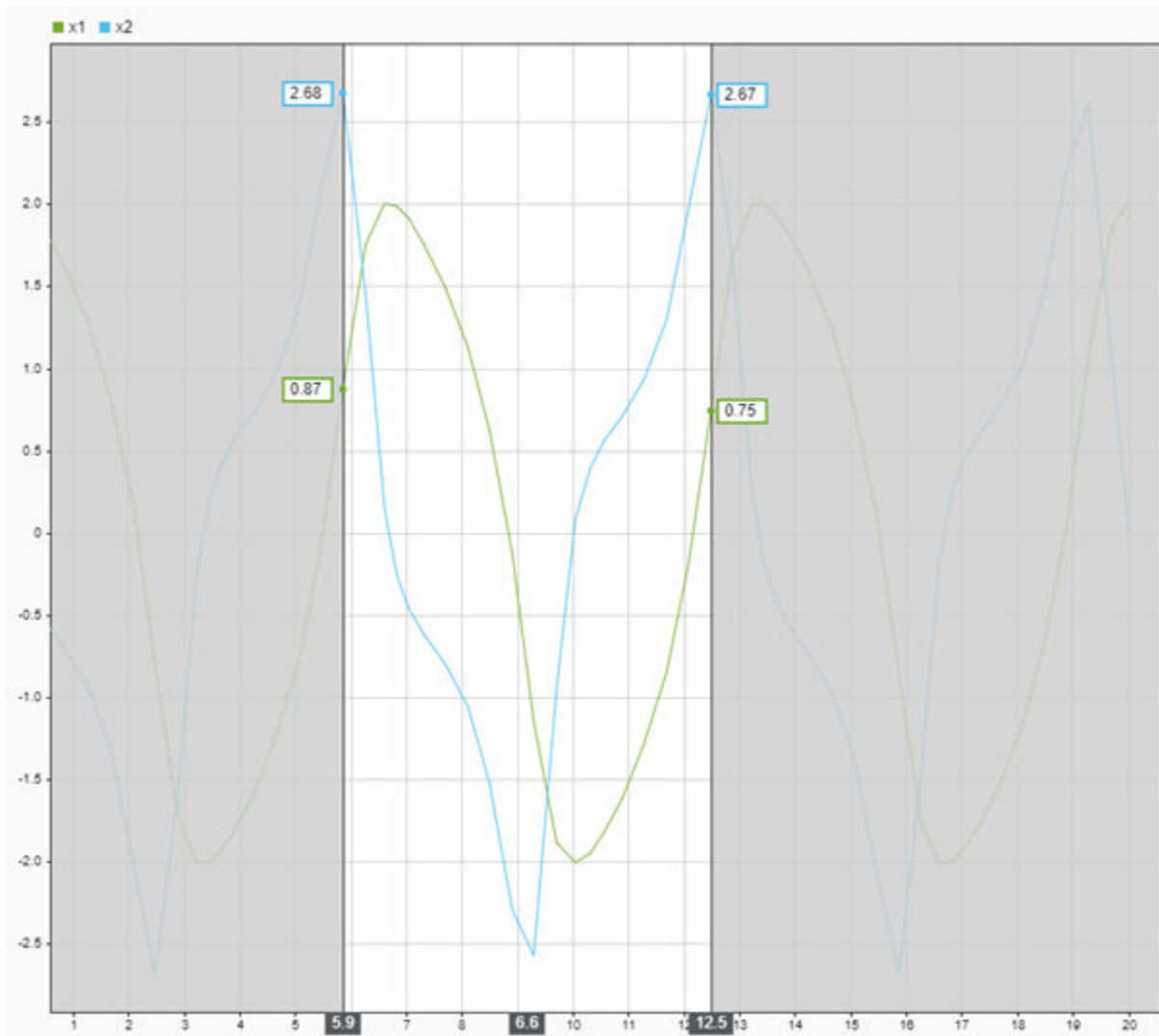


Use the **Cursor Options** window to configure whether the shading emphasizes or de-emphasizes the signal in the shaded region, select a region to shade, and specify the color and opacity of the shading.

For example, the shading in this plot emphasizes the signal in the plot area that leads the cursor.



The shading in this plot de-emphasizes the signal to focus attention on a single period of the signal.



You can configure the cursor options independently for the **Inspect** and **Compare** panes.

For more information, see [Create Plots Using the Simulation Data Inspector](#).

Configure programmatic Simulation Data Inspector comparisons and save comparison results

You can use the `Simulink.sdi.compareRuns` function to configure signal metadata to compare and end the comparison at the first detected metadata mismatch or out-of-tolerance signal. You can configure the comparison to check whether signal data types, sample times, and start and stop times match.

The `Simulink.sdi.DiffRunResult` object is also enhanced to provide more information about the comparison configuration and results. The `Simulink.sdi.DiffRunResult` object has these additional properties:

- `Options` — Array of metadata comparison options specified for the run comparison

- **Status** — Run comparison status
- **StopReason** — `Simulink.sdi.DiffSignalResult` object for the signal comparison that caused the run comparison to stop

The `saveResult` function saves a `Simulink.sdi.DiffRunResult` object to an MLDATX-file you can load into the Simulation Data Inspector.

Enhanced Simulation Data Inspector support for parallel computing

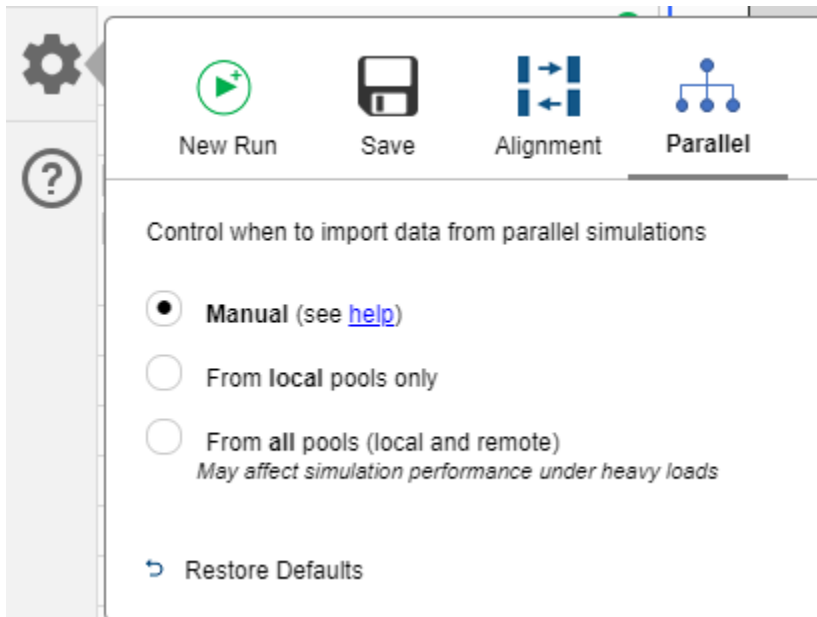
In the **Inspect** tab, a color-coded circle next to each run name indicates the run status, which allows you to monitor the progress of a set of parallel simulations.



The colored circle indicates one of these statuses:

- Green — The simulation that created the run has completed successfully.
- Blue — The simulation that created the run is in progress.
- Orange — The simulation that created the run is paused.
- Red — The simulation that created the run encountered an error.
- No circle — The run was created from imported data and does not have a simulation status to report.

You can also control when data automatically streams from parallel simulations in the Simulation Data Inspector **Preferences**. By default, the Simulation Data Inspector is configured to support manually importing data logged in parallel simulations. For more information, see `Simulink.sdi.enablePCTSupport`.



Enhanced task-based sorting algorithm for determining block execution order

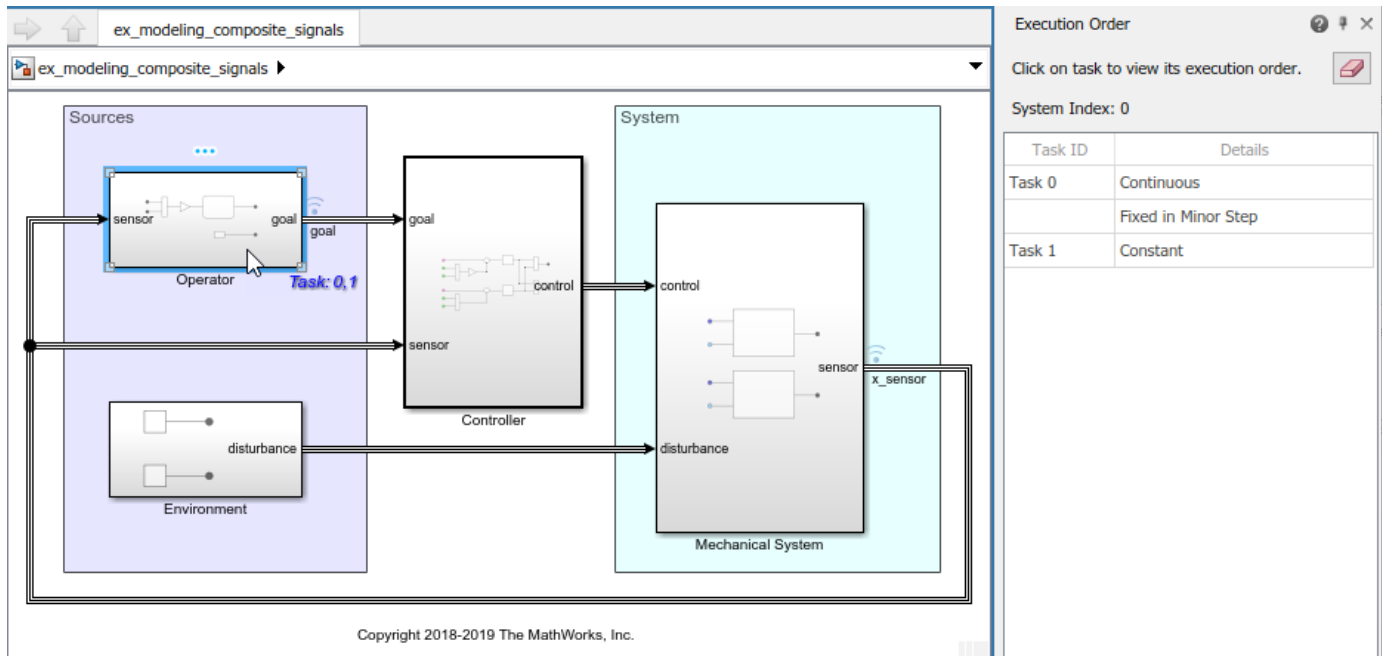
Simulink now has an enhanced algorithm for task-based sorting to support models with asynchronous rates.

In R2020a, task-based sorting does not support the union of multiple asynchronous sample times. Models containing the union of multiple asynchronous rates will default to the global execution order.

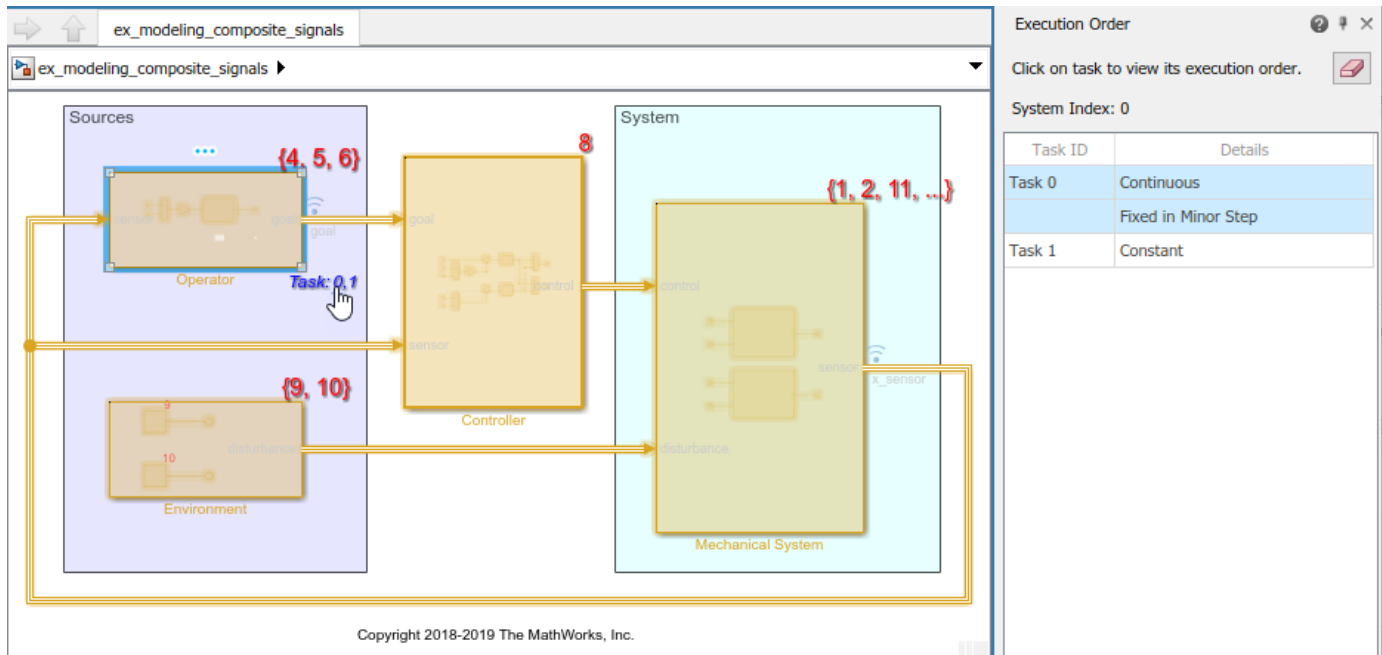
For models created in previous versions, the block execution order may change when task-based sorting is on. For more information, see [Check for Execution Order Changes Involving Data Store Memory Blocks](#).

Navigate from blocks to tasks in the Execution Order viewer

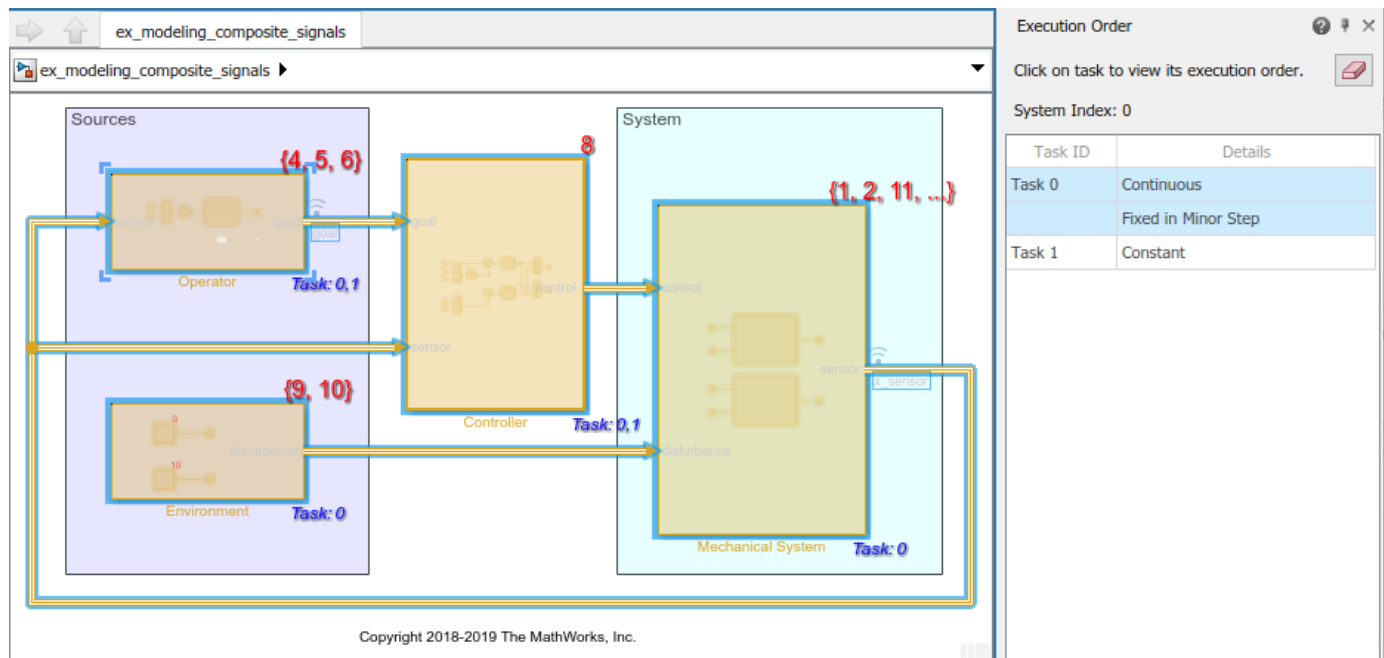
To display the tasks in which a block executes, you can now click on a block.



To display the execution order for a task, you can click one of the task numbers in the label. When a model has many tasks, these links provide an alternative to scrolling to the task and clicking it in the Execution Order pane.



To see the tasks in which multiple blocks execute, select multiple blocks.



For more information on task-based sorting and execution order, see [Control and Display Execution Order](#).

Updated Simulink Profiler

You can view and interact with the Simulink Profiler from within the Simulink Editor.

Launch the Profiler from the **Debug** tab of the toolstrip. Select **Performance Advisor > Simulink Profiler** and click **Profile** to profile the model.

Anti-Windup PID Control Example

Copyright 2009-2020 The MathWorks, Inc.



Profiler Report (sldemo_antiwindup @ 21-Dec-2020 12:39:31)

Path	Time Plot (Dark Band = Self Time)	Total Time (s)	Self Time (s)	Number of Calls
sldemo_antiwindup		4.247	1.198	18013
Scope		2.398	2.398	6004
Scope1		0.642	0.642	6004
PID Controller		0.004	0.000	0
Filter		0.001	0.000	0
Integrator		0.001	0.000	0
D Gain		0.000	0.000	0
I Gain		0.000	0.000	0
N Gain		0.000	0.000	0

Self Time (dark band): 0.000
Total Time: 0.001

Exec Node	Time (s)	Calls
simulationPhase	0.767	1
initializationPhase	0.338	1
terminationPhase	0.034	1
sldemo_antiwindup...	0.019	1
sldemo_antiwindup...	0.014	1
solverPhase	0.007	2000
sldemo_antiwindup...	0.006	2001
simulate(sldemo_a...	0.004	1
sldemo_antiwindup...	0.003	4000
ode3.Integrate	0.002	2000
sldemo_antiwindup...	0.002	1
sldemo_antiwindup...	0.001	5999
sldemo_antiwindup...	0.001	2001
sldemo_antiwindup...	0.001	1
sldemo_antiwindup...	0.000	1
reset	0.000	1
sldemo_antiwindup...	0.000	1
sldemo_antiwindup...	0.000	1

You can navigate to a block of interest in the editor by clicking on its row in the **Profiler Report** table.

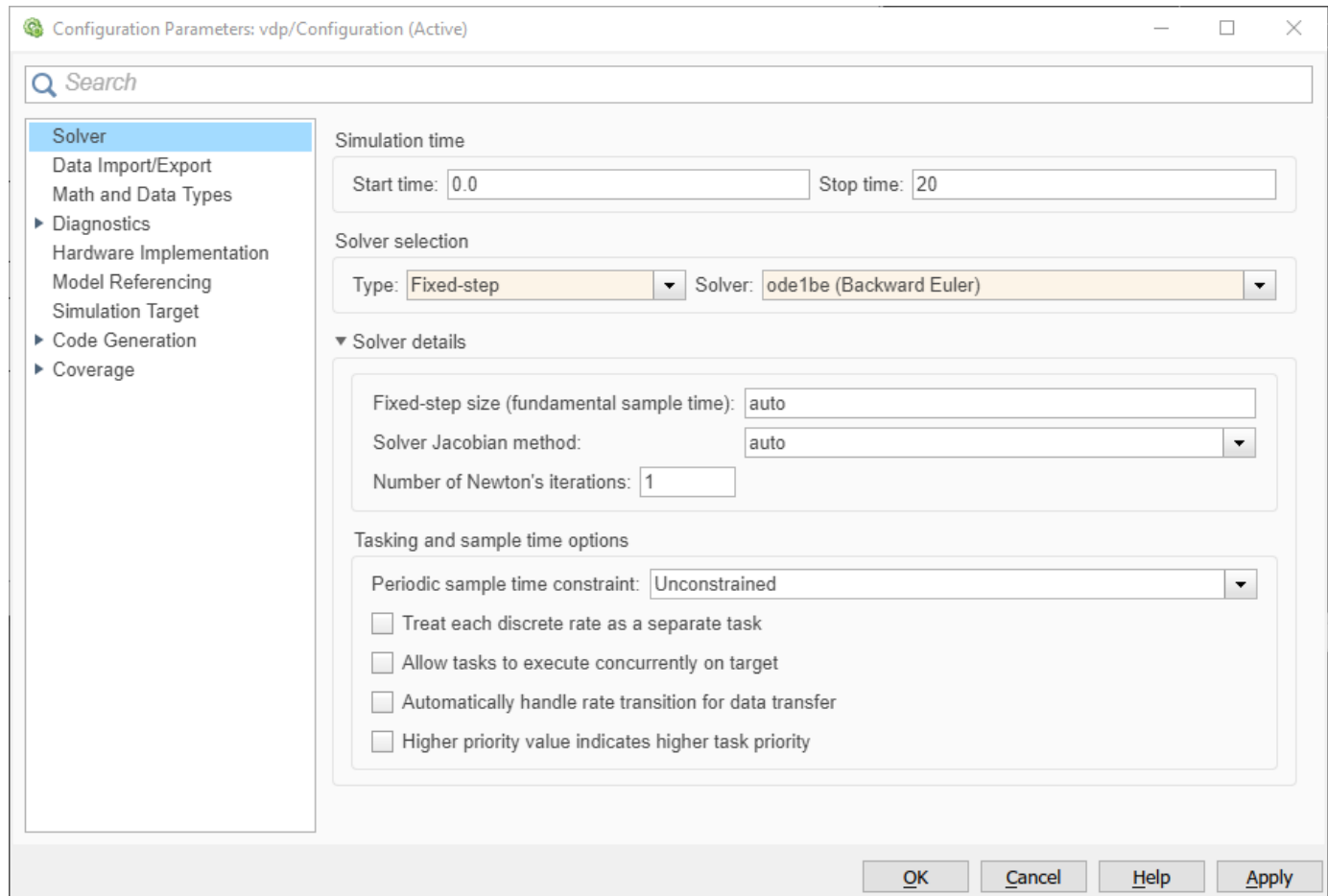
Switch to the model hierarchy view clicking , and switch to the execution view by clicking .

ode1be implicit fixed step solver

The ode1be solver is a Backward Euler type solver that uses a fixed number of Newton iterations, and incurs only a fixed cost. As such, ode1be supports real-time code generation for fixed-step targets.

You can use the ode1be solver as a computationally inexpensive fixed-step alternative to the ode14x solver.

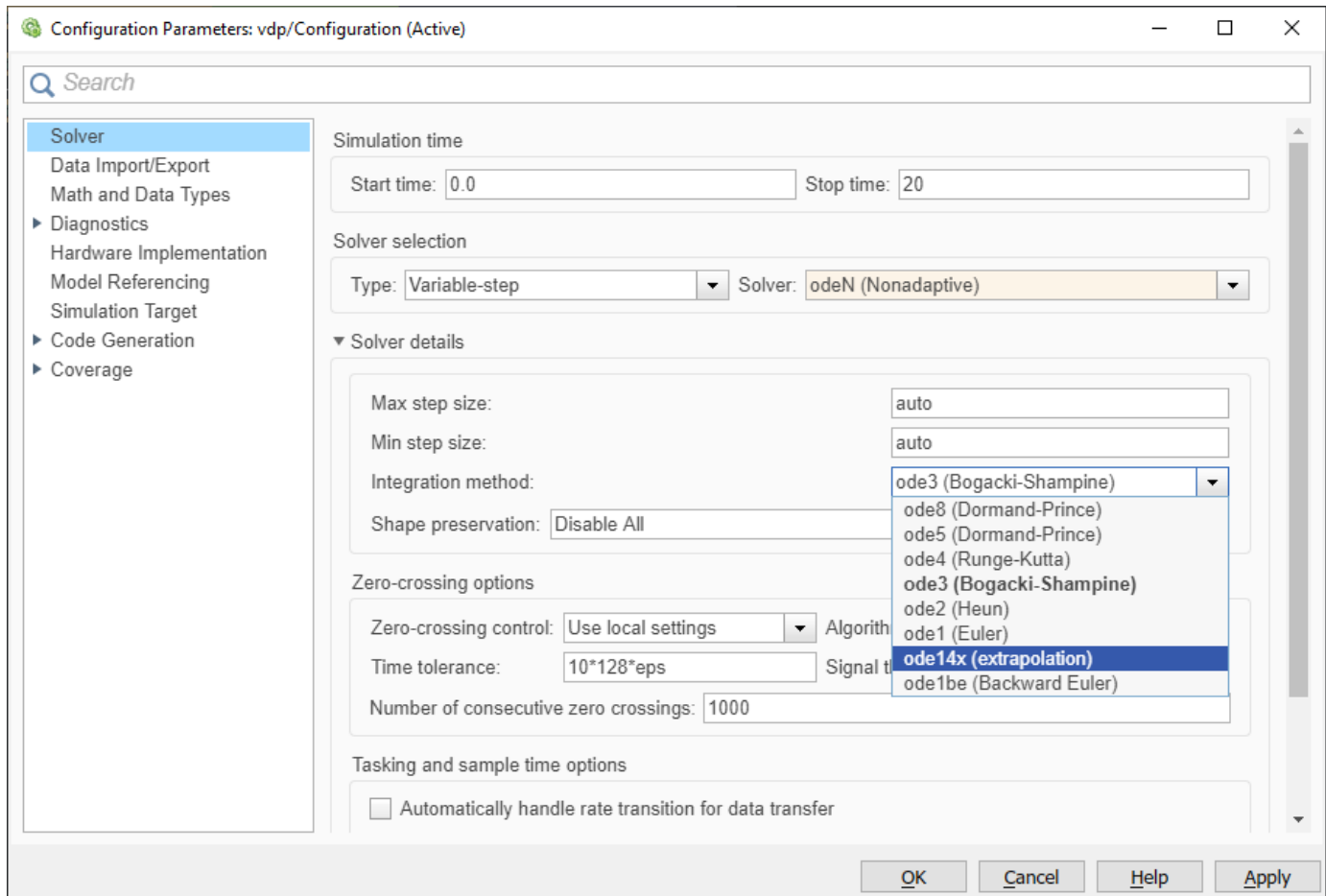
To select the ode1be solver, select Fixed-Step for **Type** in the **Solver Selection** section of the **Solver** pane of the model Configuration Parameters.



Implicit solver support for odeN solver

The odeN solver now includes two implicit Integration methods—the ode14x and ode1be solvers—to allow you to simulate stiff systems, Simscape models, and implicit system dynamics. As with the other integration methods of the solver, use odeN for models where the bottleneck for the step size is either a very fast discrete rate, or when there are many zero-crossing events. In this type of model, the error-control mechanism of other solvers is rarely used, which makes odeN more efficient.

To use ode14x or ode1be in your model, in the Configuration Parameters dialog box, in the Solver pane, under **Solver details**, select one of the new solvers under **Integration Method**.



Note In previous releases, you set the integration method of the `odeN` solver by using `set_param(model, 'SolverOrder', <solverorder>)`, where `<solverorder>` is an integer. The `SolverOrder` parameter has been renamed to `ODENIntegrationMethod` in order to accommodate the implicit `ode14x` and `ode1be` solvers.

You can still use `SolverOrder` to set the integration orders 1-5 and 8. However, you can only enable `ode14x` or `ode1be` via the `ODENIntegrationMethod` parameter for `set_param`.

Project tab Tool-Coupling FMU option name change

The **Project** tab Tool-Coupling FMU option name has changed to Tool-Coupling FMU.

Intel C and C++ toolchain support for Windows

You can now use Intel C and C++ compilers to run simulations on Windows computers. For model referencing, accelerator mode, and rapid accelerator mode, R2020a supports:

- Intel Parallel Studio XE 2017 with Microsoft Visual Studio 2015, 2017
- Intel Parallel Studio XE 2018 with Microsoft Visual Studio 2015, 2017

- Intel Parallel Studio XE 2019 with Microsoft Visual Studio 2015, 2017, 2019

For more information, see Supported Compilers.

Functionality being removed or changed

odeN solver parameter renamed from SolverOrder to ODEIntegrationMethod

In previous releases, you set the integration method of the odeN solver using `set_param(model, 'SolverOrder', <solverorder>)`, where <solverorder> is an integer. The SolverOrder parameter has been renamed to ODEIntegrationMethod in order to accommodate the implicit ode14x and ode1be solvers.

You can still use SolverOrder to set the integration orders 1-5 and 8. However, you can only enable ode14x or ode1be via the ODEIntegrationMethod parameter for `set_param`.

Signal data no longer streams to the Simulation Data Inspector when signal logging is disabled

Behavior change

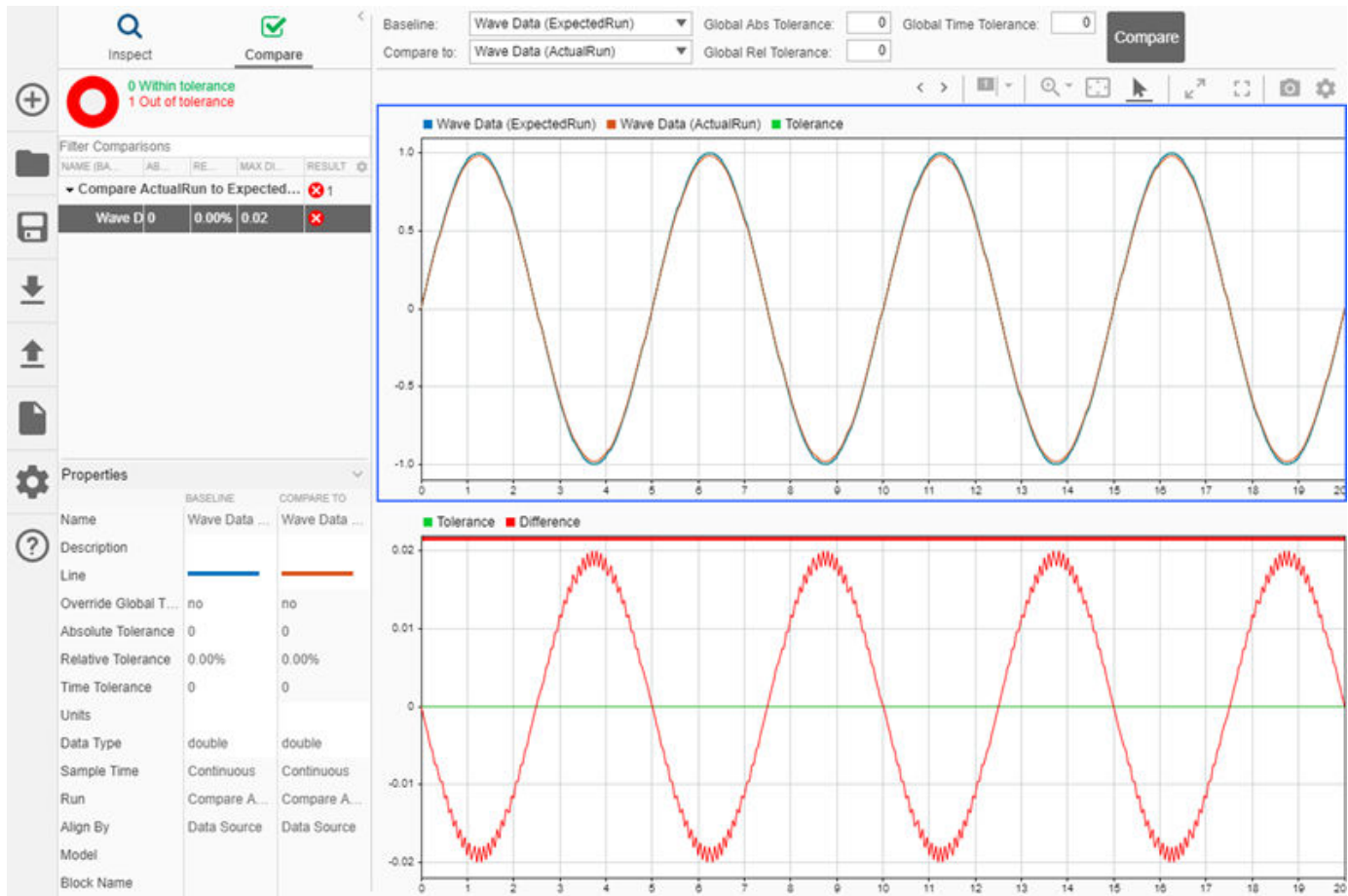
Since R2017a, signals marked for logging have streamed to the Simulation Data Inspector, regardless of the setting for **Signal logging** in the model configuration parameters. Starting in R2020a, signals marked for logging stream to the Simulation Data Inspector *only* when signal logging is enabled for a model.

To view data in the Simulation Data Inspector or access data using the Simulation Data Inspector programmatic interface, you must enable signal logging for the model. Open **Model Settings** from the toolbar, navigate to the **Data Import/Export** pane, and select **Signal logging**.

Simulink.sdi.constraints.MatchesSignal constraint provides enhanced diagnostic information

Behavior change

Starting in R2020a, when a test that uses the `Simulink.sdi.constraints.MatchesSignal` constraint fails, the diagnostic information provides a path and link to a Simulation Data Inspector session file that includes the comparison results and the original data compared by the test. Open the session file to inspect the data and comparison results in the Simulation Data Inspector.



In previous releases, the diagnostic information included a path and link to a static image of the comparison results. The diagnostic information no longer includes the image.

Simulink.sdi.enablePCTSupport function ignores logical inputs

Behavior change

Starting in R2020a, the `Simulink.sdi.enablePCTSupport` function ignores logical inputs. In code that specifies a logical input for the `Simulink.sdi.enablePCTSupport` function, replace a `0` or `false` input with the `'manual'` input option and a `1` or `true` input with the `'all'` option to achieve equivalent behavior.

Simulink.sdi.enablePCTSupport function no longer supports 'none' input option

Errors

Starting in R2020a, the `Simulink.sdi.enablePCTSupport` function no longer accepts the `'none'` input option. Use the `'manual'` option if you do not want data from parallel simulations to automatically import into the Simulation Data Inspector.

Invalid function-call connection configuration parameter has been removed

Errors

The **Invalid function-call connection** configuration parameter has been removed. In previous releases, it was possible to set the **Invalid function-call connection** configuration parameter to

give a warning when Simulink detects the incorrect use of a function-call subsystem. Starting in R2020a, the incorrect use of a function-call subsystem will result termination of the simulation and display of an error message.

Component-Based Modeling

Simulink Units: Create and load custom units

By default, Simulink supports only the units listed in Allowed Units. You can now create and load custom unit databases. For more information, see [Working with Custom Unit Databases](#).

Simulate referenced models with unsaved changes in accelerator mode

You can now simulate referenced models with unsaved changes in accelerator mode. These unsaved changes increase the time required to compile the model hierarchy and check whether model reference simulation targets are up to date. For information on the differences among model reference simulation modes, see [Choose Simulation Modes for Model Hierarchies](#).

Simulate protected models from a previous release in accelerator mode

Starting in R2020a, you can use accelerator mode to simulate models protected in R2018b or later. To simulate these protected models in accelerator mode:

- You must directly simulate the parent of the protected model. Simulating a model hierarchy that contains the parent model as a referenced model is not supported. To interactively simulate the parent model, you must open it as the top model in its own window. For more information, see [Navigate Model Hierarchies](#).
- The simulation must use Just-in-Time acceleration and must not require C-code generation. For more information on acceleration modes, see [How Acceleration Modes Work](#).

For more information on cross-release support for protected models, see [Reference Protected Models from Third Parties](#).

find_mdhrefs option to exclude top model in results

When using the `find_mdhrefs` function, you can now exclude the top model from the returned list of referenced models by setting `ReturnTopModelAsLastElement` to `false`.

Messages support fixed-point data type

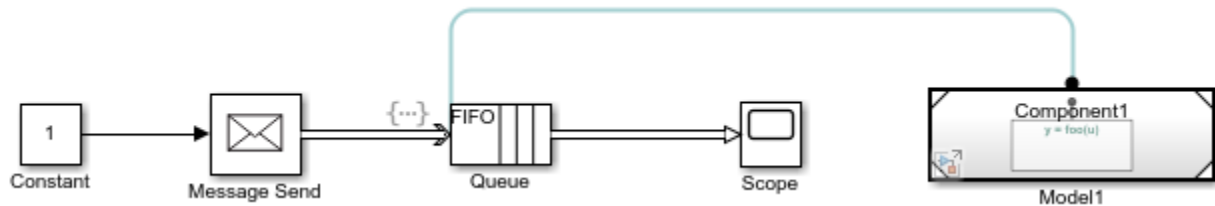
Messages support fixed-point data type. Using fixed-point data can reduce the memory requirements and increase the speed of code generated from a model. To learn more about the fixed-point data type, see [Specify Fixed-Point Data Types](#).

Event actions support calling a scoped Simulink Function block

Message blocks can call Simulink functions when the **Function visibility** parameter of the Simulink Function block is set to `scoped`. You can use the name of the subsystem or the Model block together with the function name to call the scoped Simulink function.

In this example, a Simulink Function block with the function signature `foo()` is in a Model block called `Model1`. The **Function visibility** parameter of the Simulink Function block is set to `scoped`.

When a message enters the Queue block, the Simulink function `foo()` is called by using the code `Model1.foo()` in the Queue block's entry event action.



For an example, see Model a Message Receive Interface that Runs on Message Availability.

Verify digital signatures on protected models

In R2020a, you can verify digital signatures on protected models. The author of a protected model can sign the model by using a certificate from a trusted certificate authority. Use digital signature verification to check that the protected model does not have any of the following issues:

- Changes made to the protected model after it was signed
- A signature from a publisher that does not use a certificate from a trusted certificate authority
- A signature with an expired or self-issued certificate

You can view the digital signature information of a protected model in the **Details** pane. You can also verify the model by using the two new functions `Simulink.ProtectedModel.getPublisher` and `Simulink.ProtectedModel.verifySignature`. For more information, see Reference a Protected Model.

Improved workflow of variants and masking

Starting in R2020a, `set_param` command in the **Initialization** tab of Mask Editor is improved and made more reliable. You can use the `set_param` to change the active variant of variant blocks.

Here is an example showing usage of `set_param` command:

```
set_param(vssblock, 'LabelModeActiveChoice', choice)
```

Before R2020a, using `set_param` command for changing active variant from mask caused inconsistent results and error messages.

Convert Configurable Subsystem blocks to Variant Subsystem blocks programmatically

Starting in R2020a, you can convert a Configurable Subsystem block in a model to a Variant Subsystem block programmatically using any of these syntaxes:

- `Simulink.VariantManager.convertToVariant(gcb)`
- `Simulink.VariantManager.convertToVariant(gcbh)`

Here is an example command:

```
open_system('sldemo_variant_subsystems');
```

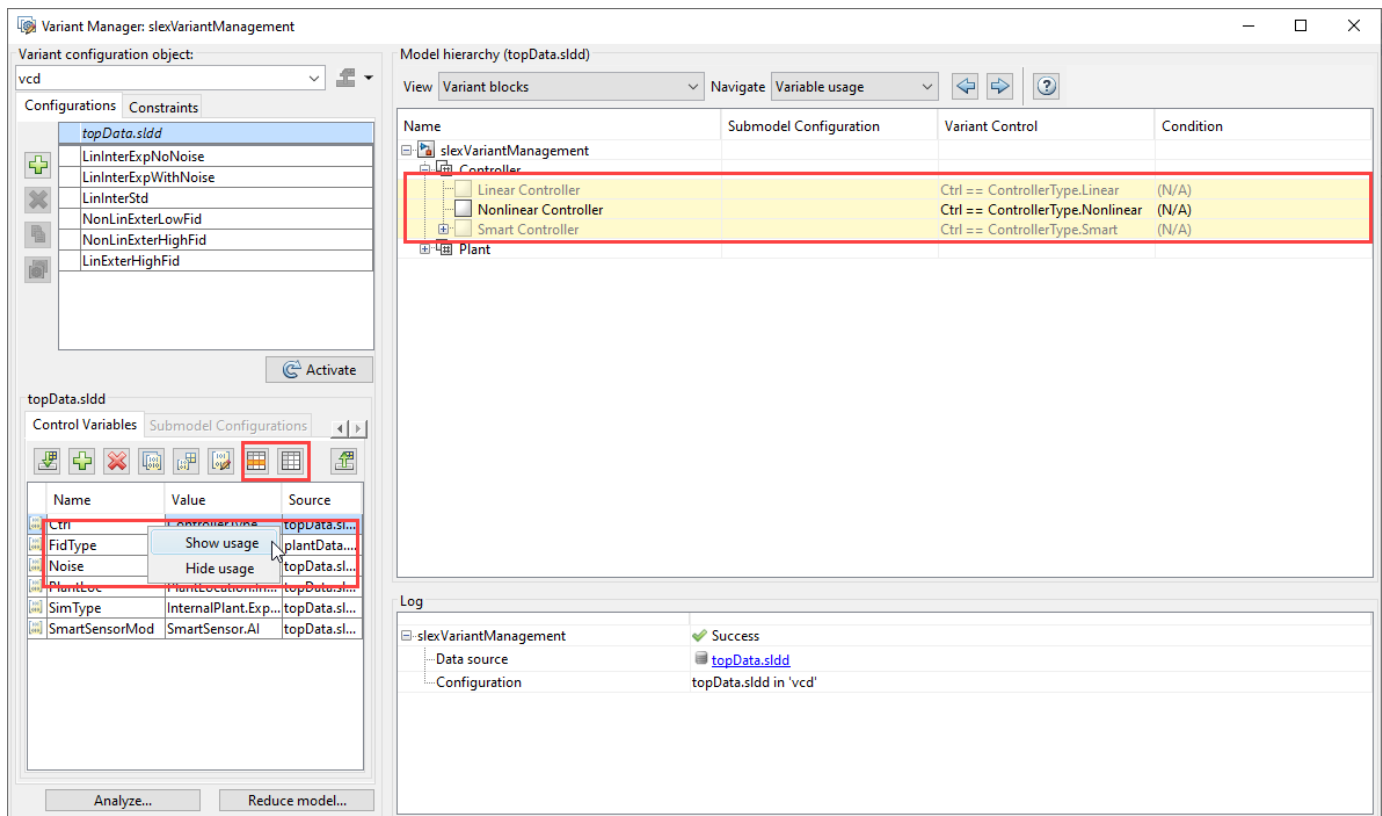
```
Simulink.VariantManager.convertToVariant('sldemo_variant_subsystems/  
Controller');
```

For more information on converting Configurable Subsystem blocks to Variant Subsystem blocks, see [Convert Configurable Subsystem Blocks to Variant Subsystem Blocks Programmatically](#)

Improved performance of Variant Manager: Control Variables usage changed to on-demand from automatic

Starting in R2020a, the variant control variable usage is not computed when launching Variant Manager or activating a variant configuration. It is instead computed on an on-demand basis, where you can now choose to show or hide the control variable usage in the model hierarchy table of the Variant Manager.

To do so, in the Variant Manager window, right-click the control variable and select **Show usage** or **Hide usage**. Alternatively, you can also click on the Show usage or Hide usage icons.



This results in improved performance while launching the Variant Manager or activating a variant configuration in the Variant Manager.

Before R2020a, the variant control variable usage was computed by default and highlighted in the model hierarchy while selecting a variant control variable.

Improved code generation with Variant Subsystem

Starting in R2020a, additional structures corresponding to block IO and Dworks are removed in the generated code for models containing Variant Subsystem. This reduces global memory usage and improves efficiency in generated code. In addition to this, comment generation is improved and pre-processor conditions (`#if`) are optimized in generated code. For information on preparing variant-containing model for code generation, see [Prepare Variant-Containing Model for Code Generation](#)

C++ code generation with Inline Variants

Starting in R2020a, C++ code generation is enabled for Inline Variant blocks such as Variant Source, Variant Sink, Simulink Functions, Initialize Function, Reset Function, and Terminate Function blocks. The parameters with their corresponding variant conditions are guarded in the generated code of Simulink models containing inline Variant blocks. For information on preparing variant-containing model for code generation, see [Prepare Variant-Containing Model for Code Generation](#).

Support for C++ class interfaces for explicitly partitioned models

Models configured for concurrent execution that use explicit partitioning can now generate C++ class interfaces. In past releases, explicitly partitioned models could only generate non-reusable code interface packaging.

To generate code with C++ class interfaces, in the **Configuration Parameters > Code Generation** pane, set **Language** to C++. In the **Code Generation > Interface** pane, set **Code interface packaging** to C++ class.

Code generation requires a Simulink Coder or an Embedded Coder license.

Subsystem reference enhancements

Subsystem reference is improved with the following enhancements and usability improvements:

- In Simulink Test, you can create a test harness for Subsystem Reference blocks.
- You can change the subsystem file referenced by a Subsystem Reference block in your model initialization callback.

For more information on subsystem reference, see [Subsystem Reference](#).

Search, sort, and group items in the forwarding table dialog

Forwarding table dialog now allows you to:

- Sort column values in ascending or descending order.
- Group columns by their values.
- Search for any text inside the table entries by using the search filter below the table.

For more information on forwarding tables, see [Forwarding Tables](#).

add_block command uses forwarding table to add appropriate block

Starting in 2020a, the `add_block` command looks for a forwarding table and uses the mapping in the table to add the current library block.

For more information on `add_block`, see `add_block`.

To know more about the functions of a Forwarding Table, see Forwarding Tables.

Schedule Editor API: Programmatically perform operations on the schedules of the model

In R2020a, you can programmatically perform operations on the schedules of the models created in the Schedule Editor. The `Schedule` objects can be created from a model with using the new `Schedule` parameter, modified, and applied back to the model. The API includes a `Schedule` object that contains an editable table. The `Schedule` object contains the execution order of the model and other information such as partition names, rates, type of partitions and the index. You can use the properties of the `Schedule` object to modify the schedule and apply to the model. The API also includes a `RateSection` property that you can use to modify the execution order of the partitions of the same rate.

Functionality being removed or changed

Converting subsystems to models no longer creates `Simulink.Bus` objects for virtual buses *Behavior change*

When you convert a subsystem to a referenced model, the conversion minimizes the number of `Simulink.Bus` objects it creates for virtual bus inputs and outputs. In `Bus Element` and `Out Bus Element` blocks allow virtual buses to cross the model boundary without `Bus` objects.

Control of execution context has been removed

The **Execution context** information overlay has been removed from the Simulink Toolstrip. When you simulate a model from an earlier release, the **Execution context** information overlay will not be visible.

The block parameter `Propagate execution context across subsystem boundary` has been removed for conditionally executed `Subsystem` blocks.

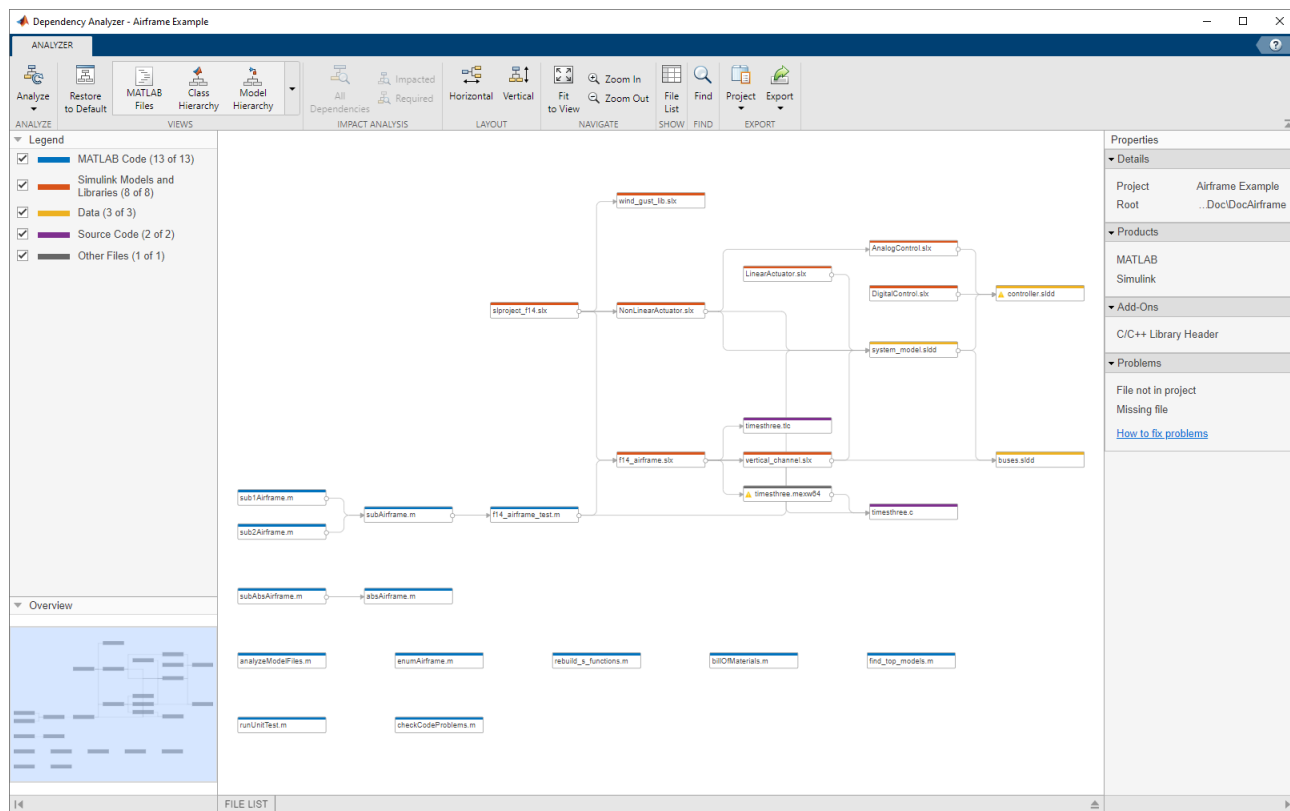
If a model was saved in a previous release with the `Propagate execution context across subsystem boundary` parameter set to **on**, the execution context parameter will be ignored. Blocks connected to a conditionally executed subsystem will not inherit execution context from that subsystem. This may change run-time performance and may produce run-time errors.

Project and File Management

Dependency Analyzer: Unified App for analyzing project dependencies and model hierarchies

Use a single app, Dependency Analyzer, to analyze project dependencies and model hierarchies.

- For a project, on the **Project** tab, in the **Tools** section, click **Dependency Analyzer**.
- For a model, on the **Modeling** tab, in the **Design** section, click **Dependency Analyzer**.



Run a dependency analysis to:

- Visualize the design structure and dependencies when you setup or explore a design for the first time. For more details, see [Explore the Dependency Graph, Views, and Filters](#).
- Find products and toolboxes required by your design. For more details, see [Find Required Products and Toolboxes](#).
- Investigate and resolve problems before sharing, packaging or submitting your project to source control. For more details, see [Check Dependency Results and Resolve Problems](#).
- Assess the impact of the changes you made on the rest of the design. For more details, see [Perform an Impact Analysis](#).
- Export the dependency analysis results as workspace variables, `.graphml` files, or send files for custom task processing. For more details, see [Save, Open, and Compare Dependency Graphs](#).

You can specify the analysis scope to examine dependencies inside user-defined toolboxes, introduced by code generation, or introduced by MATLAB code in model parameters. For more details about available options, see Analysis Scope.

For an example showing how to perform an impact analysis to find and run the tests affected by modified files, see Perform Impact Analysis with a Project.

Behavior Change

Workflow	R2019b	R2020a
Examine project dependencies and problem files using the file list.	At the top right of the Dependency Analysis view, select Table View . MATLAB switches to the file list view.	In the Dependency Analyzer, click File List . MATLAB opens the file list in the same view as the graph. See Investigate Problem Files in File List.
Investigate how two files are related and highlight where the dependency is introduced.	Expand the file in the graph by clicking the arrows next to the file name. To highlight a dependent block in the model, double-click the block name in the expanded file.	Select the dependency arrow. In the Properties pane, in the Details section, you can see the full paths of the files you are examining, the dependency type, and where the dependency is introduced. To highlight where the dependency is introduced, in the Details section, click the link under Impacted . See Investigate Dependency Between Two Files.
Color files in the dependency graph by type, source control status, or label.	On the Dependency Analysis tab, in the Group By menu, select the option you want.	In the Dependency Analyzer, in the Views section, use the different views to explore your project files dependencies. See Explore the Dependency Graph, Views, and Filters.

Project Checks: Run all project checks programmatically

You can now use runChecks to run all project checks programmatically.

Model Comparison: Create PDF model comparison reports

You can now save your model comparison results in a PDF report. For more details, see Save Printable Report.

Model Comparison: Color legend for comparison reports

The comparison report now uses a color legend to denote insertion, deletion and modification. To learn more about comparing models, see Compare and Merge Simulink Models.

Project API: Get latest Git revision programmatically

You can now programmatically get the latest Git revision for every file in your project.

For a project under Git source control, use `currentProject` to create a project object from the currently loaded project.

```
proj = currentProject;
```

Get the latest Git revision of the project file number *fileNumber*.

```
proj.Files(fileNumber).Revision
```

For an example on how to get the latest revisions for modified files, see [List Modified Files in Project \(MATLAB\)](#).

Simulink Model Files: Determine update release number used to save SLX models

Use `ReleaseUpdateLevel` property to determine the update release number that was used to save the SLX model. For more details, see [Simulink.MDLInfo](#).

Simulink Template: Specify template group in the template dialog box

When you create a project or a model template, you can now specify your template group in the template dialog box. This was only possible programmatically. For more details, see [Create a Template from a Model and Create a Template from the Current Project](#).

Case-sensitive model and library names

Starting R2020a, Simulink model and library names are case-sensitive. For models and projects saved in previous releases, Simulink provides an upgrade check to identify and fix case-insensitive references to models and libraries. This check is available in the model Upgrade Advisor and in the Upgrade Project tool. For more details, see [Check for case mismatches in references to models and libraries](#).

Version History

Function Element Name	What Happens When You Use This Function Element	Use This Instead	Compatibility Considerations
Case-insensitive model or library name	Simulink errors and suggests closest match.	Specify exact case.	Case-insensitive matching has been removed.

Data Management

Signal Editor user interface updates

Signal Editor user interface now allows you to:

- Graphically and tabularly edit existing signal data in the new **Edit** tab. For existing signal data, you can add or delete data, points, lines, and so forth. You can also recast data types by using the data type drop down list.
- Use the **Undo/Redo** buttons to back out or redo changes. These are contextual buttons that let you back out or redo changes in the signal hierarchy, signal plot, and tabular edit areas. These buttons are available on the **Signal Editor**, **Edit**, and **Draw** tabs.
- Cast signal data type to and from fixed-point or enumerated data types in the plot area using the data type drop down list. In previous releases, you were not able to recast signal data to fixed-point or enumerated data types.
- On the **Edit** and **Draw** tabs, more fully use multi-touch gestures or your mouse to zoom in and out and along the x-y axis.
- Import variables in custom formats. Use the accompanying API to register your own variable types with the Signal Editor user interface, then use the **Open > Import** command to import these custom variables.

For more information, see Create and Edit Signal Data.

Import and export custom format variables using `Simulink.io.FileType` objects

You can now import and export signal data in custom formats into the Signal Editor user interface using the `Simulink.io.FileType` class and its methods. For more information, see Create Custom File Type for Import to Signal Editor.

`createInputDataset` function allows signal format specification

The `createInputDataset` function allows specification of the signal formats for the signal elements in the dataset.

Specify value of `Simulink.LookupTable` argument by using simple numeric value

Previously, when you created a `Simulink.LookupTable` object in a model workspace and configured the object as a model argument, the instance-specific value of that object in a model block could only be specified as a new `Simulink.LookupTable` object in the parent model. The field values of the new object were required to match the values of the object in the referenced model. In R2020a, you can specify the instance-specific value of a `Simulink.LookupTable` argument as a simple MATLAB structure or array.

When you set **Specification** to `Explicit` value or `Even` spacing, the value can be:

- The name of a valid MATLAB structure variable, such as `Model1_LUT2`

- A literal structure expression, such as `struct('Table', ..., 'BP1', ..., 'BP2', ...)`
- Other expressions that return a valid structure, such as `Params.Model1.LUT2` or a call to a MATLAB function

When you set **Specification** to **Reference**, the value can be:

- A literal numeric array value, such as `[1 5 7; 2 8 13]`
- The name of a numeric array variable, such as `Model1_LUT2`
- Other expressions that return a valid numeric array, such as `Params.Model1.LUT2` or a call to a MATLAB function

When you specify the instance-specific value of a `Simulink.LookupTable` argument as a structure, the following rules apply:

- Each field of the model argument definition must be specified in the structure and the number of fields and the names of the fields must match.
- The dimensions of the table and the breakpoint data in the structure must match that of the model argument definition.
- If the data type of a structure field is `double`, the value is cast to the data type of the corresponding model argument field. Otherwise, the value must match the data type of the corresponding model argument field.

You can specify the value as a simple numeric value for any simulation mode and for code generation. For code generation, if you configure the model argument with a storage class of `Auto`, the structure or numeric array variable is not preserved in the generated code. If you set the storage class to any other value, the structure or numeric array is similar to other model arguments in that the value is used to initialize the tunable argument in the generated code.

For more information, see [Configure Instance-Specific Data for Lookup Tables](#).

64-bit integer data types

You can now specify `int64` and `uint64` data types for block outputs as well as other block parameters. You can also use a MATLAB variable with a 64-bit integer data type to specify parameter values. For example, you can use a variable `x = int64(7)`, defined in the base workspace, to specify the **Gain** parameter of a Gain block.

Stateflow charts also now support 64-bit integer data types.

When assigned into the output of a MATLAB Function block, 64-bit integer data types now behave the same way 32- 16- and 8-bit integers behave.

On signal lines, Simulink displays signed and unsigned 64-bit integer data types as `int64` and `uint64`. In previous releases, these data types displayed as `sfix64` or `ufix64`, respectively.

External Data node for model in Model Explorer

In Model Explorer, the **External Data** node under the model node provides access to the external data sources for that model, including:

- The base workspace. Access data in the base workspace from this node when you select the model property **Enable model access to base workspace** or the data dictionary property **Enable dictionary access to base workspace**.
- An attached data dictionary.

The following user interface enhancements support this change:


- In the Model Properties dialog box, the **Data** tab is renamed **External Data**.
- In the lower-left corner of the model canvas, when you click the badge that is used to access model data, the **Data Dictionary** menu option is renamed **External Data**. Select this menu item to navigate to the **External Data** node for the model in Model Explorer.

For more information on elements in the **Model Hierarchy** pane in Model Explorer, see **Model Explorer**.


Simulink.Bus object enhancements

- You can now create and modify elements of `Simulink.Bus` objects in the Model Explorer. Edit Bus objects in the Model Explorer when you have many Bus objects or the Bus objects are stored in multiple locations. The Model Explorer provides quick startup regardless of the number of Bus objects and allows you to easily switch between editing Bus objects in the base workspace and data dictionaries.
- When using the `Simulink.Bus.cellToObject` function, you can now use an array to define the elements of the Bus object.
- The `Simulink.Bus.createMATLABStruct` function can now cast values in a partial structure to the data types expected by the corresponding bus elements. For casting to happen, the conversion must be lossless. For example, suppose that the partial structure specifies 10.0 with data type `double`, but the corresponding bus element expects data type `uint8`. The resulting structure uses 10 with data type `uint8` for the bus element. If the partial structure specified a value of 10.5 instead of 10.0, you would receive an error because converting 10.5 with data type `double` to 10 with data type `uint8` results in a loss of precision.


Improved usability for specifying model arguments and data type block parameters

When you specify a value for a model argument or a data type block parameter, you can create a variable where you edit the value. In the Model Data Editor, the block dialog box, Property Inspector, or Model Explorer, click  near the value text box. Then select the option to create the variable from the menu. In the Create New Data dialog box, set the name and location for the new variable, then click **Create**. The text box now displays the new variable.

In addition, if you specify an undefined variable as a value, an edit-time check provides an error cue on the block. Clicking the error cue provides more information and allows you to create the missing variable by using a **Fix** button.

- Model arguments — When you specify an instance-specific value for a model argument in the Model block, click  to create an undefined variable or to create a variable from a simple numeric expression.

If you specify an undefined variable as an instance-specific value, an edit-time check provides an error cue on the Model block.

- Data type block parameters — When you specify a data type for a parameter, click  to create an undefined variable. The menu provides an option to create a variable that is relevant to the current value of the parameter.

In the case of data type block parameters, an edit-time check provides notification for undefined variables both on the block and on the block dialog box.

For more information on specifying a model argument, see [Parameterize Instances of a Reusable Referenced Model](#).

For more information block parameter data types, see [Control Block Parameter Data Types](#).

Name change for properties that allow access to base workspace

Previously, in the Model Properties dialog box and the Model Explorer, the property to enable model access to the base workspace and the property to enable data dictionary access to the base workspace both had the name, **Enable access to base workspace**. In R2020a, these options are renamed for clarity. In the Model Properties dialog box, on the **External Data** tab, the property for model access is **Enable model access to base workspace**. In the Model Explorer, in the dialog box pane for a data dictionary, the property for data dictionary access is **Enable dictionary access to base workspace**.

For more information on these properties, see [Continue to Use Shared Data in the Base Workspace](#).

View referenced dictionary hierarchy by using the Dependency Analyzer

Previously, in the **Dialog** pane of the Model Explorer, the **Referenced Dictionaries** section displayed the entire hierarchy tree for the selected data dictionary. In R2020a, this section displays each directly referenced data dictionary as a top level node and the indirectly referenced data dictionaries as a flat list below each top level node. To see the full dependency tree of referenced dictionaries in the Dependency Analyzer, click **View Hierarchy**.

For more information on using referenced dictionaries, see [Partition Dictionary Data Using Referenced Dictionaries](#).

Block Enhancements

C Function Block: Integrate and call C code from the model

Use the C Function block to call external C code from the model. You can:

- Write C code to preprocess data to call a C function. Similarly, you can write C code to postprocess data after calling a C function.
- Specify different code for simulation and code generation.
- Call multiple functions.
- Initialize and work with states cached in the block.

To use the C Function block, click **Library Browser** and find the C Function block under **Simulink > User Defined Functions**.

Delay and Unit Delay blocks support strings

These blocks now accept the string data type at inport ports:

- Delay — Can accept and output string data type if
 - The block is configured for the default value of the **Initial condition** parameter (0).
 - The **Delay length** parameter value is 1 or less.
- Unit Delay

Delay block now accepts bus input for initial condition

The Delay block now accepts nonvirtual bus data type for the initial condition port, x_0 .

Label enhancements for In Bus Element and Out Bus Element blocks

In the block label of In Bus Element and Out Bus Element blocks, you can now edit the port name. The label now consists of two interactive text fields: the port name and the bus element.

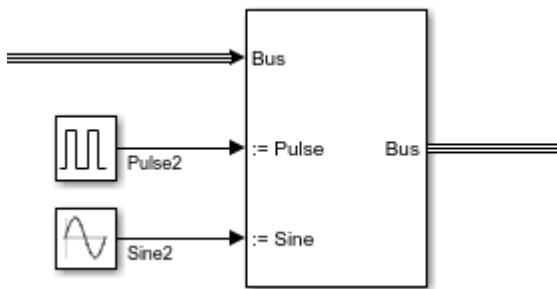


When you change the bus name for an In Bus Element or Out Bus Element block, all blocks that share the port update to reflect the new name.

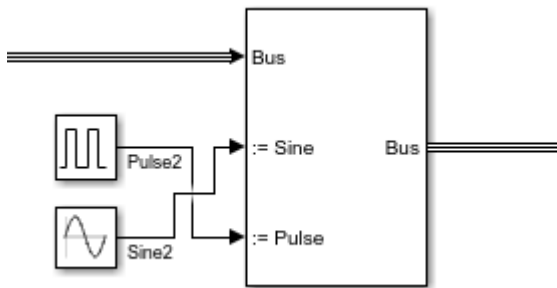
Bus Assignment block input signals stay connected to corresponding ports when reordered

In the Bus Assignment block dialog box, when you reorder the list of bus elements under the **Signals that are being assigned** parameter, the corresponding ports now stay connected to the same input signals.

For example, suppose signals `Pulse2` and `Sine2` are assigned to bus elements `Pulse` and `Sine` respectively.



When you reorder these bus elements in the dialog box, the corresponding ports change places, but signals `Pulse2` and `Sine2` remain assigned to bus elements `Pulse` and `Sine` respectively.



Programmatically refresh Model blocks

To refresh a Model block, you can now use the `Simulink.ModelReference.refresh` function. For more information on refreshing Model blocks, including how to interactively refresh a Model block, see [Refresh Model Blocks](#).

New string blocks for pattern matching

New string blocks include:

- String Contains block and its convenience blocks, String Starts With and String Ends With blocks, support pattern matching within a string, at the start of a string, and at the end of a string.
- String Count counts occurrences of pattern in string.

String Find and String Length blocks now support output data type

The String Find and String Length blocks now support the specification of output data types.

String blocks and rapid accelerator mode

Rapid accelerator mode simulation now supports Simulink strings. Simulation from the MATLAB command line is not supported for rapid accelerator mode.

Direct Lookup Table (n-D) block has new parameter

The Direct Lookup Table (n-D) block has a new parameter, `Remove protection against out-of-range input in generated code`, to remove code that checks for out-of-range input values.

Simulink toolstrip Lookup Table tab

The Simulink toolstrip **Block** tab changes to a **Lookup Table** tab when you select one of these lookup table blocks. The **Lookup Table** tab contains lookup table specific widgets, such as the **Lookup Table Editor** button.

- n-D Lookup Table
- 1-D Lookup Table
- 2-D Lookup Table
- Prelookup

Signal Editor block supports timetable objects

The Signal Editor block now supports `timetable` objects.

Step block now supports single data type

The Step now lets you select output data of type `single` using the **Output data type** parameter.

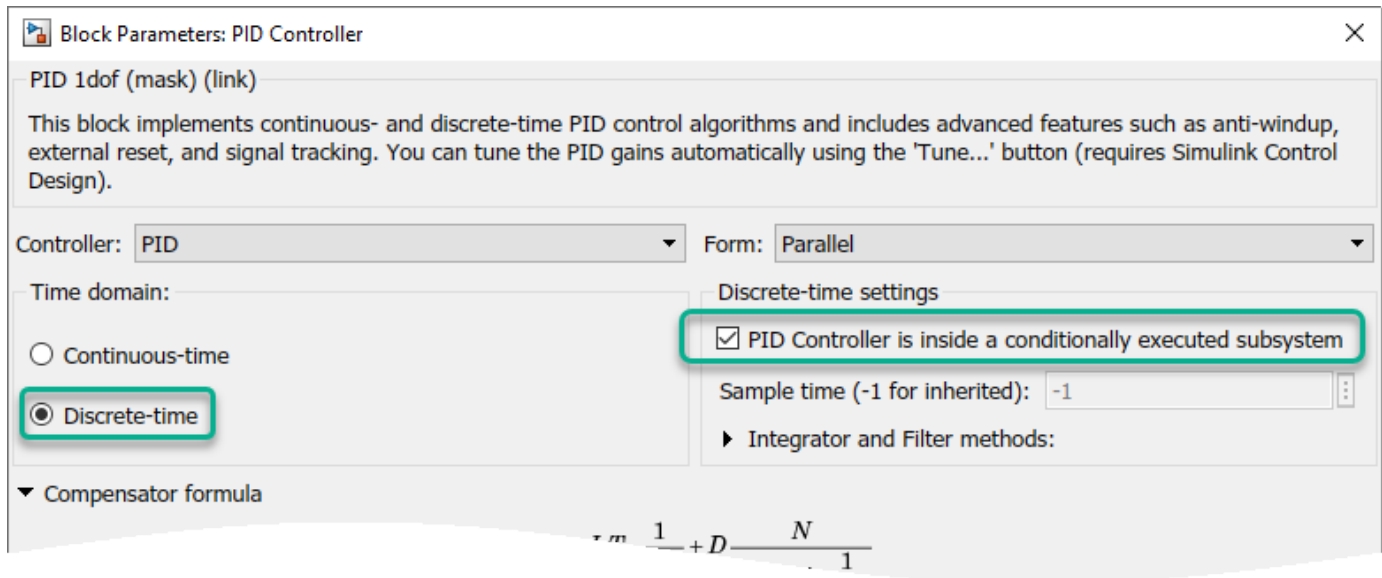
PID Controller Simulink blocks: Support for asynchronous interrupts with external targets

The Discrete PID Controller and Discrete PID Controller (2DOF) blocks now let you use your own value of discrete-time integrator sample time to control an external target. The value of the discrete-time integrator sample time should match the average sampling rate of the external interrupts, when the blocks are used inside a conditionally-executed subsystem.

Previously, the PID blocks inherited the sample time for discrete integration from model base rate, when used within a conditionally-executed subsystem. This behaviour led to the following scenarios:

- The discrete-integrator worked as intended if base rate, T_s , was the same as the asynchronous interrupt trigger rate from the external target.
- The discrete-integrator failed to integrate correctly when base rate, T_s , was different than the asynchronous interrupt trigger rate.

To ensure proper discrete integration when the PID controller blocks are used in a conditionally executed subsystem, check the option **PID Controller is inside a conditionally executed subsystem** in the block parameters menu to enable the T_{DTI} port. Use the T_{DTI} port to provide a scalar value of Δt for accurate discrete-time integration.



Search Simulink blocks using internationalized keywords

Starting in R2020a, you can search Simulink blocks using languages other than English.

Also, the Quick Insert menu is enhanced with more keywords associated to blocks to improve the results while searching for Simulink blocks.

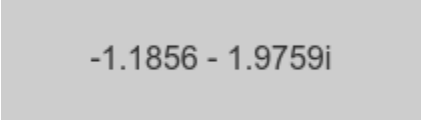


For more information on keywords in Quick Insert and to configure your own keywords for blocks, see [Configure Block with Keywords for Quick Insert](#).

Dashboard library Display block supports vector, 2-D matrix, and complex signals, and has additional format options

The Display block in the Dashboard library supports complex, vector, and 2-D matrix signals.

Complex signal values are displayed in the form of $a+bi$.



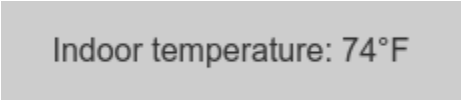
-1.1856 - 1.9759i

Vector and 2-D matrix signals display in a grid, with a cell for each signal element. You can control whether the grid is displayed and specify its color.

2	4	6	8
3	5	7	9

The Display block also has new **Integer** and **Custom** format options. When using the **Integer** format, the Display block rounds the input to a whole number.

The **Custom** format allows you to specify a custom **Format String** in which to display the data. You specify the **Format String** using the same format and options available for the `compose` function. For example, you can display the value for the simulated indoor temperature value using the **Format String** `Indoor temperature: %.0f°F`.



Indoor temperature: 74°F

New custom Horizontal Gauge and Vertical Gauge blocks in the Dashboard library

Two new custom blocks in the Dashboard library facilitate creating intuitive and photo-realistic dashboards for interacting with your models. The Horizontal Gauge block and Vertical Gauge block provide a similar out-of-the-box look and feel that you can use as-is in your model or customize with your own background image, needle image, and scale appearance.



For example, you could customize the Vertical Gauge block to look like a thermometer in a model that simulates temperature.



Root-level Inport blocks support interpolation for fixed-point data types and loading `matlab.io.datastore.sd datastore` data

In R2020a, root-level Inport blocks can interpolate fixed-point data you load into your model. You can also stream input data for a simulation from a `matlab.io.datastore.sd datastore` object using Inport blocks in the top level of your model.

From Workspace block supports loading timetable data

You can use the From Workspace to load a `timetable` that contains data for a single variable as external input for a simulation.

Rate Transition block deterministic mode support for concurrent execution

R2020a adds deterministic mode support to the Rate Transition block for models partitioned and configured for concurrent execution. You can configure a model for concurrency and configure Rate Transition blocks in that model so that the code generator produces code that transfers data predictably. To enable concurrent task execution, select model configuration parameter **Allow tasks to execute concurrently on target**. To enable deterministic mode for a Rate Transition block, select block parameter **Ensure deterministic data transfer (maximum delay)**. For example, specify these configuration settings if you are checking for a match between simulation and code generation numerical results for a model that uses concurrent task execution on multicore target hardware.

For more information, see Rate Transition block and Multicore Processor Targets.

For Each Subsystem block supports test-pointed signals

Starting in R2020a, you can use test-pointed signals with For Each Subsystem blocks.

Expanded DWork buffer length

Prior to R2020a, the total DWork buffer length in a model was $2^{31}-1$ bytes on 64-bit systems. In R2020a, there is no limit for the maximum number of bytes for the total DWork buffer length in a model. For more information, see Maximum Size Limits of Simulink Models.

Hit Crossing block outputs function-call events

Starting in R2020a, you can use a function-call event as the output type for the Hit Crossing block.

Functionality being removed or changed

The Fcn block in the Discrete library is not recommended.

Still runs

The Fcn block in Simulink/User-Defined Functions library is not recommended. Use the MATLAB Function block instead.

Connection to Hardware

Create display of gauges on Android devices

Starting in R2020a, the Gauge and the Half Gauge blocks from the Simulink Dashboard library are enhanced to generate code on Android devices. The code generation allows you to deploy these blocks as a standalone executable on the device. You can visualize your Android application using these blocks without any active connection between the device and the computer. For more information, see [Monitor Accelerometer Data Using Gauges on Android \(Simulink Support Package for Android Devices\)](#).

Improved build and deployment speeds of Simulink Support Package for Arduino Hardware

The Simulink Support Package for Arduino Hardware now builds and deploys Simulink models faster. To improve speeds, the support package now provides:

- **Parallel builds** — In previous releases, the generated code and the driver source codes were compiled sequentially. From this release, the codes are compiled in parallel, thus reducing the time taken to build the models. You can change this default behavior and choose to build the model sequentially by selecting the **Disable parallel build** parameter.
- **Static library** — In previous releases, the Arduino library files were built each time you compiled the Simulink model. From this release, the Arduino library files are saved in a static library file. This library file is created only when the Simulink model is compiled for the first time. For all consecutive compilations, the support package uses the same library file, resulting in a faster build. To change this default behavior, select the **Enable force build for static library** parameter.

The **Disable parallel build** and **Enable force build for static library** parameters are available in the **Target hardware resources** section of the Configuration Parameters dialog box.



Set PWM frequency on Arduino hardware

You can use the **Frequency** parameter available in the PWM block parameter dialog box to generate the desired pulse width modulation (PWM) frequency on the specified pin of the Arduino hardware. When you set the **Frequency** as:

- **Default** — The block generates a PWM signal with default frequency.
- **Specify** — The block generates a PWM signal with a frequency that is closest to the frequency specified in **Desired (Hz)**.

Capture data using Arduino sensor blocks

The Simulink Support Package for Arduino Hardware now supports the MPU9250 IMU Sensor and MPU6050 IMU Sensor blocks.

- MPU9250 IMU Sensor — Measures temperature, acceleration, angular rate, and magnetic field along the axes of the MPU9250 sensor connected to the Arduino hardware.
- MPU6050 IMU Sensor — Measures temperature, acceleration, and angular rate along the axes of the MPU6050 sensor connected to the Arduino hardware.

Support added for Raspberry Pi 4B model board

The Simulink Support Package for Raspberry Pi Hardware now supports deploying Simulink models on the Raspberry Pi 4B board. You can use all the blocks from the Simulink Support Package for Raspberry Pi Hardware with the Raspberry Pi 4B board.

Use Connected I/O to communicate with Raspberry Pi in Normal mode simulation

The Simulink Support Package for Raspberry Pi Hardware supports Connected I/O during Normal Mode (Simulink Support Package for Raspberry Pi Hardware) simulation. When you simulate a model in Connected I/O, the model communicates with the IO peripherals on the hardware, enabling you to verify the model design before deploying it on the hardware.

The Connected I/O is enabled on these Raspberry Pi blocks:

- GPIO Read and GPIO Write
- LED
- Standard Servo Write
- PWM
- I2C Master Read and I2C Master Write
- SPI Master Transfer, SPI Register Read, and SPI Register Write

Read data from web socket server using Raspberry Pi hardware

You can use the WebSocket Subscribe block to read the data from a web socket server using the Raspberry Pi hardware. The block outputs the received data along with the status indicating the validity of the data. The data must be in JavaScript Object Notation (JSON) format.

Pixy2 Vision Sensor Block

The Simulink Support Package for LEGO MINDSTORMS EV3 Hardware provides a new block - Pixy2 Vision Sensor. You can use this block in your Simulink models to deploy vision-based applications on the LEGO MINDSTORMS EV3 brick mounted with the CMUCam5 Pixy2 Vision Sensor. You can use

the PixyMon utility to train the vision sensor, and the block provides three color tracking modes and one line tracking mode.

Enhancements to Parrot Flight Control Interface

The Parrot® Flight Control Interface in Simulink Support Package for Parrot Minidrones is enhanced to include a new button, **Land**, which helps you to land the Parrot minidrone any time after you start the flight of the drone. When you click **Land**, the drone descends (along z-axis) from its current position to a height of 0.3 meters from the ground and then shuts down the motors.

MATLAB Function Blocks

Call scoped Simulink functions from MATLAB Function block

MATLAB Function block can call scoped Simulink functions. To call a scoped Simulink function inside a subsystem from a MATLAB Function block, use the following syntax:

```
subsystemName.simulinkFunctionName()
```

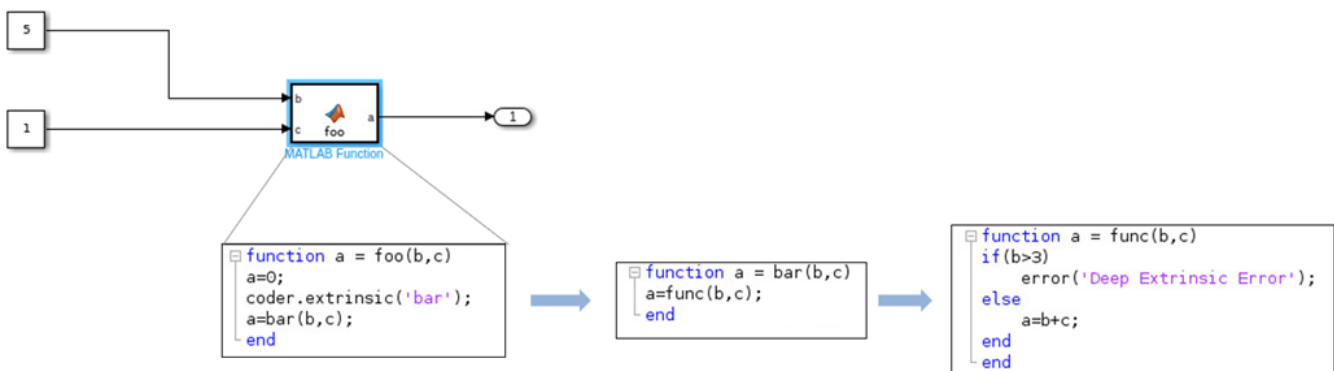
64-bit integer type support for MATLAB Function block

MATLAB Function blocks support 64-bit integer data. Charts implement `int64` and `uint64` data types as fixed-point numbers with a word length of 64 bits and a fraction length of 0.

Enhanced error reporting from MATLAB Function blocks

Starting in R2020a, MATLAB Function blocks provide more specific information about the error. The error message displays the full run-time error stack with hyperlinks to the block causing the error.

Here is an example of a model that contains a MATLAB Function block with an extrinsic function call that causes an error.



When you simulate this model, the reported error in the Diagnostic Viewer is:

```
Deep Extrinsic Error
Error in func.m (line 3)
    error('Deep Extrinsic Error');
Error in bar.m (line 2)
    a=func(b,c);
Error in 'DeepExtrinsicError/MATLAB Function' (line 4)
    a=bar(b,c);
Component: Simulink | Category: Model error
```

Code generation for datetime arrays

In R2020a, you can generate C/C++ code for `datetime` arrays. For more information, see Code Generation for Datetime Arrays.

Code generation for timetables

In R2020a, you can generate C/C++ code for timetables that have duration vectors as row times. For more information, see Code Generation for Timetables.

Code generation for more MATLAB functions

- `array2timetable`
- `convhull`
- `datetime`
- `fscanf`
- `ischange`
- `isnat`
- `isregular`
- `NaT`
- `onCleanup`
- `smoothdata`
- `table2timetable`
- `timetable`
- `timetable2table`
- `topkrows`

Code generation for more toolbox functions

5G Toolbox

- `nrCORESETConfig`
- `nrPDCCHConfig`
- `nrPDCCHResources`
- `nrPDCCHSpace`
- `nrPDSCHConfig`
- `nrPDSCHDMRS`
- `nrPDSCHDMRSConfig`
- `nrPDSCHDMRSIndices`
- `nrPDSCHIndices`
- `nrPDSCHPTRS`
- `nrPDSCHPTRSConfig`
- `nrPDSCHPTRSIndices`
- `nrPDSCHReservedConfig`
- `nrPRACH`
- `nrPRACHConfig`
- `nrPRACHGrid`

- nrPRACHIndices
- nrPUSCHConfig
- nrPUSCHPTRSIndices
- nrPUSCHPTRSConfig
- nrPUSCHPTRS
- nrPUSCHIndices
- nrPUSCHDMRSIndices
- nrPUSCHDMRSConfig
- nrPUSCHDMRS
- nrSearchSpaceConfig
- nrSRS
- nrSRSConfig
- nrSRSIndices

Audio Toolbox

- acousticLoudness
- acousticSharpness
- calibrateMicrophone
- detectSpeech
- phon2sone
- sone2phon

Automated Driving Toolbox

- cubicLaneBoundary
- findCubicLaneBoundaries
- findParabolicLaneBoundaries
- insertLaneBoundary
- parabolicLaneBoundary

Communications Toolbox

- bluetoothIdealReceiver
- bluetoothPhyConfig
- bluetoothWaveformConfig
- bluetoothWaveformGenerator
- channelDelay
- cranerainpl

Computer Vision Toolbox

See “Code Generation: Generate C/C++ code using MATLAB Coder” (Computer Vision Toolbox).

DSP System Toolbox

- `dsp.SOSFilter`

Fixed-Point Designer

- `nextpow2`
- `normalizedReciprocal`

Image Processing Toolbox

- `imbilatfilt`

Instrument Control Toolbox

- `maskWrite`
- `modbus`
- read from a MODBUS server
- write to a MODBUS server
- `writeread`

Model Predictive Control Toolbox

- `mpcActiveSetOptions`
- `mpcActiveSetSolver`
- `mpcInteriorPointOptions`
- `mpcInteriorPointSolver`
- `mpcmoveCodeGeneration`
- `nlmpcmoveCodeGeneration`

Navigation Toolbox

- copy of `plannerRRT` and `plannerRRTStar`
- `getParticles`
- `monteCarloLocalization`
- plan of `plannerHybridAStar`
- plan of `plannerRRT` and `plannerRRTStar`
- `plannerHybridAStar`
- `plannerRRT`
- `plannerRRTStar`
- `validatorVehicleCostmap`

Optimization Toolbox

See “Code Generation for Quadratic Problems: Generate C code for problems with linear constraints and quadratic objectives” (Optimization Toolbox).

Phased Array System Toolbox

- blkdiagbfweights
- cranerainpl

Robotics System Toolbox

- checkCollision
- collisionBox
- collisionCylinder
- collisionMesh
- collisionSphere

Sensor Fusion and Tracking Toolbox

- angvel
- deleteBranch, initializeBranch of trackerTOMHT
- deleteTrack, initializeTrack of trackerPHD
- residual, residualmag, residualaltimeter of ahrs10filter
- residual, residualaccel, residualgps, residualmag, residualgyro of insfilterAsync
- residual, residualgps, residualmvo of insfilterErrorState
- trackGOSPAMetric

SerDes Toolbox

- impulse2pulse
- optPulseMetric
- prbs
- pulse2impulse
- pulse2wave
- wave2pulse

Signal Processing Toolbox

See “C/C++ Code Generation Support: Generate code for time-frequency analysis, feature extraction, spectral analysis, multirate signal processing, and filter design” (Signal Processing Toolbox).

Wavelet Toolbox

- dualtree
- dualtree2
- dwpt
- haart
- haart2
- hht
- idualtree

- `idualtree2`
- `idwpt`
- `ihaart`
- `ihaart2`
- `modwtvar`
- `qbiorthfilt`
- `qorthwavf`
- `wdenoise`

WLAN Toolbox

- `getTRSConfiguration`
- `wlanHETBConfig`

Functionality being removed or changed

Behavior in MATLAB Function blocks with 64-bit fixed-point type inputs could change

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

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

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

To preserve the behavior of previous releases in your MATLAB Function block, use the function `cast64BitIntToFi`.

This behavior also applies to `ufix64` data types.

Modeling Guidelines

Support for MathWorks Advisory Board (MAB) v5.0 modeling guidelines

Version 5.0 of the MathWorks Advisor Board (MAB) modeling guidelines represents a new global standard for the usage of MATLAB, Simulink, Stateflow, and Embedded Coder in your Model-Based Design workflow. These guidelines are derived from JMAAB v5.1 and MAAB v3.0 modeling guidelines.

The MAB modeling guidelines are developed by an independent industry working group composed of major OEMs and suppliers across multiple industries. The guidelines include contributions from both Japan MATLAB Automotive Advisory Board (JMAAB) and North American MathWorks Automotive Advisory Board (NA-MAAB) subgroups. JMAAB is a subgroup that includes automotive manufacturers and suppliers in Japan. NA-MAAB is a subgroup of manufacturers and suppliers from multiple industries in the US and Europe.

To reflect the contributions and broad usage across multiple industries, the title of the guidelines changed from MathWorks Automotive Advisory Board (MAAB) modelling guidelines to MathWorks Advisor Board (MAB) modeling guidelines.

The MAB guidelines are an important basis for project success and teamwork. Observing the guidelines can help you:

- Avoid integration problems with well-defined interfaces.
- Improve readability and understandability of models with uniform appearance.
- Increase model reuse.
- Avoid problems exchanging models.

You can also view the guidelines in Microsoft Word .docx or .pdf format on the MathWorks website , see Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow - Version 5.0.

New MAB v5.0 guidelines are:

- jc_0008: Definition of signal names
- jc_0009: Signal name propagation
- jc_0110: Direction of block
- jc_0222: Usable characters for signal and bus names
- jc_0232: Usable characters for parameter names
- jc_0241: Length restriction for model file names
- jc_0242: Length restriction for folder names
- jc_0247: Length restriction for block names
- jc_0244: Length restriction for Inport and Outport names
- jc_0245: Length restriction for signal and bus names
- jc_0246: Length restriction for parameter name
- jc_0247: Length restriction for block names
- jc_0602: Consistency in model element names

- jc_0603: Model description
- jc_0604: Using block shadow
- jc_0610: Operator order for multiplication and division block
- jc_0611: Input sign for multiplication and division blocks
- jc_0621: Usage of Logical Operator blocks
- jc_0622: Usage of Fcn blocks
- jc_0623: Usage of continuous-time Delay blocks and discrete-time Delay blocks
- jc_0624: Usage of Tapped Delay blocks/Delay blocks
- jc_0626: Usage of Lookup Table blocks
- jc_0627: Usage of Discrete-Time Integrator blocks
- jc_0628: Usage of Saturation blocks
- jc_0630: Usage of Multipoint Switch blocks
- jc_0640: Initial value settings for Output blocks in conditional subsystems
- jc_0641: Sample time setting
- jc_0642: Integer rounding mode setting
- jc_0643: Fixed-point setting
- jc_0644: Type setting
- jc_0645: Parameter definition for calibration
- jc_0650: Block input/output data type with switching function
- jc_0651: Implementing a type conversion
- jc_0653: Delay block layout in feedback loops
- jc_0655: Prohibition of logical value comparison in Stateflow
- jc_0656: Usage of Conditional Control blocks
- jc_0657: Retention of output value based on conditional control flow blocks and Merge blocks
- jc_0659: Usage restrictions of signal lines input to Merge blocks
- jc_0700: Unused data in Stateflow block
- jc_0701: Usable number for first index
- jc_0702: Use of named Stateflow parameters and constants
- jc_0711: Division in Stateflow
- jc_0712: Execution timing for default transition path
- jc_0721: Usage of parallel states
- jc_0722: Local data definition in parallel states
- jc_0723: Prohibited direct transition from external state to child state
- jc_0730: Unique state name in Stateflow blocks
- jc_0731: State name format
- jc_0732: Distinction between state names, data names, and event names
- jc_0733: Order of state action types
- jc_0733: Order of state action types
- jc_0736: Uniform indentations in Stateflow blocks

- jc_0738: Usage of Stateflow comments
- jc_0739: Describing texts inside states
- jc_0740: Limitation on use of exit state action
- jc_0741: Timing to update data used in state chart transition conditions
- jc_0751: Backtracking prevention in state transition
- jc_0752: Condition action in transition label
- jc_0753: Condition actions and transition actions in Stateflow
- jc_0760: Starting point of internal transition
- jc_0762: Prohibition of state action and flow chart combination
- jc_0763: Usage of multiple internal transitions
- jc_0770: Position of transition label
- jc_0771: Comment position in transition labels
- jc_0772: Execution order and transition conditions of transition lines
- jc_0773: Unconditional transition of a flow chart
- jc_0774: Comments for through transition
- jc_0775: Terminating junctions in flow charts
- jc_0790: Action language of Chart block
- jc_0791: Duplicate data name definitions
- jc_0792: Unused Data
- jc_0794: Division in Simulink
- jc_0795: Usable characters for Stateflow data names
- jc_0796: Length restriction for Stateflow data names
- jc_0797: Unconnected transitions / states / connective junctions
- jc_0800: Comparing floating-point types in Simulink
- jc_0801: Prohibited use of the /* and */ comment symbols
- jc_0802: Prohibited use of implicit type casting in Stateflow
- jc_0803: Passing values to library functions
- jc_0804: Prohibited use of recursive calls with graphical functions
- jc_0805: Numerical operation block inputs
- jc_0806: Detecting incorrect calculation results

Version History

This table maps the previously existing MAAB guidelines to the new MAB v5.0 guidelines. Please note, while only some guideline titles have changed in R2020a, all of the guideline rules are updated.

MAAB Guideline (R2019a and earlier)	MAB Guidelines (R2020a)
ar_0001: Filenames	ar_0001: Usable characters for file names
ar_0002: Directory	ar_0002: Usable characters for folder names
jc_0201: Usage characters for Subsystem names	jc_0201: Usable characters for subsystem names

MAAB Guideline (R2019a and earlier)	MAB Guidelines (R2020a)
jc_0231: Usage characters for block names	jc_0231: Usable characters for block names
jc_0211: Usable characters for Inport blocks and Outport blocks	jc_0211: Usable characters for Inport blocks and Outport block
na_0019: Restricted Variable Names	na_0019: Restricted variable names
jc_0011: Optimization parameters for Boolean data types	jc_0011: Optimization parameters for Boolean data types
jc_0021: Model diagnostic settings	jc_0021: Model diagnostic settings
na_0004: Simulink model appearance	na_0004: Simulink model appearance settings
db_0043: Simulink font and font size	db_0043: Model font and font size
jm_0002: Block resizing	jm_0002: Block resizing
db_0142: Position of block namesna_0006	db_0142: Position of block names
jc_0061: Display of block names	jc_0061: Display of block names
db_0140: Display of basic block parameters	db_0140: Display of block parameters
db_0081: Unconnected signals, block inputs, and block outputs	db_0081: Unconnected signals and blocks
db_0032: Simulink signal appearance	db_0032: Signal line connections
db_0141: Signal flow in Simulink models	db_0141: Signal flow in Simulink models
jc_0171: Maintaining signal flow when using Goto and From blocks	jc_0171: Clarification of connections between structural subsystems
jc_0281: Naming of Trigger port block and Enable Port block	jc_0281: Trigger signal names
db_0143: Similar block types on the model level	db_0143: Usable block types in model hierarchy
db_0144: Use of Subsystems	db_0144: Use of subsystems
hd_0001: Prohibited Simulink sinks	hd_0001: Prohibited Simulink sinks
na_0010: Grouping data flows into signals	na_0010: Usage of vector and bus signals
db_0097: Position of labels and signals for busses	db_0097: Position of labels for signals and buses
na_0008: Display of labels on signals	na_0008: Display of labels on signals
na_0009: Entry versus propagation of signal labels	na_0009: Entry versus propagation of signal labels
db_0110: Tunable parameters in basic blocks	db_0110: Block parameters
db_0112: Indexing	db_0112: Usage of index
db_0146: Triggered, enabled, conditional Subsystems	db_0146: Block layout in conditional subsystems
na_0003: Simple logical expressions in If Condition block	na_0003: Usage of If blocks
na_0002: Appropriate implementation of fundamental logical and numerical operations	na_0002: Appropriate usage of basic logical and numerical operations
jc_0121: Use of the Sum block	jc_0121: Usage of add and subtraction blocks
jc_0131: Use of Relational Operator block	jc_0131: Usage of Relational Operator blocks

MAAB Guideline (R2019a and earlier)	MAB Guidelines (R2020a)
db_0042: Port block in Simulink models	db_0042: Usage of Inport and Outport blocks
jc_0081: Icon display for Port block	jc_0081: Inport and Outport block icon display
na_0011: Scope of Goto and From blocks	na_0011: Scope of Goto and From blocks
jc_0161: Use of Data Store Read/Write/Memory blocks	jc_0161: Definition of Data Store Memory blocks
jc_0141: Use of the Switch block	jc_0141: Usage of the Switch blocks
na_0020: Number of inputs to variant subsystems	na_0020: Number of inputs to variant subsystems
na_0036: Default variant	na_0036: Default variant
na_0037: Use of single variable variant conditionals	na_0037: Use of single variable for variant condition
db_0122: Stateflow and Simulink interface signals and parameters	db_0122: Stateflow and Simulink interface signals and parameters
db_0123: Stateflow port names	db_0123: Stateflow port names
db_0125: Scope of internal signals and local auxiliary variables	db_0125: Stateflow local data
db_0126: Scope of events	db_0126: Defining Stateflow events
db_0137: States in state machines	db_0137: States in state machines
db_0129: Stateflow transition appearance	db_0129: Stateflow transition appearance
jc_0531: Placement of the default transition	jc_0531: Default transition
db_0132: Transitions in flow charts	db_0132: Transitions in flow charts
jm_0011: Pointers in Stateflow	jm_0011: Pointers in Stateflow
jc_0491: Reuse of variables within a single Stateflow scope	jc_0491: Reuse of Stateflow data
jm_0012: Event broadcasts	jm_0012: Usage restrictions of events and broadcasting events
db_0127: MATLAB commands in Stateflow	db_0127: Limitation on MATLAB commands in Stateflow blocks
jb_0481: Use of hard equality comparisons for floating point numbers in Stateflow	jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow
na_0001: Bitwise Stateflow operators	na_0001: Standard usage of Stateflow operators
jc_0451: Use of unary minus on unsigned integers in Stateflow	jc_0451: Use of unary minus on unsigned integers
jc_0501: Format of entries in a State block	jc_0501: Format of entries in a State block
jc_0521: Use of the return values from graphical functions	jc_0511: Return values from a graphical function
na_0042: Location of Simulink functions	na_0042: Usage of Simulink functions
na_0039: Use of Simulink in Stateflow charts	na_0039: Limitation on Simulink functions in Chart blocks
na_0018: Number of nested if/else and case statement	na_0018: Number of nested if/else and case statement

MAAB Guideline (R2019a and earlier)	MAB Guidelines (R2020a)
na_0025: MATLAB Function Header	na_0025: MATLAB Function header
na_0024: Global Variables	na_0024: Shared data in MATLAB functions
na_0031: Definition of default enumerated value	na_0031: Definition of default enumerated value
na_0034: MATLAB Function block input/output settings	na_0034: MATLAB Function block input/output settings
na_0016: Source lines of MATLAB Functions	na_0016: Source lines of MATLAB Functions
na_0017: Number of called function levels	na_0017: Number of called function levels
na_0021: Strings	na_0021: Strings in MATLAB functions
na_0022: Recommended patterns for Switch/Case statements	na_0022: Recommended patterns for Switch/Case statements

This table lists the guidelines that have been removed in R2020a. Where applicable, it specifies which guideline(s) can be used as a replacement or the reference material in which the guideline content is covered.

Guideline	In 2020a, see
db_0040: Model hierarchy	Available as reference documentation. See Understanding Model Architecture
db_0114: Simulink patterns for If-then-else-if constructs	Available as reference documentation. See Simulink Patterns for If, elseif, else Constructs
db_0115: Simulink patterns for case constructs	Available as reference documentation. See Simulink Patterns for case Constructs
db_0116: Simulink patterns for logical constructs with logical blocks	Available as reference documentation. See Simulink Patterns for Logical Constructs
db_0117: Simulink patterns for vector signals	Available as reference documentation. See Simulink Patterns for Vector Signals
na_0038: Levels in Stateflow charts	Available as reference documentation. See jc_0321 Limiting State Layering
db_0133: Use of patterns for flow charts	Available as reference documentation. See Usage Patterns
db_0134: Flow chart patterns for If constructs	Available as reference documentation. See Flow Chart Patterns for If, Elseif, Else Constructs
db_0135: Flow chart patterns for loop constructs	Available as reference documentation. See Flow Chart Patterns for Loop Constructs
db_0148: Flow chart patterns for conditions	Available as reference documentation. See Flow Chart Patterns for Conditions
db_0149: Flow chart patterns for condition actions	Available as reference documentation. See Flow Chart Patterns for Condition Actions
db_0150: State machine patterns for conditions	Available as reference documentation. See State Machine Patterns for Conditions

Guideline	In 2020a, see
db_0151: State machine patterns for transition actions	Available as reference documentation. See State Machine Patterns for Transition Actions
db_0159: Flow chart patterns for case constructs	Available as reference documentation. See Flow Chart Patterns for Case Constructs
jc_0111: Direction of Subsystem	Covered by guideline jc_0110
jc_0221: Usable characters for signal line names	Covered by guideline jc_0222
jc_0301 Controller model	Available as reference documentation. See Hierarchical Structure of a Controller Model
jc_0311: Top layer/root level	Available as reference documentation. See Top Layer
jc_0321: Trigger layer	Available as reference documentation. See Relationship between Simulink Models and Embedded Implementation
jc_0331: Structure layer	Available as reference documentation. See Subsystems
jc_0341: Data flow layer	Available as reference documentation. See Data Flow Layers
jc_0351: Methods of initialization	Available as reference documentation. See Initialization
jc_0541: Use of tunable parameters in Stateflow	Available as reference documentation. See Simulink Patterns for Vector Signals
jm_0001: Prohibited Simulink standard blocks inside controllers	Covered by guideline db_0143
jm_0010: Port block names in Simulink models	Covered by guidelines jc_0602, jc_0061, and jc_0081
na_0005: Port block name visibility in Simulink models	Covered by guideline jc_0061
na_0006: Guidelines for mixed use of Simulink and Stateflow	There is not a replacement guideline.
na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines	There is not a replacement guideline.
na_0012: Use of Switch vs. If-Then-Else Action Subsystem	Available as reference documentation. See Using Switch and If, Elseif, Else Action Subsystems
na_0013: Comparison operation in Stateflow	Covered by guideline jc_0802
na_0014: Use of local language in Simulink and Stateflow	Covered by guidelines jc_0201, jc_0211, jc_0222, jc_0232, jc_0231
na_0026: Consistent software environment	Available as reference documentation. See Considerations for Determining Guideline Operation Rules
na_0027: Use of only standard library blocks	There is not a replacement guideline.

Guideline	In 2020a, see
na_0028: Use of If-Then-Else Action Subsystem to Replace Multiple Switches	Available as reference documentation. See Use of If, Elseif, Else Action Subsystem to Replace Multiple Switches
na_0030: Usable characters for Simulink Bus names	Covered by guideline jc_0222
na_0032: Use of Merge Blocks	Covered by guidelines jc_0659 and jc-0657
na_0033: Enumerated Types Usage	Available as reference documentation. See Enumerated Types
na_0035: Adoption of naming conventions	Covered by guidelines jc_0201, jc_0211, jc_0222, jc_0232, jc_0231
na_0040: Number of states per container	Available as reference documentation. See Number of States per Stateflow Container
na_0041: Selection of function type	Available as reference documentation. See Function Types Available in Stateflow

High-Integrity System Modeling guidelines to verify compliance with DO, IEC, EN, and ISO safety standards

This table identifies new and modified modeling guidelines. For a complete list of High-Integrity System Modeling guidelines, including their applicable Model Advisor checks, see Model Advisor Checks for High-Integrity Modeling Guidelines.

High-Integrity Modeling Guideline	Rationale
hisl_0310: Configuration Parameters > Diagnostics > Model Referencing	Removed configuration parameter Model block version mismatch from the guideline. The parameter has been removed from Model Advisor check Check safety-related diagnostic settings for model referencing (Simulink Check).

R2019b

Version: 10.0

New Features

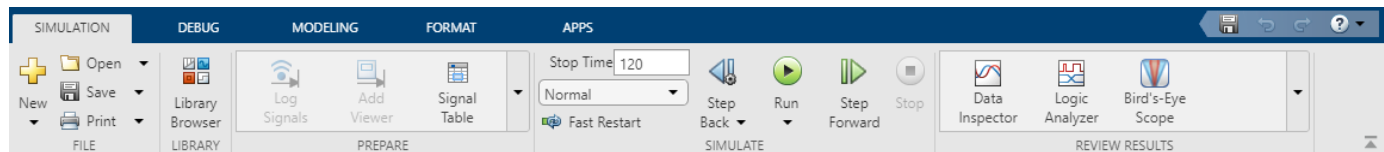
Bug Fixes

Version History

Simulink Editor

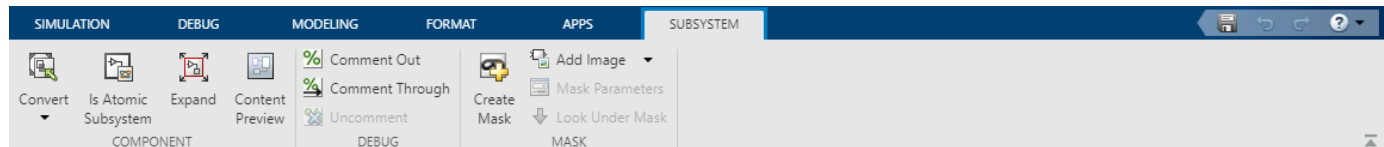
Simulink Toolstrip: Access and discover Simulink capabilities when you need them

The Simulink Toolstrip contains components that were previously available in menus and toolbars. To support common user workflow tasks, new tabs called **Simulation**, **Debug**, **Modeling**, **Format**, and **Apps** provide functionality corresponding to each task.



The **Apps** tab provides a gallery of applications from the Simulink family of products. Apps may open a new contextual tab, a separate window, or they may be a shortcut to the configuration parameters.

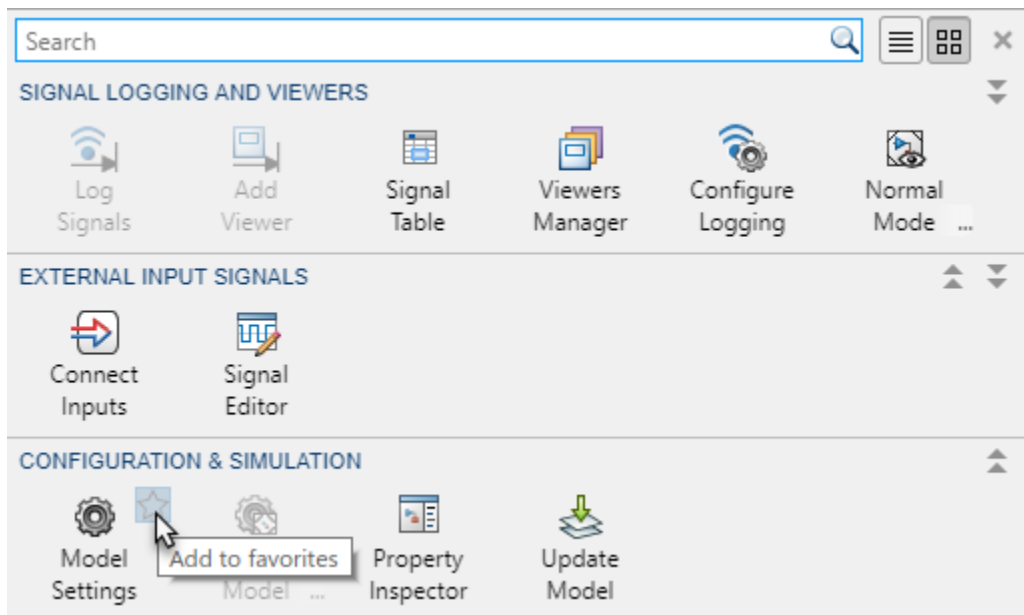
When you select a model component, a contextual tab activates, displaying the tools to assist you in a step in your workflow. For example, when you select a subsystem block, the **Subsystem** tab appears. If you select a Stateflow chart, the **State Chart** tab appears. When the block or chart is no longer selected, the contextual tabs disappear.



The quick access toolbar contains frequently used options such as save, undo, and redo.



After opening a gallery, you can mark buttons as favorites. Buttons that are favorites appear at the top of the gallery for quicker access.



You can customize your toolstrip options by adding commands and submenus to a menu bar and context menus for the Simulink® Editor and Stateflow® Editor. See [Add Items to Model Editor Menus](#).

Simulink Editor Changes

- “Mapping from Simulink Editor to the Simulink Toolstrip” on page 6-3
- “File Menu” on page 6-4
- “Edit Menu” on page 6-5
- “View Menu” on page 6-6
- “Display Menu” on page 6-8
- “Diagram Menu” on page 6-11
- “Simulation Menu” on page 6-14
- “Analysis Menu” on page 6-16
- “Code Menu” on page 6-20
- “Tools Menu” on page 6-23

Mapping from Simulink Editor to the Simulink Toolstrip


The following tables list the new Simulink Toolstrip items that are different from the Simulink Editor menu bar items. Many of the features and options that were previously hidden within the toolbar menus are now directly available from tabs on the Simulink Toolstrip.




File Menu

Menu Bar Item	Toolstrip Equivalent
New > <ul style="list-style-type: none"> • Blank Model (Ctrl+N) • Model • Chart • Library • Project • Architecture 	Simulation > New > <ul style="list-style-type: none"> • Blank Model (Ctrl+N) • Model • Chart • Library • Project • Architecture
Open (Ctrl+O)	Simulation > Open (Ctrl+O)
Open Recent	Simulation > Open > Recent Files.
Save (Ctrl+S)	Simulation > Save (Ctrl+S)
Save as	Simulation > Save > Save as
Project > Create Project from Model	Simulation > New > Project > New Project from this Model
Project > <ul style="list-style-type: none"> • Find Dependencies • Add to Project • Remove from Project • Get File Lock • Discard Local Changes and Release Locks • Compare to Ancestor • Compare to Revision 	Open your project. Simulation > Project > <ul style="list-style-type: none"> • Run Dependency Analysis • MATLAB desktop > Add Files • Remove Model From Project • Get File Lock • Discard Local Changes and Release Locks • Compare to Ancestor • Compare to Revision
Export Model to > <ul style="list-style-type: none"> • Web • Previous Version • Protected Model • Template • Architecture Model 	Simulation > Save > <ul style="list-style-type: none"> • Web View • Previous Version • Protected Model • Template • Architecture Model
Reports > System Design Description	Modeling > Design Gallery > System Design Report
Reports > System Requirements	Apps > Requirements Manager > Share > Generate Model Traceability Report
Model Properties > Model Properties	Modeling > Model Settings > Model Properties



Menu Bar Item	Toolstrip Equivalent
Model Properties > Link to Data Dictionary	Modeling > Design Gallery > Link to Data Dictionary
Model Properties > <ul style="list-style-type: none"> • Chart Properties • State Machine Properties 	Open your Stateflow chart. Modeling > Chart Properties > <ul style="list-style-type: none"> • Chart Properties • Machine Properties
Link to Data Dictionary	Modeling > Model Settings > Model Properties > Data tab
Print > <ul style="list-style-type: none"> • Print • Print to File • Print Details • Page Setup • Enable Tiled Printing • Show Page Boundaries 	Simulation > Print > <ul style="list-style-type: none"> • Print • Click Print. Select the Print To File check box. • Print Details • Page Setup • Enable Tiled Printing • Show Page Boundaries
Simulink Preferences	Modeling > Environment > Simulink Preferences
Stateflow Preferences > <ul style="list-style-type: none"> • Style • Syntax Highlighting 	Open your Stateflow chart. Format > Style > <ul style="list-style-type: none"> • Style • Syntax Highlighting
Exit MATLAB (Ctrl+Q)	Exit MATLAB from MATLAB

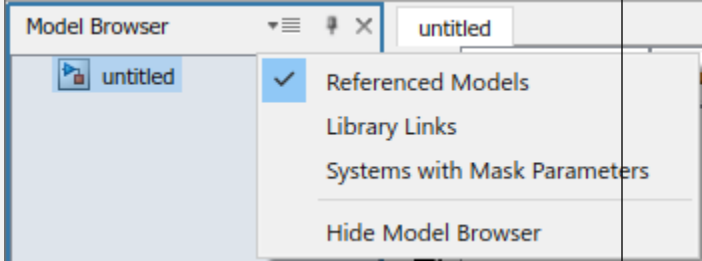

Edit Menu

Menu Bar Item	Toolstrip Equivalent
Undo (Ctrl+Z)	(Ctrl+Z)
Redo (Ctrl+Y)	(Ctrl+Y)
Cut (Ctrl+X)	(Ctrl+X)
Copy (Ctrl+C)	(Ctrl+C)
Copy Current View to Clipboard > <ul style="list-style-type: none"> • Metafile • Bitmap 	Format > Screenshot  > <ul style="list-style-type: none"> • Use Windows Metafile Format (.wmf) • Use Bitmap Format (.bmp)
Paste (Ctrl+V)	(Ctrl+V)
Paste Duplicate Inport	Right-click. Paste Duplicate Inport
Select All (Ctrl+A)	(Ctrl+A)

Menu Bar Item	Toolstrip Equivalent
Comment Through (Ctrl+Shift+Y)	Debug > Comment Through  (Ctrl+Shift+Y)
Comment Out (Ctrl+Shift+X)	Debug > Comment Out  (Ctrl+Shift+X)
Delete	Use the Delete key.
Find	Modeling > Find 
Find Referenced Variables	Modeling > Find > Find Ref Variables
Find & Replace in Chart	Open your Stateflow chart. Modeling > Find > Find & Replace in Chart
Bus Editor	Modeling > Design Gallery > Bus Editor
Lookup Table Editor	Modeling > Design Gallery > Lookup Table Editor












View Menu












Menu Bar Item	Toolstrip Equivalent
Library Browser	Simulation > Library Browser 
Model Explorer > Model Explorer	Modeling > Design Gallery > Model Explorer
Model Explorer > <ul style="list-style-type: none"> • Base Workspace • Data Dictionary • Model Workspace 	Modeling > Design Gallery > <ul style="list-style-type: none"> • Base Workspace • Data Dictionary • Model Workspace
Variant Manager	Modeling > Design Gallery > Variant Manager
Project	Open your project. Simulation > Project > View Project
Model Dependency Viewer > <ul style="list-style-type: none"> • Models & Libraries • Models Only • Referenced Model Instances 	Modeling > Design > Model Dependencies > <ul style="list-style-type: none"> • Models & Libraries • Models Only • Referenced Model Instances
Diagnostic Viewer	Debug > Diagnostics > Diagnostic Viewer 
Schedule Editor	Modeling > Design Gallery > Schedule Editor



















Menu Bar Item	Toolstrip Equivalent
Requirements at This Level <ul style="list-style-type: none"> • Link to Selection in MATLAB • Link to Current Test Case • Link to Selection in Word • Link to Selection in Excel • Select for Linking with Simulink • Add Link to Selected Object(s) • Open Outgoing Links dialog • Copy URL to Clipboard 	Right-click the canvas. Requirements at This Level > <ul style="list-style-type: none"> • Link to Selection in MATLAB • Link to Current Test Case • Link to Selection in Word • Link to Selection in Excel • Select for Linking with Simulink • Add Link to Selected Object(s) • Open Outgoing Links dialog • Copy URL to Clipboard
Model Browser > Show Model Browser	Modeling > Environment > Model Browser
Model Browser > <ul style="list-style-type: none"> • Include Referenced Models • Include Library Links • Include Systems with Mask Parameters 	Modeling > Environment > Model Browser > 
Property Inspector	Modeling > Design Gallery > Property Inspector (Ctrl+Shift+I)
Notes	Modeling > Design Gallery > Notes (Ctrl+Shift+N)
Model Data Editor	Modeling > Design Gallery > Model Data Editor (Ctrl+Shift+E)
Viewmarks > Viewmark This View	Simulation > Save > Save Viewmark
Viewmarks > Show Viewmarks	Simulation > Open > Open Viewmarks
Configure Toolbars	Modeling > Environment > Simulink Preferences
Toolbars	Modeling > Environment > Toolstrip
Status Bar	Modeling > Environment > Status Bar
Explorer Bar	Modeling > Environment > Explorer Bar
Navigate > <ul style="list-style-type: none"> • Back • Forward • Up to Parent 	Use the navigation arrows  in the top left of the Simulink Editor.
Navigate > Previous Tab	(Ctrl+Shift+Backtab)
Navigate > Next Tab	(Ctrl+Tab)

Menu Bar Item	Toolstrip Equivalent
Zoom > <ul style="list-style-type: none"> • Zoom In • Zoom Out • Normal View (100%) • Fit to View 	Modeling > Environment> Zoom > <ul style="list-style-type: none"> • Zoom In (Ctrl++) • Zoom Out (Ctrl+-) • Normal View (100%) (Alt+1) • Fit to View (Spacebar)
Smart Guides	Modeling > Environment > Smart Guides

Display Menu

Menu Bar Item	Toolstrip Equivalent
Interface	Modeling > Design Gallery > Model Interface
Library Links > <ul style="list-style-type: none"> • None • Disabled • User Defined • All 	Debug > Information Overlays  > <ul style="list-style-type: none"> • Hide All Links  • Disabled Links  • User-Defined Links  • Show All Links 
Sample Time > <ul style="list-style-type: none"> • All • Annotations • Colors • Off • Timing Legend 	Debug > Information Overlays  > <ul style="list-style-type: none"> • Colors  • Text  • Legend 
Function Connectors	Debug > Information Overlays  > Function Connectors 

Menu Bar Item	Toolstrip Equivalent
<p>Blocks ></p> <ul style="list-style-type: none"> • Block I/O Mismatch for Referenced Models • Block Version for Referenced Models • Sorted Execution Order • Variant Conditions • Variant Condition Legend 	<p>Debug > Information Overlays  ></p> <ul style="list-style-type: none"> • Ref. Model I/O Mismatch  • Ref. Model Version  • Execution Order  • Variant Conditions  • Variant Legend 
<p>Blocks > Tool Tip Options ></p> <ul style="list-style-type: none"> • Block Name • Parameter Names & Values • Description • Variant Condition 	<p>Debug > Information Overlays  ></p> <ul style="list-style-type: none"> • Name in Tooltip  • Parameters in Tooltip  • Description in Tooltip  • Variant Conditions 
<p>Errors & Warnings</p>	<p>Debug > Diagnostics > Edit-Time Errors & Warnings</p>

Menu Bar Item	Toolstrip Equivalent
<p>Signals & Ports ></p> <ul style="list-style-type: none"> • Signal Dimensions • Wide Nonscalar Lines • Port Data Types • Port Data Type Display Format > Alias Type, Base Type, Base and Alias Types • Port Units • Propagated Signal Labels • Design Ranges • Signal to Object Resolution Indicator • Execution Context Indicator • Storage Class • Testpoint & Logging Indicators • Viewer Indicator • Linearization Indicators • Hidden Rate Transition Block Indicators 	<p>Debug > Information Overlays  ></p> <ul style="list-style-type: none"> • Signal Dimensions  • Nonscalar Signals  • Base Data Types  • Alias Data Types  • Units  • Propagated Signal Labels  • Signal Data Ranges  • Signal Resolves to Object  • Execution Context  • Storage Class  • Log & Testpoint  • Viewers  • Linearization Indicators  • Automatic Rate Transitions 
<p>Simscape ></p> <ul style="list-style-type: none"> • Domain Styles • Legend 	<p>Debug > Information Overlays  ></p> <ul style="list-style-type: none"> • Simscape Domains  • Simscape Legends 
<p>Simscape ></p> <ul style="list-style-type: none"> • Toggle Sparklines When Clicked • Remove All Sparklines 	<p>Debug > Output Values ></p> <ul style="list-style-type: none"> • Enable Sparkline Plots • Remove Sparklines



























Menu Bar Item	Toolstrip Equivalent
Data Display in Simulation > <ul style="list-style-type: none"> Remove All Value Labels Show Value Label of Selected Port Show Value Labels When Hovering Toggle Value Labels When Clicked Options 	Debug > Output Values > <ul style="list-style-type: none"> Remove Value Displays  Show Output Value of selected signal  Options > Show When Hovering check box Toggle Value Displays Options 
Stateflow Animation	See Stateflow Release Notes.
Highlight Signal to Source	Debug > Trace Signal > Trace to Source 
Highlight Signal to Destination	Debug > Trace Signal > Trace to Destination 
Remove Highlighting	Debug > Trace Signal > Remove trace 
Hide Automatic Names	Select the block(s). Block > Name  > <ul style="list-style-type: none"> Auto On Off Hide Automatic Block Names
Hide Markup	Format > Show Markup 

Diagram Menu




Menu Bar Item	Toolstrip Equivalent
Refresh Blocks	Modeling > Update Model > Refresh Blocks  (Ctrl+K)
Subsystem & Model Reference > Create Subsystem from Selection	Select the blocks to create a subsystem. Multiple > Create Gallery > Create Subsystem
Subsystem & Model Reference > Expand Subsystem	Select the subsystem. Subsystem > Expand 
Subsystem & Model Reference > Convert to > <ul style="list-style-type: none"> Referenced Model Variant Subsystem 	Select the subsystem. Subsystem > Convert > <ul style="list-style-type: none"> Convert to Model Block Convert to Variant
Subsystem & Model Reference > Model Block Normal Mode Visibility	Simulation > Prepare Gallery > Normal Mode Visibility









Menu Bar Item	Toolstrip Equivalent
Subsystem & Model Reference > Refresh Selected Model Block	Select the Model block. Model Block > Refresh
Subsystem & Model Reference > Create Protected Model for Selected Model Block	Select the Model block. Model Block > Protect
Format > Font Style	Format
Format > Text Alignment	Double-click the text. Use the pop-up menu.
Format > <ul style="list-style-type: none"> • Enable TeX Commands • Foreground Color • Background Color • Connector Width • Shadow • Show Block Name • Content Preview • Port Labels • Canvas Color 	Format > <ul style="list-style-type: none"> • Enable Equations  • Select block. Foreground • Select block. Background • Click the text. Connector > Line Width • Select block. Shadow  • Name  • Select block. Content Preview  • Select block. Port Labels  • Click the canvas. Background
Rotate & Flip > <ul style="list-style-type: none"> • Clockwise • Counterclockwise • Flip Block • Flip Block Name 	Select the block. Format > <ul style="list-style-type: none"> • Rotate 90 clockwise  • Rotate 90 counterclockwise  • Flip left-right  • Flip up-down  • Flip Name 






Menu Bar Item	Toolstrip Equivalent
Arrange > <ul style="list-style-type: none"> • Arrange Automatically • Align • Distribute • Match • Bring to Front • Send to Back 	Format > <ul style="list-style-type: none"> • Auto Arrange  • Align  Select blocks. Align • Distribute  Select blocks. Distribute • Match  Select blocks. Match • Bring to Front  • Send to Back 
Mask > <ul style="list-style-type: none"> • Create Mask • Add Icon Image • Mask Parameters • Look Under Mask 	Select the block. Block > <ul style="list-style-type: none"> • Create Mask • Add Image • Mask Parameters • Look Under Mask
Mask > Create Model Mask	Modeling > Component Gallery > Create Model Mask
Library Link > <ul style="list-style-type: none"> • Go To Library Block • Disable Link • Resolve Link • View Changes 	Select a library linked block. Subsystem > <ul style="list-style-type: none"> • Go to Library • Disable Link • Restore Link • View Changes
Signal & Ports > <ul style="list-style-type: none"> • Signal & Scope Manager • Viewers 	Simulation > Prepare Gallery apps
Signals & Ports > <ul style="list-style-type: none"> • Input Port Signal Properties • Output Port Signal Properties 	Right-click block. Signals & Ports > <ul style="list-style-type: none"> • Input Port Signal Properties • Output Port Signal Properties
Signal & Ports > Signal Hierarchy	Click signal. Signal > Signal Hierarchy

Menu Bar Item	Toolstrip Equivalent
Block Parameters	Right-click block. Block Parameters
Properties	Right-click block. Properties







Simulation Menu


Menu Bar Item	Toolstrip Equivalent
Update Diagram	Modeling > Update Model (Ctrl+D)
Model Configuration Parameters	Modeling > Model Settings
Mode > <ul style="list-style-type: none"> • Normal • Accelerator • Rapid Accelerator 	Simulate > <ul style="list-style-type: none"> • Normal • Accelerator • Rapid Accelerator
Mode > <ul style="list-style-type: none"> • Software-in-the-Loop (SIL) • Processor-in-the-Loop (PIL) 	Apps > SIL/PIL Manager > SIL/PIL <ul style="list-style-type: none"> • SIL/PIL Mode > Software-in-the-Loop (SIL) • SIL/PIL Mode > Processor-in-the-Loop (PIL)
Mode > External	<p>A Hardware tab will appear to the left of the Apps tab for all models configured to run on hardware. To configure your system to be run on hardware, in the Modeling tab, click Model Settings. In the Hardware Implementation pane of the Configuration Parameters, choose your hardware board from the dropdown list. Once your hardware board is selected, open the Target hardware resources options and under Groups choose External Mode.</p>
Data Display > <ul style="list-style-type: none"> • Remove All Value Labels • Show Value Label of Selected Port • Show Value Labels When Hovering • Toggle Value Labels When Clicked • Options 	Debug > Output Values > <ul style="list-style-type: none"> • Remove Value Displays  • Show Output Value of selected signal  • Options > Show When Hovering check box • Toggle Value Displays • Options 


Menu Bar Item	Toolstrip Equivalent
Stateflow Animation > <ul style="list-style-type: none"> • Lightning Fast • Fast • Medium • Slow • None • Maintain Highlighting 	Open your Stateflow chart. Debug > Animation Speed > <ul style="list-style-type: none"> • Lightning Fast • Fast • Medium • Slow • None • Highlighting 
Fast Restart	Simulation > Fast Restart 
Step Back	Simulation > Step Back 
Run	Simulation > Run 
Pacing Options	Simulation > Run > Simulation Pacing 
Step Forward	Simulation > Step Forward 
Stop	Simulation > Stop 
Output > <ul style="list-style-type: none"> • Simulation Data Inspector • Logic Analyzer • Bird's-Eye Scope 	Simulation > Review Results Gallery > <ul style="list-style-type: none"> • Data Inspector • Logic Analyzer • Bird's-Eye Scope
Output > <ul style="list-style-type: none"> • Log Selected Signals • Log Chart Signals • Configure Logging 	Simulation > Prepare Gallery <ul style="list-style-type: none"> • Log Signals • In R2019b, the Stateflow Signal Logging dialog box is no longer available. To log multiple signals from your Stateflow chart, see the Stateflow release notes. • Configure Logging
Stepping Options >	Simulation > Step Back > Configure simulation stepping 


Menu Bar Item	Toolstrip Equivalent
Debug > <ul style="list-style-type: none"> • Add Conditional Breakpoint • Conditional Breakpoints List • Debug Model 	Debug > <ul style="list-style-type: none"> • Add Breakpoint  • Breakpoints List  • Breakpoints List > Debug Model 
Debug > Stateflow Breakpoints and Watch Window	Open your Stateflow chart. Debug > Breakpoints List 
Debug > Simulation Target for MATLAB and Stateflow	Modeling > Model Settings > Simulation Target
Debug > <ul style="list-style-type: none"> • Set Stateflow Breakpoints • Clear Stateflow Breakpoints 	Open your Stateflow chart. Debug > <ul style="list-style-type: none"> • Add Breakpoint  • Breakpoints List > Clear Breakpoints
Debug > MATLAB and Stateflow Error Checking Options > Detect Cycles	Debug > Diagnostics > Detect Cyclical Behavior





Analysis Menu

Menu Bar Item	Toolstrip Equivalent
Model Advisor > <ul style="list-style-type: none"> • Display Advisor Checks in Editor • Model Advisor • Model Advisor Dashboard • Configure Advisor Edit-Time Checks • Preferences • Upgrade Advisor 	Modeling > Model Advisor  > <ul style="list-style-type: none"> • Edit-Time Checks • Model Advisor  • Model Advisor Dashboard  • Customize Edit-Time Checks  • Preferences  • Upgrade Advisor 
Metrics Dashboard	Apps > Metrics Dashboard
Refactor Model > <ul style="list-style-type: none"> • Identify Modeling Clones • Model Transformer 	Apps > <ul style="list-style-type: none"> • Clone Detector • Model Transformer
Refactor Model > Model Transformer	Apps > Model Transformer
Model Slicer	Apps > Model Slicer




Menu Bar Item	Toolstrip Equivalent
Model Dependencies > Model Dependency Viewer > <ul style="list-style-type: none"> • Models & Libraries • Models Only • Referenced Model Instances 	Modeling > Design Gallery > Model Dependencies
Model Dependencies <ul style="list-style-type: none"> • Generate Manifest • Edit Manifest Contents • Compare Manifests • Export Files in Manifest 	Modeling > Compare  > Manifest  > <ul style="list-style-type: none"> • Generate Manifest • Edit Manifest • Compare Manifests • Export Files in Manifest
Compare To	Modeling > Compare > Compare Models 
Simscape > <ul style="list-style-type: none"> • Statistics Viewer • Variable Viewer 	Apps > <ul style="list-style-type: none"> • Simscape Statistics Viewer • Simscape Variable Viewer
Performance Tools > <ul style="list-style-type: none"> • Show Profiler Report • Performance Advisor • Solver Profiler 	Debug > Performance Advisor > <ul style="list-style-type: none"> • Activate Profiler • Performance Advisor • Solver Profiler
Requirements > <ul style="list-style-type: none"> • Requirements Perspective • Requirements Editor • Highlight Model 	Apps > Requirements Manager > <ul style="list-style-type: none"> • Requirements • Requirements > Requirements Editor  • Requirements > Highlight Links 
Requirements > Reports > <ul style="list-style-type: none"> • Check Consistency • Generate Report • Generate Web View 	Apps > Requirements Manager > <ul style="list-style-type: none"> • Check Consistency  • Share > Generate Model Traceability Report  • Share > Export Model to Web View 








Menu Bar Item	Toolstrip Equivalent
Requirements > Requirements at This Level > <ul style="list-style-type: none"> • Link to Selection in MATLAB • Link to Current Test Case • Link to Selection in Word • Link to Selection in Excel • Select for Linking with Simulink • Add Link to Selected Object(s) • Open Outgoing Links dialog • Copy URL to Clipboard 	Right-click the canvas. Requirements at This Level > <ul style="list-style-type: none"> • Link to Selection in MATLAB • Link to Current Test Case • Link to Selection in Word • Link to Selection in Excel • Select for Linking with Simulink • Add Link to Selected Object(s) • Open Outgoing Links dialog • Copy URL to Clipboard
Requirement > Links File > <ul style="list-style-type: none"> • Load Links • Save Links • Copy to Model 	Apps > Requirements Manager > <ul style="list-style-type: none"> • Load Links  • Save All > Save Links Only • Links Settings > Save Links in Model File
Requirements > Settings	Apps > Requirements Viewer > Link Settings
Control Design > <ul style="list-style-type: none"> • Steady State Manager • Linear Analysis • Frequency Response Estimation • Control System Designer • Control System Tuner • Model Discretizer • Linearize Block • Specify Selected Block Linearization 	Apps > <ul style="list-style-type: none"> • Steady State Manager • Model Linearizer • Frequency Response • Control System Designer • Control System Tuner • Model Discretizer • Linearization Manager > Linearize Block • Linearization Manager > Specify Block Linearization





Menu Bar Item	Toolstrip Equivalent
Control Design > Linear Analysis Points > <ul style="list-style-type: none"> • Open-loop Input • Open-loop Output • Loop Transfer • Loop Break • Input Perturbation • Output Measurement • Sensitivity • Complementary Sensitivity • Trim Output Constraint • Help Me Select 	Right-click signal. Apps > Linearization Manager > <ul style="list-style-type: none"> • Open-loop Input • Open-loop Output • Loop Transfer • Loop Break • Input Perturbation • Output Measurement • Sensitivity • Complementary Sensitivity • Trim Output Constraint • Help with Selecting Analysis Point
Parameter Estimation	Apps > Parameter Estimator
Response Optimization	Apps > Response Optimizer
Sensitivity Analysis	Apps > Sensitivity Analyzer
Test Harness > <ul style="list-style-type: none"> • Create for Model • Create for Selection • Import for Model • Import for Selection • Manage Test Harnesses 	Apps > Simulink Test > <ul style="list-style-type: none"> • Add Test Harness  • Import Test Harness  • Manage Test Harnesses 
Observers > <ul style="list-style-type: none"> • Go to Observer Block • Add Observer Here • Add Observer Port Here 	Right-click the canvas. Observers > <ul style="list-style-type: none"> • Go to Observer Block • Add Observer Here • Add Observer Port Here
Test Manager	Apps > Simulink Test > Simulink Test Manager






Menu Bar Item	Toolstrip Equivalent
Design Verifier <ul style="list-style-type: none"> • Check Compatibility • Detect Design Errors • Generate Tests • Prove Properties • Results • Options 	Apps > Design Verifier > <ul style="list-style-type: none"> • Check Compatibility  • Toggle Mode to Design Error Detection . Click Detect Design Errors. • Toggle Mode to Test Generation . Click Generate Tests. • Toggle Mode to Property Proving . Click Prove Properties. • Results • Test Generation Settings > Settings
Design Verifier > Model Slicer	Apps > Model Slicer
Coverage > Open Results Explorer <ul style="list-style-type: none"> • Open Results Explorer • Settings 	Apps > Coverage Analyzer > <ul style="list-style-type: none"> • Results Explorer • Settings
Data Type Design > <ul style="list-style-type: none"> • Fixed-Point Tool • Single Precision Converter • Lookup Table Optimizer 	Apps > <ul style="list-style-type: none"> • Fixed-Point Tool • Single Precision Converter • Lookup Table Optimizer

Code Menu


Menu Bar Item	Toolstrip Equivalent
C/C++ Code > Code Generation Report > <ul style="list-style-type: none"> • Open Model Report • Open Subsystem Report • Options 	<p>These options can be found by using Embedded Coder, Simulink Coder, or the AUTOSAR Component Designer. The path is shown using Embedded Coder. For Simulink Coder or AUTOSAR Component Designer, select those apps from the Apps gallery.</p> <p>Apps > Embedded Coder > C Code ></p> <ul style="list-style-type: none"> • View Code  • Open Latest Report  • Open Latest Report > Report Options 



Menu Bar Item	Toolstrip Equivalent
<p>C/C++ Code ></p> <ul style="list-style-type: none"> • Embedded Coder Quick Start • Code Generation Advisor • Code Generation Options • Configure Model in Code Perspective • Build Model • Embedded Coder Dictionary • Navigate to C/C++ Code 	<p>These options can be found by using Embedded Coder, Simulink Coder, or the AUTOSAR Component Designer. The path is shown using Embedded Coder. For Simulink Coder or AUTOSAR Component Designer, select those apps from the Apps gallery.</p> <p>Apps > Embedded Coder > C Code ></p> <ul style="list-style-type: none"> • Quick Start • C/C++ Code Advisor  • Settings > Code Generation Settings • Settings > Code Perspective Help check box • Build  • Settings > Embedded Coder Dictionary  (Embedded Coder only) • View Code 
<p>C/C++ Code > Build Selected Subsystem</p>	<p>Right-click the subsystem. C/C++ Code > Build This Subsystem</p>
<p>C/C++ Code ></p> <ul style="list-style-type: none"> • Export Functions • Generate S-Function 	<p>Right-click the function or S-function. C/C++ Code ></p> <ul style="list-style-type: none"> • Export Functions • Generate S-Function
<p>HDL Code ></p> <ul style="list-style-type: none"> • Add HDL Coder Configuration to Model • HDL Workflow Advisor • Options • Generate HDL • Generate Test Bench • Remove HDL Coder Configuration from Model 	<p>Apps > HDL Coder > HDL Code ></p> <ul style="list-style-type: none"> • Settings > Add HDL Configuration to Model • Workflow Advisor  • Settings > HDL Code Generation Settings • Generate HDL Code  • Generate Test Bench  • Settings > Remove HDL Configuration to Model

Menu Bar Item	Toolstrip Equivalent
PLC Code > <ul style="list-style-type: none"> • Check Subsystem Compatibility • Generate Code for Subsystem • Generate and Import Code for Subsystem • Generate, Import, and Verify Code for Subsystem • Navigate to Code • Options 	Apps > PLC Coder > PLC Code > <ul style="list-style-type: none"> • Check Compatibility • Generate PLC Code  • Settings > Import Code into IDE • Settings > Verify Code in IDE • Navigate to Code • Settings > PLC Code Generation Settings
Data Objects > <ul style="list-style-type: none"> • Data Object Wizard • Custom Storage Class Designer 	Modeling > Design Gallery > <ul style="list-style-type: none"> • Data Object Wizard • Custom Storage Class Designer
External Mode Control Panel	Apps > Run on Custom Hardware > Hardware > Prepare Gallery > Control Panel
Simulink Code Inspector	Apps > Code Inspector
Verification Wizards > <ul style="list-style-type: none"> • HDL Co-Simulation • FPGA-in-the-Loop (FIL) 	Apps > <ul style="list-style-type: none"> • HDL Cosimulator • FIL Wizard
Polyspace > <ul style="list-style-type: none"> • Verify Code Generated for • Annotate Selected Block • Options • Remove Options from Current Configuration • Open Results • Open Job Monitor • Open Metrics 	Apps > Polyspace Code Verifier > <ul style="list-style-type: none"> • Analyze Code from • Add Annotation • Settings > Polyspace Settings • Settings > Remove Polyspace Configuration from Model • Review Results Gallery > Analysis Results • Review Results Gallery > Remote Job Monitor • Review Results Gallery > Code Quality Metrics
DO Qualification Kit > <ul style="list-style-type: none"> • Artifacts Explorer • Bug Reports • Traceability Matrix 	Apps > DO Qualification Kit > <ul style="list-style-type: none"> • Artifacts Explorer  • Bug Reports  • Traceability Matrix 

Menu Bar Item	Toolstrip Equivalent
DO Qualification Kit > Qualification Tests <ul style="list-style-type: none"> Polyspace Bug Finder Polyspace Code Prover Simulink Report Generator Simulink Model Comparison Simulink Check Simulink Code Inspector Simulink Coverage Simulink Design Verifier Test Requirements 	Apps > DO Qualification Kit > Run Qualification Tests  > <ul style="list-style-type: none"> Polyspace Bug Finder Polyspace Code Prover Simulink Report Generator Simulink Model Comparison Simulink Check Simulink Code Inspector Simulink Coverage Simulink Design Verifier Simulink Test Simulink Requirements
IEC Certification Kit > <ul style="list-style-type: none"> Artifacts Explorer Bug Reports Traceability Matrix 	Apps > IEC Certification Kit > <ul style="list-style-type: none"> Artifacts Explorer  Bug Reports  Traceability Matrix 
IEC Certification Kit > Validation Tests > <ul style="list-style-type: none"> Polyspace Bug Finder Polyspace Code Prover Embedded Coder Simulink Check Simulink Coverage Simulink Design Verifier Simulink Test Simulink Requirements 	Apps > IEC Certification Kit > Run Validation Tests  > <ul style="list-style-type: none"> Polyspace Bug Finder Polyspace Code Prover Embedded Coder Simulink Check Simulink Coverage Simulink Design Verifier Simulink Test Simulink Requirements

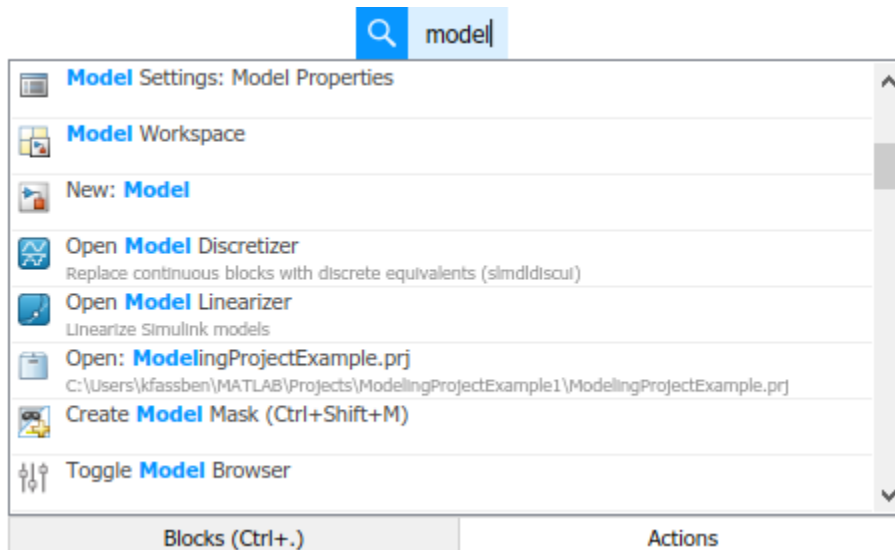
Tools Menu

Menu Bar Item	Toolstrip Equivalent
Library Browser	Simulation > Library Browser 
Model Explorer	Modeling > Design Gallery > Model Explorer
Report Generator	Apps > Report Generator

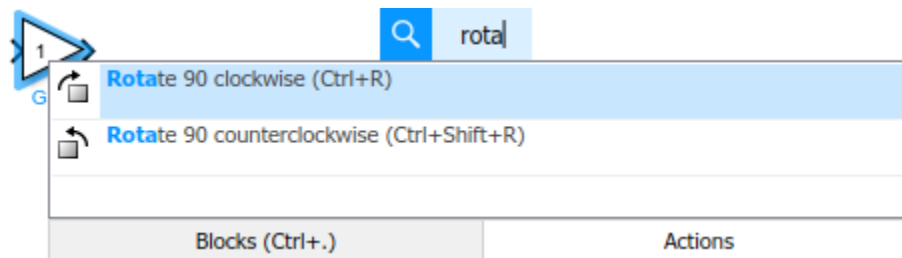
Menu Bar Item	Toolstrip Equivalent
Requirements Editor	Apps > Requirements Manager > Requirements Editor 
MPlay Video Viewer	Apps > Video Viewer
Robot Operating System > <ul style="list-style-type: none"> • Configure Network Addresses • Manage Array Sizes 	Open your ROS model. Simulation > Prepare Gallery > <ul style="list-style-type: none"> • ROS Network • Variable Size Messages
Simulink Real-Time	Apps > Simulink Real-Time
Run on Target Hardware > <ul style="list-style-type: none"> • Prepare to Run • Install/Update Support Package 	Apps > <ul style="list-style-type: none"> • Run on Hardware Board • Get Hardware Support Packages
SoC Builder	Open your SoC project. System on Chip > Configure, Build & Deploy
Help	Use the quick access toolbar 

Find actions to perform using a Simulink Editor search tool

You can now access actions in the Simulink Editor using the **Ctrl+.** shortcut.



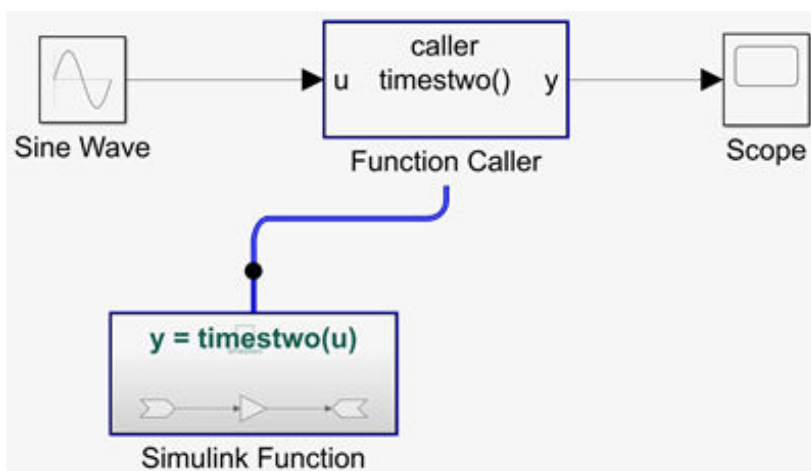
To enable context-sensitive search, select a part of the model before using the action menu shortcut.



Animation for messages and Simulink functions

Use animation to highlight and animate function calls and the flow of messages between blocks.

This example shows a still of an animated Simulink function call.



To access animation, in a model window, right-click and select **Animation Speed**, and specify the speed of the animation as **Slow**, **Medium**, or **Fast**.

Invoke text editing for a selection using a keyboard shortcut

Use **F2** on Linux and Windows platforms and **command+Enter** on Macintosh platforms to edit block, line, area, and annotation text.

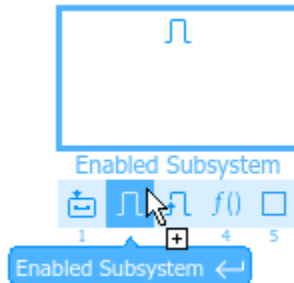
For more keyboard shortcuts, see [Keyboard and Mouse Actions for Simulink Modeling](#).

Update line routing affected by scripts

When using a script to update a model, lines can overlap other lines and obstacles in the model. You can now improve line routing using the model parameter `LineUpdate` with values `'immediate'`, `'deferred'`, or `'none'` (default). When set to `'deferred'`, line routing is deferred until the script is paused or complete. To perform deferred line routing within a script, use function `Simulink.BlockDiagram.routeLine` or save a model with function `save_system`.

Create conditionally executed subsystems from empty selections

When you drag a selection box around an empty area of the canvas, you can now create enabled, triggered, and function-call subsystems in addition to the previously available subsystems and areas.

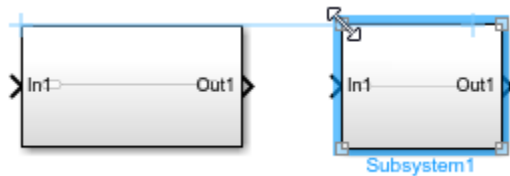


For efficiency, you can use these shortcuts.

- To select the highlighted option, you can click the selection box. For example, after drawing a selection box, click it to create a subsystem.
- To select the highlighted option, you can press **Enter**. For example, after navigating to an option with the arrow keys, press **Enter** to create the displayed option.
- To select an option without navigating to it, press the number associated with the option. For example, to create a function-call subsystem, press **4**.

Align edges while resizing blocks and annotations

While resizing blocks and annotations, smart guides now help you align their edges.



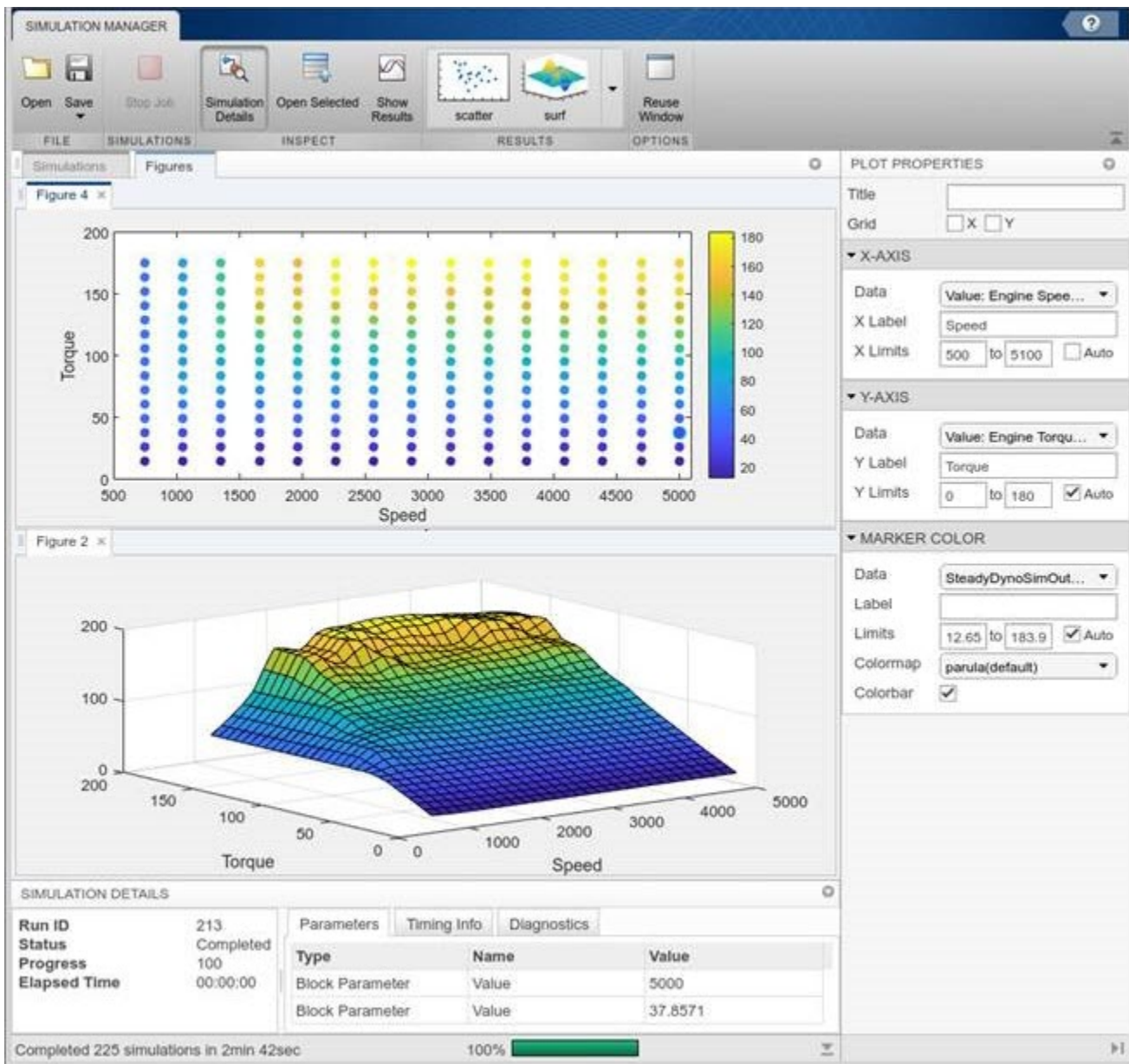
Simulation Analysis and Performance

Simulation Manager: Configure simulation visualization and save the simulation sessions

View the status of simulation input parameters and outputs as the simulations are progressing. Design view allows you to spot trends in simulations and determine if the simulations should continue.

The design view enables you to select a scatter plot or a surface plot to visualize simulation data including inputs, outputs, and status. You can configure the plot properties, such as plot labels, color maps, and plot axes to suit your visualization needs.

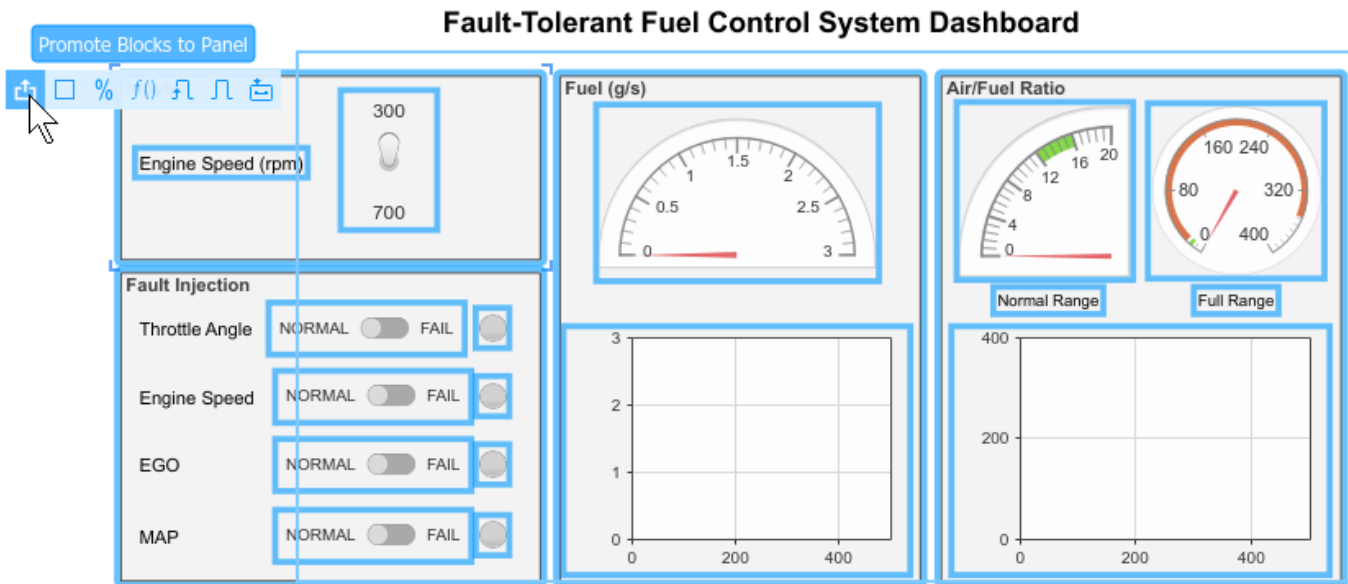
Simulation Manager now also allows you to save your simulation results and plot configurations in a file to reuse with new simulations.



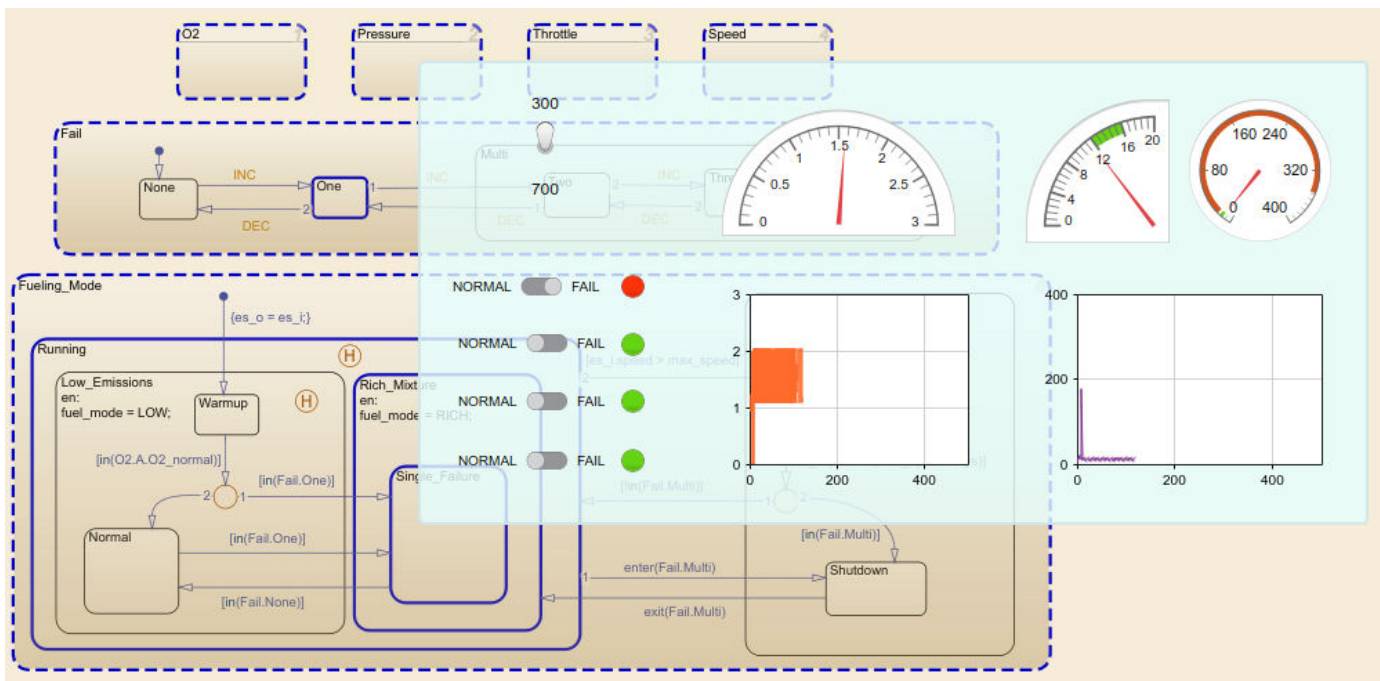
Dashboard Panels: Test and debug your models using a dashboard that is accessible throughout the model hierarchy

Promote a group of Dashboard blocks into a panel to create a dashboard of controls and indicators that overlays your model and is displayed consistently throughout a model hierarchy. The panel remains in the same place when you navigate into a referenced model, subsystem, or Stateflow chart.

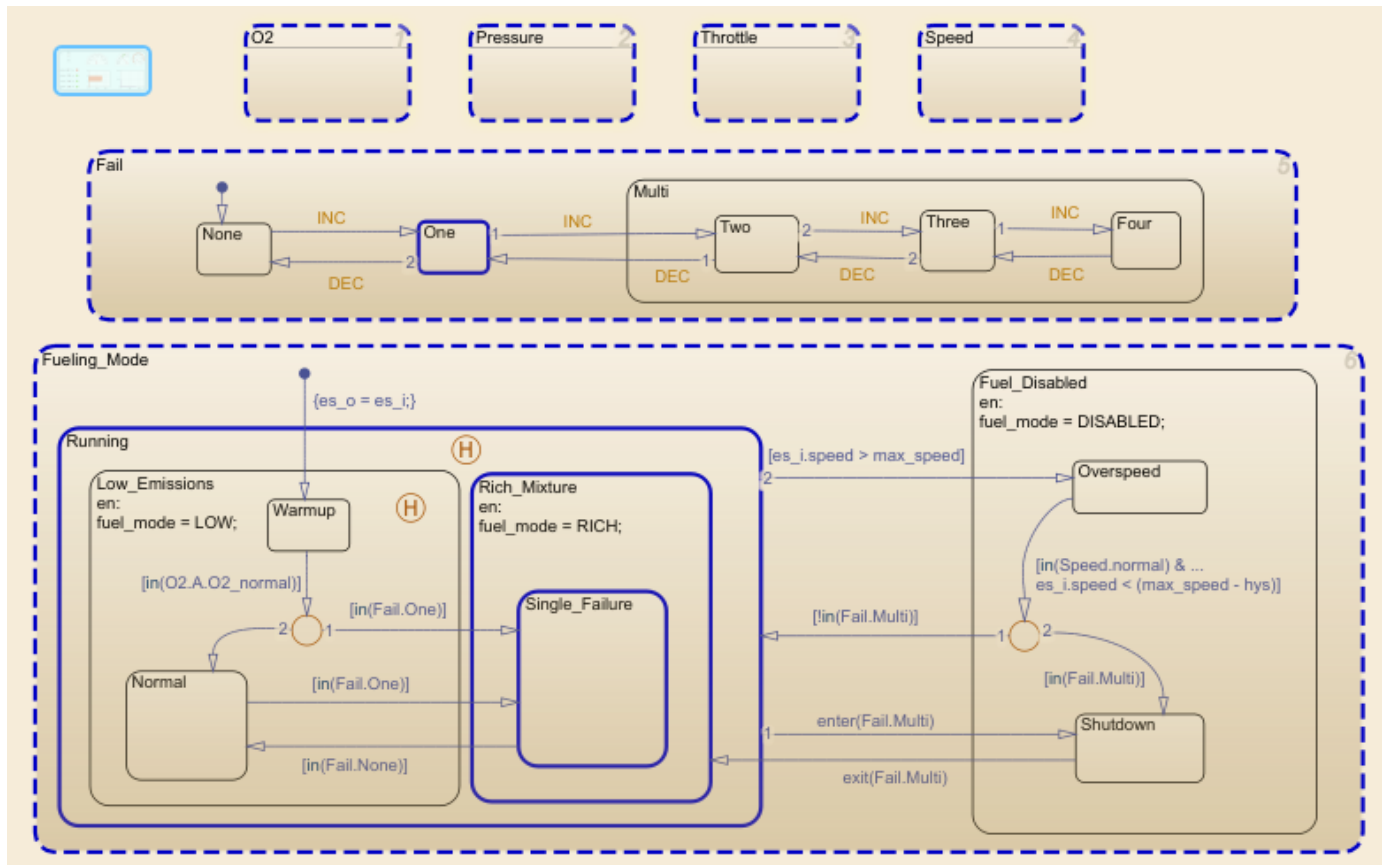
To create a panel, select one or more Dashboard blocks in your model, pause on the ellipsis that appears above the selection, and click the **Promote Blocks to Panel** button.



The blocks are promoted to a panel that overlays the model in the canvas. You can navigate through the model hierarchy, including into a Stateflow chart, and the panel remains as an overlay in the same position. Click and drag the panel to change its position.



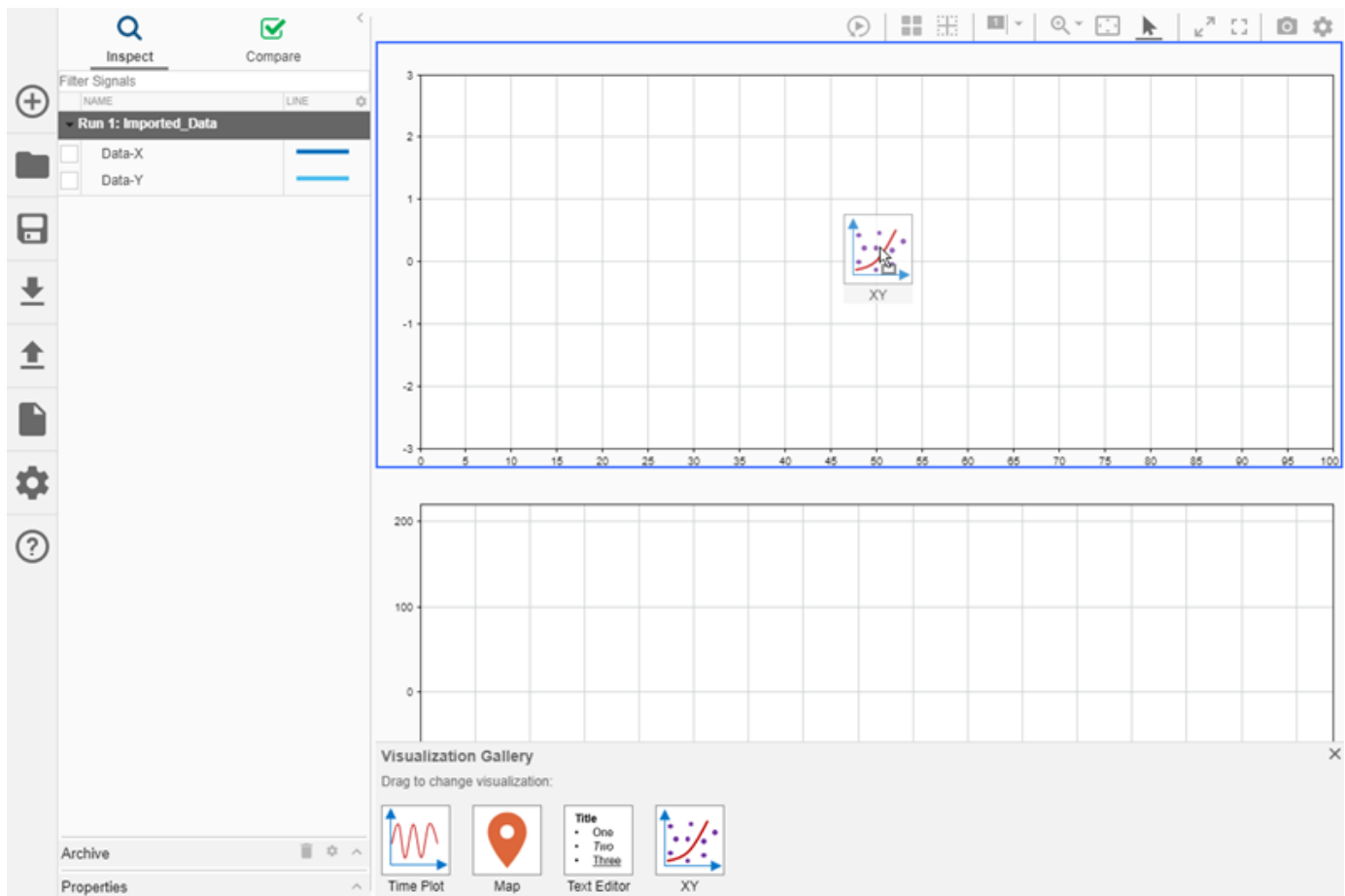
When you are not using a panel, you can double-click to minimize it. The panel remains visible in the model and takes up less space. When you want to use the panel, double-click it again.



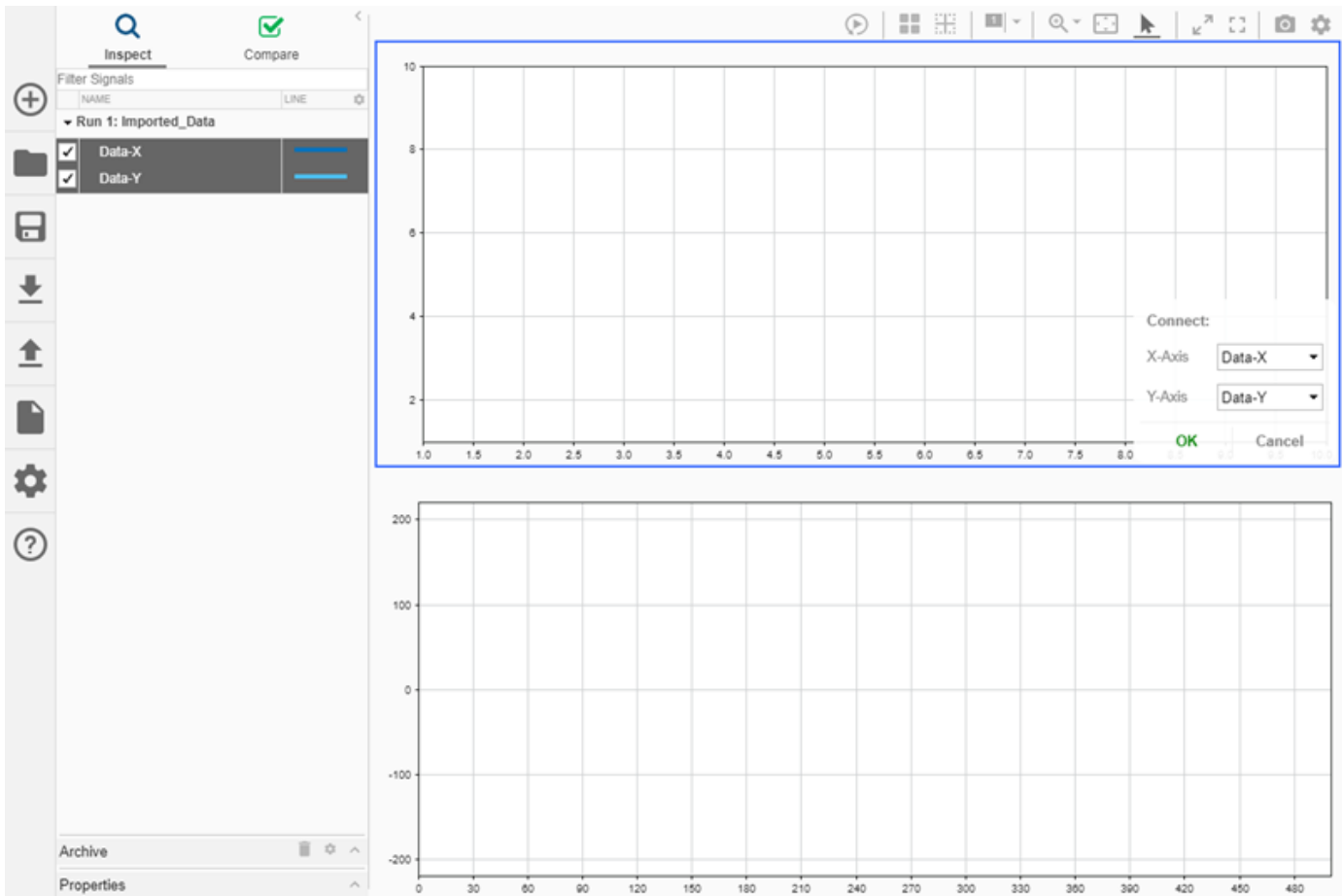
XY Plots: Visualize relationships between signals in the Simulation Data Inspector

To visualize relationships between logged signals, you can add an XY visualization to your layout in the Simulation Data Inspector. You can view logged signal data during simulation. Using the **XY Visualization Settings**, you can choose to display the data as a line or scatter plot, customize the plot appearance, and choose to show a trend line.

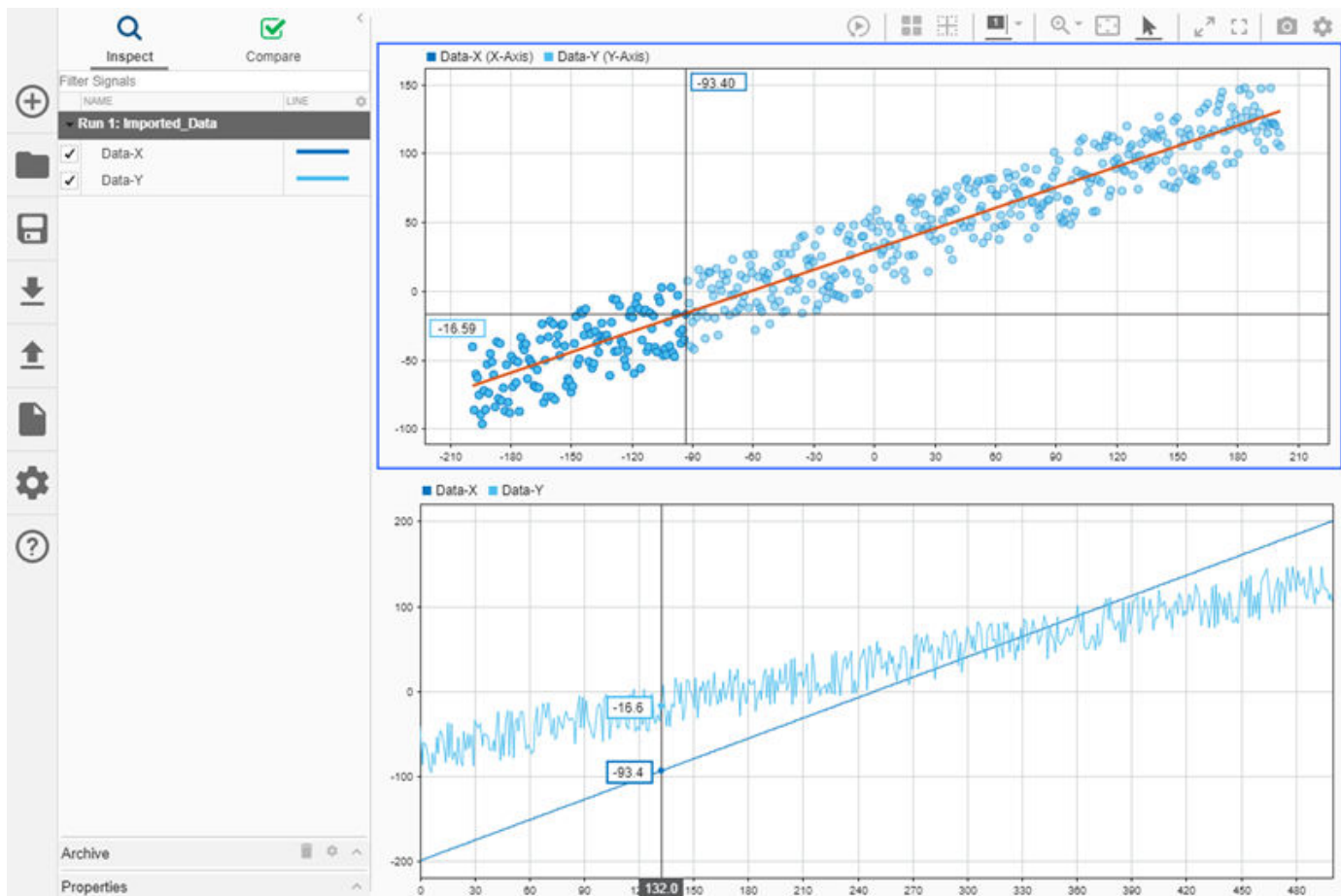
To add an XY visualization to your layout, expand the **Layout** menu and select **Edit View**. Then, drag and drop the **XY** icon onto the subplot where you want the XY plot.



To plot data on the XY plot, select the signals corresponding to your x and y data. Then, drag and drop them onto the XY plot. Specify which signal corresponds to the x data and which corresponds to the y data in the menu that appears in the lower-right corner.



You can add a cursor to inspect the data. The cursor in the XY plot is synchronized with the Time Plot cursor.



Connect Dashboard blocks in your model during simulation

You can connect Dashboard blocks in your model to parameters, variables, and signals in your model during normal and accelerator mode simulations. You can modify connections and add new connections. For example, you can modify a Display block in your model to connect to an upstream signal, or you can connect additional signals to a Dashboard Scope block in your model.

Connecting a Dashboard block to a signal does not mark the signal for logging

Connecting a Dashboard block to a signal in your model no longer automatically marks that signal for logging. You can choose whether you want to log data to the workspace or a file independent of Dashboard block connections in your model. Data for signals, parameters, and variables connected to Dashboard blocks always logs to the Simulation Data Inspector.

Save Simulink.SimulationData.Dataset objects to a MAT-file you can use in earlier releases

When you use the `exportToPrevious` method to save data in a `Simulink.SimulationData.Dataset` object to a MAT-file, you can load the data in an earlier release. The save function does not support cross-release compatibility for Dataset objects.

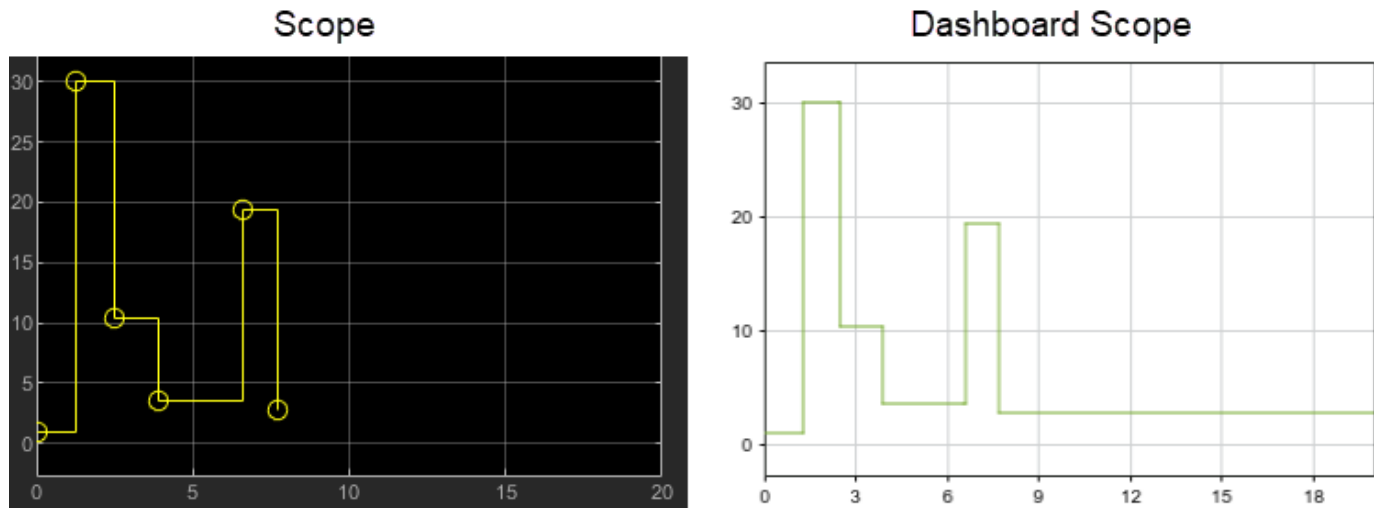
Constant signal logging enhancement

Signal logging saves a single time point at value transitions for signals with constant sample time. In previous releases, signals with constant sample time were logged at the fastest discrete rate in the model, resulting in many more logged points.

Version History

When you simulate a model from an earlier release that logs a signal with constant sample time, the logging variable for that signal contains fewer data points.

The Scope block displays signals with constant sample time by holding values between value transitions. The **Simulation Data Inspector** and the Dashboard Scope block display signals with constant sample time by holding values throughout the simulation.



Programmatically plot data for any simulation output object in the Simulation Data Inspector

The plot functions for simulation output objects now have consistent default behavior to plot data in the Simulation Data Inspector. The `plot` function for the `Simulink.SimulationData.Dataset` and `Simulink.SimulationData.DatasetRef` objects has updated default behavior.

A new `plot` function provides the same behavior for these objects:

- `Simulink.SimulationOutput`
- `Simulink.SimulationData.DataStoreMemory`
- `Simulink.SimulationData.Signal`
- `Simulink.SimulationData.State`
- `Simulink.SimulationData.Parameter`
- `sltest.Assessment`

The `plot` function for the `simscape.logging.Node` object now has an option to plot the data in the Simulation Data Inspector.

For example, simulate the `vdp` model, saving the output data in `Dataset` format, and plot the results:

```
simOut = sim('vdp', 'SaveOutput', 'on', 'SaveFormat', 'Dataset');  
plot(simOut)
```

You can also use the `Simulink.sdi.plot` function to programmatically plot any of the listed objects as well as `timeseries` and `timetable` data.

Save selected `Simulink.sdi.Run` objects to an `MLDATX`-file

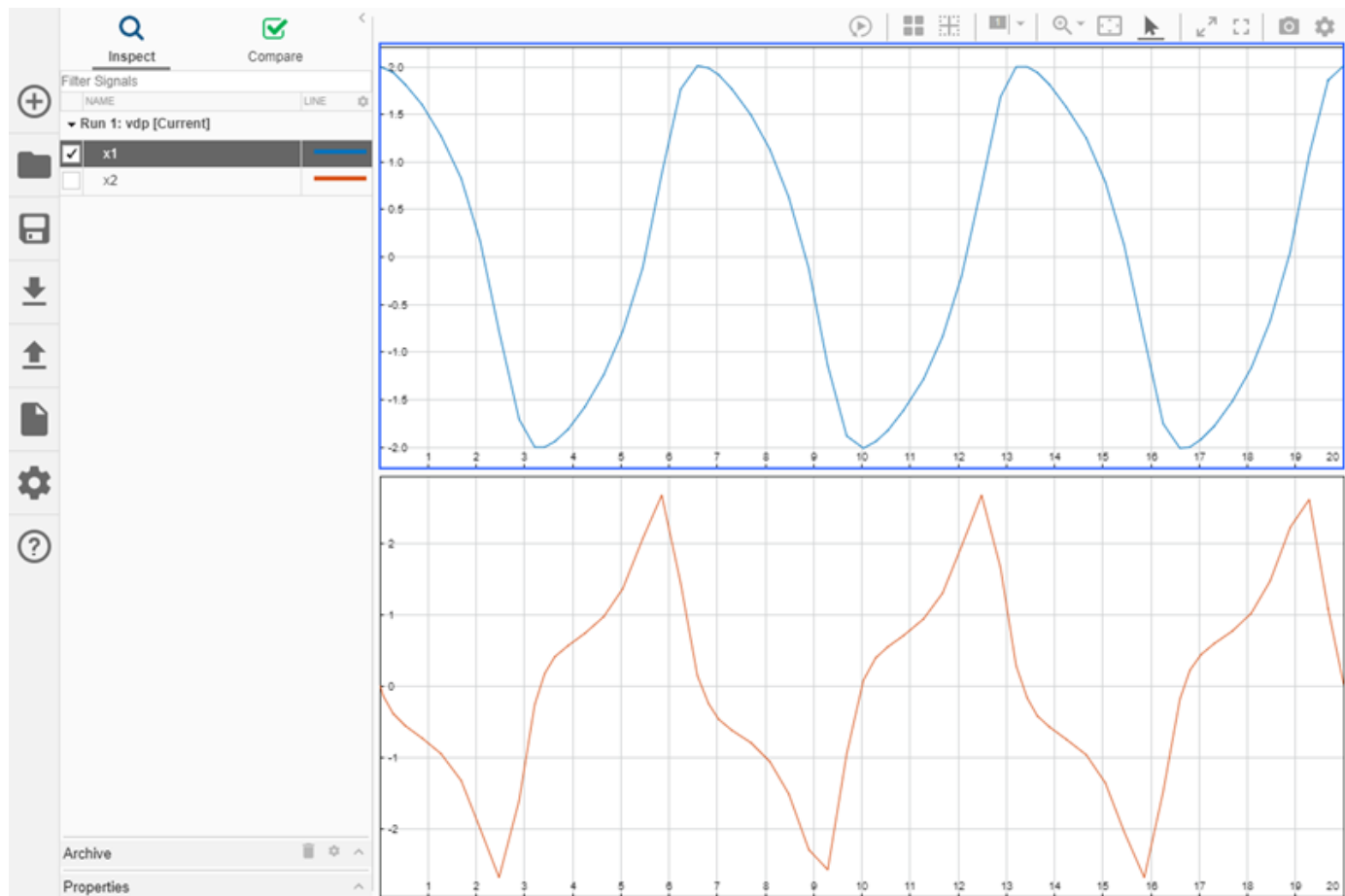
You can save one or more selected `Simulink.sdi.Run` objects to an `MLDATX`-file using the `export` method for the `Run` object or the `Simulink.sdi.exportRun` function. To view the data saved in the `MLDATX`-file, you can open the `MLDATX` file using the Simulation Data Inspector UI or you can load it using the `Simulink.sdi.load` function.

Data is stored in the Simulation Data Inspector as `Simulink.sdi.Run` objects. The `Run` object includes metadata associated with the run as well as `Simulink.sdi.Signal` objects containing the signal data. When you export data from the Simulation Data Inspector to a `MAT`-file, the data is saved as a `Simulink.SimulationData.Dataset` object that does not include the same set of metadata.

Additional styling controls for time plots in the Simulation Data Inspector

New options in the Simulation Data Inspector allow you to customize the appearance of time plots. You can independently hide or show the horizontal and vertical grid lines, choose to hide the plot border, and specify the visibility and position for the tick labels and legend.

For example, you can maximize the area in the time plot available to display the signal when you move the tick labels inside the axes and hide the legend.



Simulation Data Inspector signal property enhancements

Messages are displayed in the Simulation Data Inspector using the new **Interpolation** option none. You can also view and modify the **Description** property for signals logged from Simscape.

Additional information in Simulink.sdi.DiffRunResult and Simulink.sdi.DiffSignalResult objects

The `Simulink.sdi.DiffRunResult` and `Simulink.sdi.DiffSignalResult` objects include additional properties containing information about comparison results.

`Simulink.sdi.DiffSignalResult` objects now include a `Status` property that provides information about the signal comparison status, such as whether the comparison results were within tolerance. The `Diff` property of the `Simulink.sdi.DiffSignalResult` object is the data for the actual difference. In previous releases, the `Diff` property was the data for the absolute difference.

`Simulink.sdi.DiffRunResult` objects now include these properties:

- `GlobalTolerance` — Structure containing global tolerance values used for a run comparison.
- `Summary` — Indicates the number of signals in the run with a given status.

Hide Simulation Data Inspector Getting Started pop-up

Specify whether the Simulation Data Inspector displays the Getting Started module on start up when empty using the **Do not show this again** option in the Getting Started module or the `Simulink.sdi.showGettingStarted` function.

Data logging and signal viewer block support for export function models

R2019b adds logging and signal viewer block support for export function models. You can use logging and signal viewer blocks to test export-function models as top or referenced models and in the context of a test harness. For examples, see [Test Export-Function Model Simulation Using Function-Call Generators](#), [Test Export-Function Model Simulation Using Stateflow Chart](#), and [Test Export-Function Model Simulation Using Schedule Editor](#).

FMU Import block model reference support

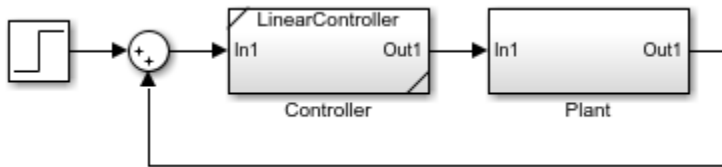
FMU Import block supports model reference for accelerator and rapid accelerator modes for simulation.

Component-Based Modeling

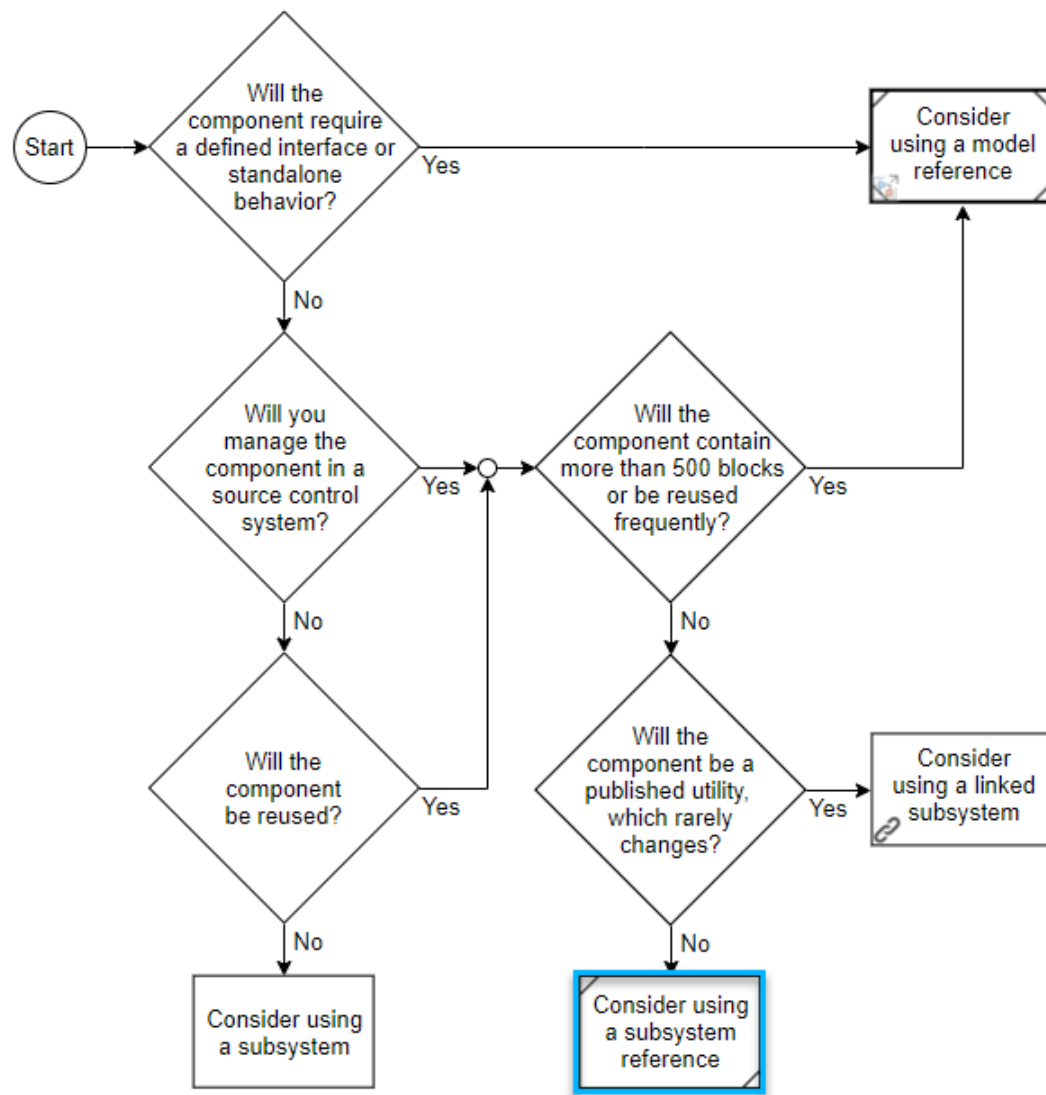
Subsystem Reference: Save subsystem contents in a file that you can reference in a model

In R2019b, you can save the contents of a subsystem in an SLX file and reference it using a Subsystem Reference block. This feature helps you manage individual subsystems as model components. You can use multiple Subsystem Reference blocks to reference the same subsystem file.

You can identify a subsystem reference by the two triangles at opposite corners of the block icon.



This flow chart provides a starting point for deciding whether to use a subsystem reference.



For more guidelines on component selection, see [Component-Based Modeling Guidelines](#) and its related topics.

To learn more about referenced subsystems, see [Subsystem Reference](#).

Messages: Model and generate C++ code for software compositions with message-based communication

You can now send and receive messages from a model's root input and output port and generate C/C++ code to connect to a middleware or an operating system communication API. For an example, see [Message-Based Communication Between Software Components](#).

Use message-based communication between model components. Messages are useful modeling artifacts that combine events with related data. Message blocks provided in the Messages & Events library manage production, consumption, and storage of messages.

Using blocks from the Messages & Events library with the blocks from the SimEvents library facilitates more complex message-based communication policies. For more information, see Modeling Message Communication Patterns with SimEvents.

Simulink Cache: Supports code generation and Stateflow

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

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

Variant Configuration Analysis Tool: Enhanced functionalities for better usability

From R2019b, you can use these enhanced features in the Variant Configuration Analysis tool:

- View variant conditions for the block in given variant configuration.
- Hide variant configuration analysis report.

For more information, see Analyze Variant Configurations in Models Containing Variant Blocks.

Execution Order Viewer

Determine execution order within tasks. Display execution order numbers on model blocks for each task. See Task-Based Sorting to Determine Block Execution Order.

Simulink Units Updates

The Simulink units capability has these changes:

- The Constant block now inherits units from the `Simulink.Parameter` object **Unit** property. In previous releases, the block did not inherit units from the `Simulink.Parameter` object.
- The unit from a `Simulink.Signal` object now specifies the incoming unit for the Data Store Read and Data Store Write blocks for a Data Store Memory area. In previous releases, these blocks ignored the unit from the `Simulink.Signal` object.
- Simulink now supports automatic unit conversions at the boundaries of virtual subsystems.

Global data on parameters are guarded

Starting in R2019b, parameters are guarded in the generated code with their corresponding variant conditions in Simulink models that contain Variant blocks. For more information, see Global Data Guarding: Parameters (Embedded Coder).

Before R2019b, parameter guarding in generated code was supported only in Simulink models that contained Variant Subsystem blocks.

Functionality being removed or changed

Rebuild settings ignore cosmetic changes

Behavior change

In R2019b, the **Rebuild** configuration parameter setting *If any changes detected* ignores cosmetic changes, such as repositioning a block.

Project and File Management

Model Comparison: Compare and three-way merge MATLAB Function blocks

In R2019b, Simulink model comparison supports three-way merge for MATLAB Function blocks. Use three-way model merge to view conflicts and merge selected changes in Function blocks.

For more information, see [Resolve Conflicts Using Three-Way Model Merge](#).

Projects: Delete project definition files

You can now use `matlab.project.deleteProject` to easily stop managing your folder with a project and delete all related project definition files without affecting the remaining files.

Compare Git Branches: Show differences and save copies

In a project under Git source control, you can now select any two revisions and examine file differences. You can show differences between two development branches and save a copy of the selected file on either branch. See [Compare Branches and Save Copies](#).

Simulink Models Files: Unset compression to reduce repository size

To reduce Git repository size, you can now save individual SLX files without compression. For more information, see [Set SLX Compression Level](#).

Data Management

Simulink data dictionary version handling

In R2019b, when you save a data dictionary in a newer version of Simulink and load the dictionary in an older version of Simulink, an error prompts you to export the data dictionary to the older version. For more information, see [Dictionary Usage for Models Created with Different Versions of Simulink](#).

Version History

Previously, when you saved a data dictionary in a newer version of Simulink and loaded it in an older version, Simulink synchronized the data in the dictionary for use with the older version.

Root Inport Mapper Updates

The Root Inport Mapper has these changes:

- The tool no longer uses the Comparison Tool to help resolve issues. Instead, messages in the **SIMULATION READINESS** section describe the issue.
- The tool can now map inputs for models that contain bus element port blocks.

createInputDataset function supports bus elements

The `createInputDataset` can now create dataset objects for models that contain bus element port blocks.

Signal Editor user interface updates

Signal Editor user interface freehand draw capability (**Draw** tab) now supports:

- Multi-touch gestures on a touchscreen — Use multi-touch gestures to pan, pinch, tap, and double-tap. In R2019a, tap and pan gestures were supported. You can now also pinch and double-tap.
- Enumerations — Interact with and plot enumerated data

For more information, see [Create Freehand Signal Data Using Mouse or Multi-Touch Gestures](#).

Edit stored integer values for fixed-point data objects


For fixed-point data objects, you can now view and edit the stored integer minimum and maximum values as well as the real-world values. When the data type of a `Simulink.Parameter`, `Simulink.Signal`, `Simulink.Breakpoint`, or `Simulink.LookupTable` object evaluates to a valid fixed-point type, the property dialog box displays the derived properties **Stored Integer Minimum** and **Stored Integer Maximum**.

For more information see:

- [Use Simulink.Signal Objects to Specify and Control Signal Attributes](#)
- [Simulink.Parameter Property Dialog Box](#)

- `Simulink.Breakpoint`
- `Simulink.LookupTable`

Create variable from numeric expression for block parameters

When a block parameter value is set to a simple numeric expression, you can create a variable for the expression in the same location where you edit the value. Click  next to the value text box, then select **Create variable**. In the Create New Data dialog box, set the name and location for the new variable, then click **Create**. The text box now displays the new variable.

For more information on specifying block parameter values by using numeric variables, see [Share and Reuse Block Parameter Values by Creating Variables](#).

Access model workspace and data dictionary from model canvas

Previously, a badge in the lower left corner of the model canvas indicated that a model was linked to a data dictionary. In R2019b, this badge also appears when there is data in the model workspace. Click the badge, then use the menu to navigate to these data sources.

For more information on using the badge, see [Model Editing Environment](#).

Model Data Editor enhancements

Several enhancements improve how you manage and edit data when using the Model Data Editor:

- Column-specific filtering makes it easier to display and edit elements of interest. To create a filter, point to a column header and click the filter icon.
- Hierarchical display allows you to view and edit complex objects such as bus element ports and lookup table objects.
- For elements on the **Signals** tab, name-value display of properties in the Property Inspector provides the same editing environment as other tabs.

For more information on using the Model Data Editor, see [Configure Data Properties by Using the Model Data Editor](#).

Breakpoint set type must match specification for `Simulink.LookupTable` objects

In R2019b, when you use the command line to set the `Breakpoints` property of a `Simulink.LookupTable` object to a breakpoint set that does not match the type specified by the `BreakpointsSpecification` property, Simulink reports an error.

Version History

Previously, when you set the `Breakpoints` property of a `Simulink.LookupTable` object from the command line, the software updated the `BreakpointsSpecification` to match the type of the breakpoint set. In R2019b, you must specify the `BreakpointsSpecification` before setting `Breakpoints`.

Convert variables to Simulink.Parameter objects from the context menu

In R2019b, you can more easily convert a variable to a `Simulink.Parameter` object by using a context menu in both the Model Explorer and the Model Data Editor:

- In the Model Explorer, in the **Contents** pane, right-click a variable and from the context menu, select **Convert to parameter object**.
- In the Model Data Editor, right-click the row that contains the variable and from the context menu, select **Convert to parameter object**.

For more on using data objects to specify values for parameters, see Data Objects.

Functions `Simulink.findIntEnumType` and `Simulink.clearIntEnumType` Applied to User-Defined Enumeration Classes

Previously, the functions `Simulink.findIntEnumType` and `Simulink.clearIntEnumType` applied to all defined enumeration classes, including those defined by data dictionaries and by Stateflow. In R2019b, these functions apply only to enumeration classes defined at the command line by using `Simulink.defineIntEnumType`.

For more information, see Use Enumerated Data in Simulink Models.

Block Enhancements

Blockset Designer: Streamline the process of blockset authoring and file management

In a project that contains many Simulink blocks, file management can be very time consuming. Blockset Designer is a Projects-based tool that streamlines the authoring and file management of custom Simulink blocks. You can use Blockset Designer to:

- Create a block, add a test, and create documentation.
- Organize blocks into manageable sublibraries.
- Manage the block artifacts.
- Search and sort added blocks according to their names and statuses.
- Publish your project as a MATLAB toolbox and install it to the Simulink Library Browser.

See [Create, Organize, and Publish Custom Blocksets](#) for more information.

New Messages & Events library

Use the blocks from the new Messages & Events library to send, receive, and queue messages.

This table lists the blocks in the library and their main uses.

Block	Action
Send	Convert data signal and send message.
Receive	Receive message and convert it to data signal.
Queue	Queue messages.
Hit Crossing Probe	Detect a crossing point and output a message.
Sequence Viewer	Visualize Simulink function connectors and message flow.

Send, Receive, and Queue blocks can also be found in the SimEvents library. The Sequence Viewer block is also located in the SimEvents, Stateflow, and Simulink Test libraries.

For an example usage of blocks provided in the Messages & Events library, see [Message-Based Communication Between Software Components](#).

Sequence Viewer Block: Visualize function calls, message communication, and Stateflow state changes and event activity over time

Use the Sequence Viewer block to visualize Simulink function calls, message transitions between blocks, and interchange of messages and events between Stateflow charts. For more information about the block, see [Use the Sequence Viewer Block to Visualize Messages, Events, and Entities](#).

n-D Lookup Table, 1-D Lookup Table, and 2-D Lookup Table blocks support modified Akima cubic Hermite interpolation

The n-D Lookup Table, 1-D Lookup Table, and 2-D Lookup Table blocks support a modified Akima cubic Hermite interpolation method (**Interpolation method > Akima Spline**). The modified Akima cubic Hermite interpolation method has these properties:

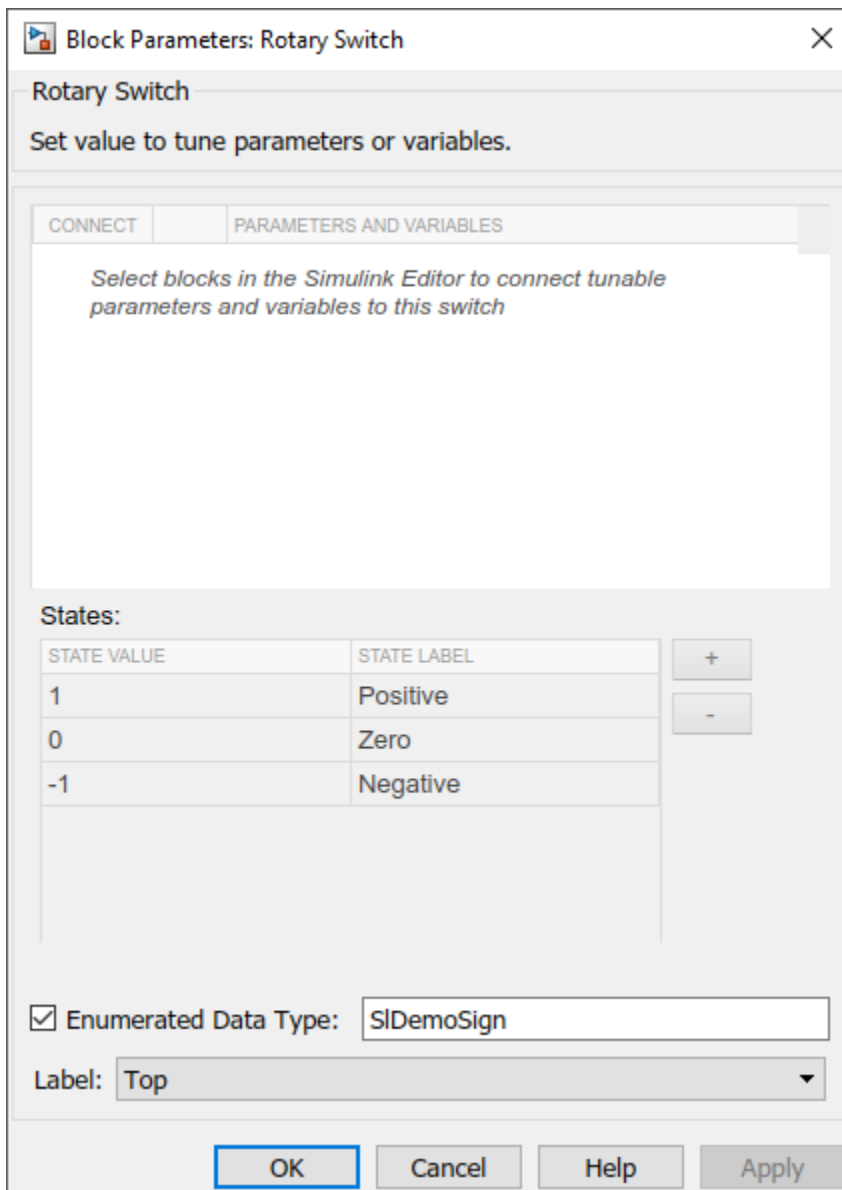
- It produces fewer undulations than Cubic spline.
- It is more efficient for real-time applications than Cubic spline.
- Unlike Cubic spline, it does not produce overshoots.
- Unlike Cubic spline, it supports nonscalar signals.

When saved to a previous release, the modified Akima cubic Hermite interpolation setting is saved as Cubic spline.

Enhanced enumerated data type support for the Rotary Switch block and Combo Box block

You can use an enumeration class to configure the values and labels for the states of a Rotary Switch block and a Combo Box block. Similar support was added for the Radio Button block in R2019a.

To configure the **States** for a Rotary Switch or Combo Box block with an enumerated data type, select **Enumerated Data Type**. Enter the name of the enumeration class you want to use in the text box. In this example, the enumeration class `SLDemoSign` defines the state values and labels for a Rotary Switch block.



When you use an enumeration class to configure the **States** for the block, you cannot manually edit, add, or remove states from the **States** table.

Configure Display block appearance

Intuitively communicate the significance of the connected signal by configuring the **Background Color**, **Foreground Color**, and **Opacity** for Display blocks in your model.

Math Function block pow function changes

The Math Function block pow has a new check box, **Signed power**. By default, the pow method calculates the power of the absolute value of the input, multiplied by the sign of the input. To calculate the power of the actual input value, clear this check box.

Additional data type and parameter support with mask parameter constraints

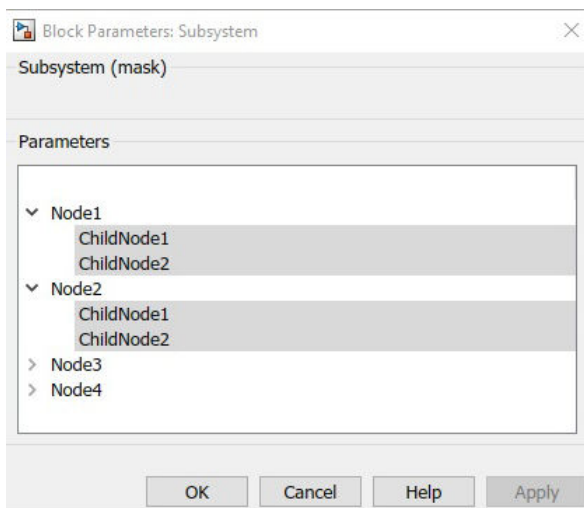
- Parameter support: In addition to associating mask parameter constraints to the **Edit** type mask parameter, you can also associate it to the **Combo box** mask parameter.
- Data type support: You can now use `int64` and `uint64` data types with mask parameter constraints. For more information on parameter constraints, see [Validating Mask Parameters Using Constraints](#).

Evaluate Combo box mask parameter during simulation

You can now choose to evaluate the **Combo box** mask parameter during simulation. To do so, select the **Evaluate** option available in the **Property editor** section of the Mask Editor. For more information on Combo box parameter, see [Controls](#).

Create a hierarchical list of data on mask dialog box

From R2019b, you can use the **Tree control** available in the Mask Editor to create a hierarchical list of data on the mask dialog box. You can select multiple values from this hierarchical list of data.



For more information on Tree Control, see [Create Hierarchical List in Mask Dialog](#).

Store mask images in the SLX file

From R2019b, you can also store the mask icon images inside the SLX file.

Before R2019b, you could link to the mask icon image from a file outside the SLX file. For more information on adding images to mask, see [Add Images in Masks](#).

New interface for managing signal viewers and generators

The **Signal and Scope Manager** has been replaced by the **Viewers and Generators Manager**. To open the **Viewers and Generators Manager**, on the Simulation tab, open the Prepare gallery and


select **Viewers Manager**. You can also right-click a signal and select **Viewers and Generators Manager**.

The Viewers and Generators Manager panel opens on the right side of the Simulink canvas. In this panel you can:

- Add and delete viewers and generators
- Connect signals to a viewer or generator
- See which signals are connected to a viewer or generator

For more information, see [Add Signals to an Existing Floating Scope or Scope Viewer](#), [Attach Signal Generator](#), or [Viewers and Generators Manager](#).

Playback control behavior changed for scopes in referenced models

When you use a scope block in a referenced model, the playback controls in the scope now match the playback controls of the last model you interacted with that contains the scope. For example, if you opened your scope from a model referenced by another model with the Model block, the run button  in the scope runs the top-level model. If the referenced model is opened as a top model, the run button runs the referenced model in isolation.

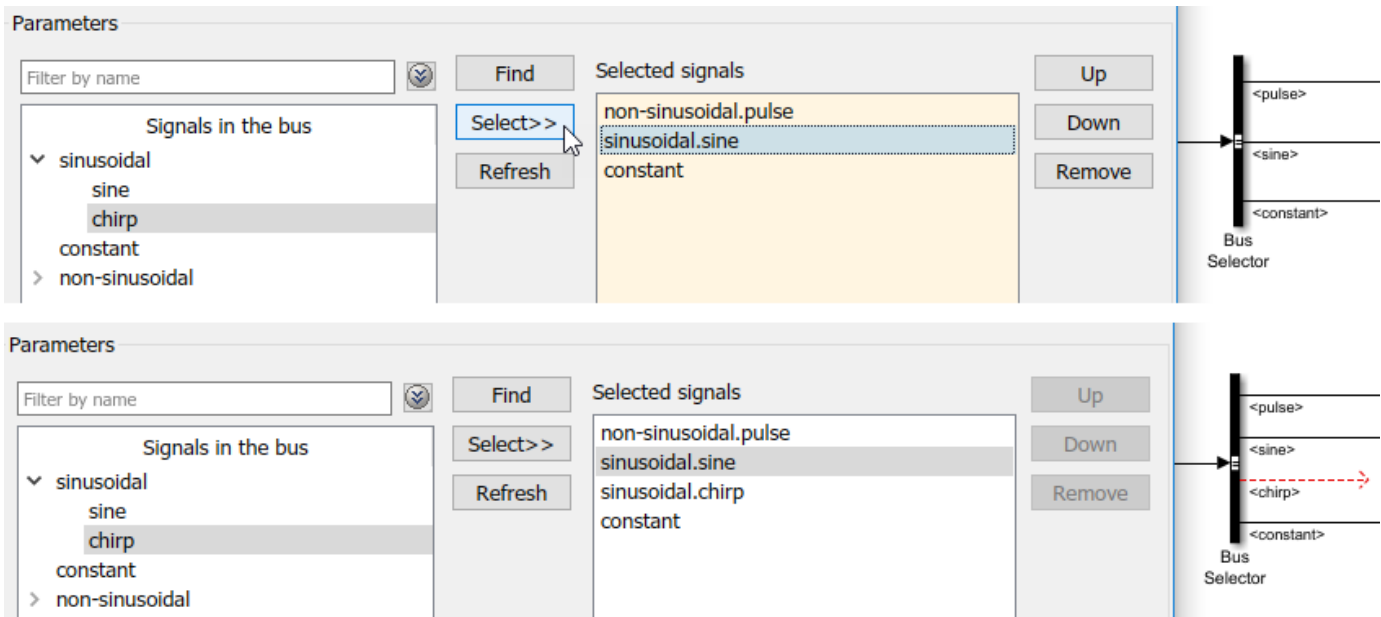
This change affects the Floating Scope, Scope Viewers, Scope, and some toolbox-specific scopes.

For more information, see [Scopes in Referenced Models](#).

Bus Selector and Bus Assignment block usability improvements

In R2019b, you can move multiple contiguous signals at the same time in the **Selected signals** list of the Bus Selector block and in the **Signals that are being assigned** list of the Bus Assignment block using the **Up** and **Down** buttons. You can also now specify where to insert selected signals in these lists.

If the block has selected signals, in the block dialog box list, click the signal that you want to precede additional signals before clicking the **Select** button. For example, by clicking the `sinusoidal.sine` signal in the **Selected signals** list before clicking the **Select** button, you can insert the `sinusoidal.chirp` signal directly below the `sinusoidal.sine` signal.



If you do not click a signal in the list before selecting additional signals, Simulink inserts the signals at the end of the list.

Control inherited sample time for System objects included in a MATLAB System block

If you want to control the sample time for your MATLAB System block, you can use the `createSampleTime` and `getSampleTimeImpl` methods in your System object.

Starting in R2019b, if your block specifies 'Inherited' sample time, you can now specify which inherited sample times you want to allow, error on, or change. See the `AllowPropagation`, `AlternatePropagation`, and `ErrorOnPropagation` properties of `createSampleTime`.

Delay block enhanced support for symbolic dimensions

Delay block has been enhanced to support symbolic dimensions for block input, block output, **Initial condition** parameter, and **State name** parameter.

String blocks updates

String blocks now support:

- External mode simulation
- Applications created by Simulink Real-Time™

For more information on Simulink strings, see Simulink Strings.

Configure C Caller block properties using command line API

The C Caller block has command line APIs to create an object and modify its properties. To create an object on a selected C Caller block:

```
myCCallerObj = get_param(gcf, 'FunctionPortSpecification');
```

This command returns a `FunctionPortSpecification` object with these properties:

- `CPrototype` displays the mapping of C function input for the C Caller block.
- `InputArguments` returns an array of the function input arguments.
- `ReturnArgument` returns a scalar of the output arguments. If the C function is a void function, this property is empty.

See `FunctionPortSpecification` for more information.

Select behavior for undefined function handling in C Caller block

Functions that are declared in the header file but not implemented in the source or library file are called undefined functions. In previous releases, such functions were stubbed automatically for the C Caller block by default. Starting in 2019b, you can select the desired undefined function behavior for your C Caller block. Models compiled in older releases continue to work, and a warning that empty stubs are created for your undefined functions displays. To change the default behavior, go to **Configuration Parameters > Simulation Target > Undefined function handling** and select the desired behavior. See `Undefined Function Handling` for more information.

New Entity Transport Delay Block

The Continuous library in Simulink has a new block, Entity Transport Delay. The block connects `SimEvents` to Simulink by allowing transport delays in a discrete event process to be modeled as a continuous-time process in Simulink.

Use the Entity Transport Delay block to apply a transport delay to entities. The block accepts entities at its first input port and the instantaneous delay for the entity at its second input port.

For example usage, see `Modeling Cyber-Physical Systems`.

For more information on transport delay, see `Variable Transport Delay`.

First Order Hold block

Use the First Order Hold block in the Simulink/Continuous block library for discrete-to-continuous signal conversion.

Note This block does not replace the First-Order Hold block in the Simulink/Discrete library. For more information, see “Functionality being removed or changed” on page 6-53

Simulink block data type support table

To display the Simulink block data type support table, at the MATLAB command line, type:

`showblockdatatypetable`

For more information, see `showblockdatatypetable`.

Alternatively, you can also double-click the Block Support Table block.

In previous releases, you accessed this table from the Simulink editor by selecting **Help > Simulink > Block Data Types & Code Generation Support > Simulink**.

Expanded block I/O buffer length

Prior to R2019b, the total block I/O buffer length in a model was $2^{31}-1$ bytes on 64-bit systems. In R2019b, the limit for the maximum number of bytes for the total block I/O buffer length in a model is $2^{64}-1$ bytes on 64-bit systems. For more information, see [Maximum Size Limits of Simulink Models](#).

MinMax Running Resettable update

The MinMax Running Resettable block has been reimplemented to correct numerical behavior during simulation with variable-step solvers.

Version History

Starting in R2019b, the code pattern for the generated code in MinMax Running Resettable differs from in previous releases, but the numerical behavior of the generated code remains the same. Numerical behavior of the simulation is now corrected for variable-step solvers.

Memory block update

The Memory block now allows its output at the initial integration step to be `inf` or `NaN`. In previous releases, these values were not allowed.

Priority set on If Action Subsystem block ignored

Simulink ignores a priority set on an If Action Subsystem block. Instead, set the priority on the If block that initiates execution of the subsystem.

Functionality being removed or changed

The Disallow property of the createSampleTime method will be removed.

Still runs

The `Disallow` property of the `createSampleTime` method will be removed in a future release. Use the `AlternatePropagation` property instead. `AlternatePropagation` provides the same functionality as `Disallow` and gives you the option to alter other sample time types besides controllable sample time.

The First-Order Hold block in the Discrete library is not recommended.

Still runs

The First-Order Hold block in Simulink/Discrete library is not recommended for continuous to discrete signal conversion. Use the Zero-Order Hold block instead.

Connection to Hardware

New blocks for communicating with Android devices

This table lists the new blocks available in the Simulink Support Package for Android Devices.

Block	Usage
Serial Transmit	Send buffered data to the serial port
Serial Receive	Receive data on the serial port
Audio File Write	Write mono or stereo audio data to a wave or m4a file

Support for MATLAB Arduino IO functions inside a MATLAB function block

The Simulink Support Package for Arduino Hardware now supports MATLAB IO functions for Arduino hardware inside a MATLAB function block in the Simulink model. You can run the model on an Arduino board that uses the following interfaces for hardware connections: ADC, PWM, and Digital Read/Write.

Support for Arduino MKR CAN Shield: Run Simulink models on Arduino boards

You can use the Simulink Support Package for Arduino Hardware to generate code for the Arduino MKR CAN Shield.

This table lists the new blocks available in Simulink Support Package for Arduino Hardware to support communication with Arduino MKR CAN Shield.

Block	Usage
CAN Transmit	Transmit CAN message to MCP2515-based CAN modules
CAN Receive	Receive CAN message from MCP2515-based CAN modules

Support for Arduino MKR WIFI 1010 Hardware: Run Simulink models on Arduino boards

You can use the Simulink Support Package for Arduino Hardware to generate code for the Arduino MKR WIFI 1010 board. The support package supports the wireless capabilities provided by the board. Using the WiFi blocks from the support package library, you can send and receive TCP/IP and UDP messages to and from the MKR WIFI 1010 board.

Support for Arduino MKR Zero: Run Simulink models on Arduino boards

You can use the Simulink Support Package for Arduino Hardware to generate code for the Arduino MKR Zero board.

Read data from files in SD Card

The Simulink Support Package for Arduino Hardware provides a new block, SD Card File Read, which reads data from a file stored in an SD card.

You must install the Simulink Support Package for Arduino Hardware to use this support. To install or update this support package, perform the steps described in [Install Support for Arduino Hardware](#) (Simulink Support Package for Arduino Hardware).

New blocks to communicate with CAN bus module connected to Raspberry Pi

This table lists the new blocks available in Simulink Support Package for Raspberry Pi Hardware that help to communicate with MCP2515-based CAN modules connected to the Raspberry Pi hardware.

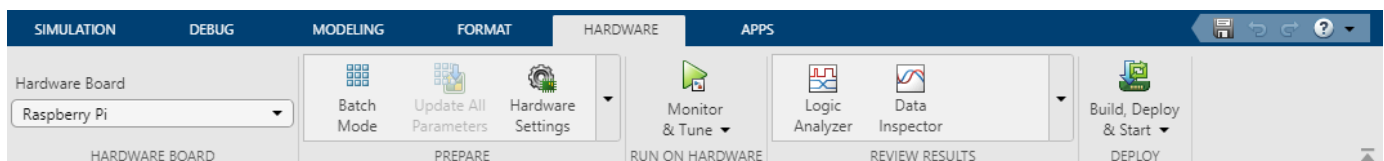
Block	Usage
CAN Transmit	Transmit CAN message to MCP2515-based CAN modules
CAN Receive	Receive CAN message from MCP2515-based CAN modules


Support for Grabber and Cannon modules of Parrot minidrone

The Simulink Support Package for Parrot Minidrones provides two new blocks - Grabber and Cannon. You can use these blocks in your Simulink model to control the Grabber module or the Cannon module connected to the Parrot Mambo minidrone.

Hardware Toolstrip: Build, deploy, and start Simulink models on supported hardware

The **Hardware** tab provides tools and features, previously available in menus and toolbars, to support user workflow tasks related to running Simulink models on hardware boards provided by product support packages.



To enable the **Hardware** tab, select **Apps > Run on Hardware Board**  from the gallery. In the **Run on Hardware Board** dialog box, select a hardware board, and click **Finish**.

Note The **Hardware** tab is part of the larger Simulink toolstrip update to Simulink and dependent products. For more information of the Simulink toolstrip update, see “Simulink Editor” on page 6-2.

Simulink Editor Changes

Mapping from Simulink Editor to the Hardware Toolstrip

The following table list the new Hardware Toolstrip items that are different from the Simulink Editor menu bar items. Many of the features and options that were previously hidden within the toolbar menus are now directly available from tabs on the Hardware Toolstrip.

Menu Bar Item	Toolstrip Equivalent
Model Configuration Parameters > Hardware Implementation > Hardware board	Hardware > Hardware Board > Hardware Board > Select Another Hardware Board
Configuration Parameters	Hardware > Prepare > Hardware Settings
Configuration Parameters > Data Import/Export > Signal Logging	Hardware > Prepare > Signal Monitoring and Tracing > Log Signals
Diagram > Signals & Ports > Viewers > Create & Connect Viewer	Hardware > Prepare > Signal Monitoring and Tracing > Add Viewer
Open a signal's Signal Properties dialog and check Test Point in the Logging and accessibility section.	Hardware > Prepare > Signal Monitoring and Tracing > Test Point
Code > External Mode Control Panel	Hardware > Prepare > Signal Monitoring and Tracing > Control Panel
View > Model Data Editor(Ctrl+Shift+E) > Signals	Hardware > Prepare > Signal Monitoring and Tracing > Signal Table
Configuration Parameters > Data Import/Export	Hardware > Prepare > Signal Monitoring and Tracing > Configure Logging
Open Code > External Mode Control Panel , enable Parameter tuning > Batch download .	Hardware > Prepare > Tune Parameters > Batch Mode
Open Code > External Mode Control Panel , click Parameter tuning > Download .	Hardware > Prepare > Tune Parameters > Update All Parameters
Run with model Simulation mode set to External .	Hardware > Run on Hardware > Monitor Signals & Tune Parameters > Monitor & Tune
Open Code > External Mode Control Panel , click Connecting and triggering > Connect .	Hardware > Run on Hardware > Step by Step Commands > Connect
Open Code > External Mode Control Panel , click Connecting and triggering > Start Real-Time Code .	Hardware > Run on Hardware > Step by Step Commands > Start
Simulation > Output >	Hardware > Review Results >
<ul style="list-style-type: none"> • Simulation Data Inspector • Logic Analyzer 	<ul style="list-style-type: none"> • Simulation Data Inspector • Logic Analyzer
View > Model Explorer > Base Workspace	Hardware > Review Results >MATLAB Workspace

Menu Bar Item	Toolstrip Equivalent
Deploy to Hardware button with Configuration Parameters > Hardware Implementation > Target hardware resources > Build options > Build action set to: <ul style="list-style-type: none">• Build• Build and load• Build, load, and run	Hardware > Deploy > <ul style="list-style-type: none">• Build Stand-Alone• Build & Deploy• Build, Deploy & Start

MATLAB Function Blocks

Simplify interfacing with row-major data by using row-major array layout

In R2019b, you can use row-major array layout in a MATLAB Function block to more easily and efficiently integrate your model with row-major data and algorithms. When you specify row-major array layout in the block, the layout applies to the block for simulation and C/C++ code generation. To specify the array layout for the block, use the `coder.rowMajor` and `coder.columnMajor` functions inside your MATLAB functions. For more information, see [Interface with Row-Major Data in MATLAB Function Block](#).

MATLAB Function blocks also support the array layout setting that you specify at the model level for C/C++ code generation. See [Array layout \(Simulink Coder\)](#).

Simulink string support in MATLAB Function block

In R2019b, Simulink strings are supported in MATLAB Function blocks.

Configure MATLAB Function block properties using command line APIs

The MATLAB Function block has command-line APIs to create an object and modify its properties. To create a configuration object on a MATLAB Function block, select the block and at the command line, type:

```
myMLFBConfigObj = get_param(gcf, 'MATLABFunctionConfiguration');
```

```
MATLABFunctionConfiguration with properties:
  Path: 'call_stats_block2/MATLAB Function'
  FunctionScript: ''
  UpdateMethod: Inherited
  SampleTime: '-1'
  Description: ''
  DocumentLink: ''
  SupportVariableSizing: 1
  AllowDirectFeedthrough: 1
  SaturateOnIntegerOverflow: 1
  TreatAsFi: FixedPoint
  FimathMode: SameAsMATLAB
  Fimath: ''
```

For more information, see [MATLABFunctionConfiguration](#).

Code generation for tables

In R2019b, you can generate code for tables. For more information, see [Code Generation for Tables](#).

Code generation for duration arrays

In R2019b, you can generate code for duration arrays. For more information, see [Code Generation for Duration Arrays](#).

Code generation for hexadecimal and binary literals

In R2019b, you can generate code for MATLAB hexadecimal and binary literals. The literals appear as integers in the generated code. In certain situations, the code generator might perform optimizations that remove these literals from the generated code.

For more information, see [Hexadecimal and Binary Values \(MATLAB\)](#).

Code generation for more MATLAB functions

- `array2table`
- `cell2table`
- `datevec`
- `days`
- `duration`
- `fgetl`
- `fgets`
- `fillmissing`
- `fileread`
- `height`
- `histcounts2`
- `hms`
- `hours`
- `islocalmax`
- `islocalmin`
- `ismissing`
- `matchpairs`
- `milliseconds`
- `minutes`
- `newline`
- `read`
- `rmissing`
- `rmoutliers`
- `seconds`
- `standardizeMissing`
- `struct2table`
- `table`
- `table2array`
- `table2cell`
- `table2struct`
- `tcpclient`

- width
- write
- years

Code generation for more toolbox functions

5G Toolbox

- nrCarrierConfig
- nrChannelEstimate
- nrCSIRS
- nrCSIRSConfig
- nrCSIRSIndices
- nrTimingEstimate

Audio Toolbox

- audioPluginGridLayout
- audioTimeScaler
- designAuditoryFilterBank
- pinknoise
- shiftPitch
- stretchAudio

Automated Driving Toolbox

- acfObjectDetectorMonoCamera
- birdsEyeView
- imageToVehicle of birdsEyeView
- segmentLaneMarkerRidge
- transformImage of birdsEyeView
- vehicleToImage of birdsEyeView

Communications Toolbox

- bleATTPDU
- bleATTPDUConfig
- bleATTPDUDecode
- bleChannelSelection
- bleGAPDataBlock
- bleGAPDataBlockConfig
- bleGAPDataBlockDecode
- bleIdealReceiver
- bleL2CAPFrame

- bleL2CAPFrameConfig
- bleL2CAPFrameDecode
- bleLLAdvertisingChannelPDU
- bleLLAdvertisingChannelPDUConfig
- bleLLAdvertisingChannelPDUDecode
- bleLLControlPDUConfig
- bleLLDataChannelPDU
- bleLLDataChannelPDUConfig
- bleLLDataChannelPDUDecode
- bleWaveformGenerator
- gsmCheckTimeMask
- gsmDownlinkConfig
- gsmFrame
- gsmInfo
- gsmUplinkConfig

Computer Vision Toolbox

- See “Code Generation: Generate C/C++ code using MATLAB Coder” (Computer Vision Toolbox).

Image Processing Toolbox

- imregcorr

Navigation Toolbox

- cart2frenet
- frenet2cart
- nav.StateSpace
- nav.StateValidator
- plan
- stateSpaceDubins
- stateSpaceReedsShepp
- stateSpaceSE2
- trajectoryOptimalFrenet
- validatorOccupancyMap

Optimization Toolbox

- See fmincon Code Generation: Generate C code for nonlinear constrained optimization (requires MATLAB Coder) (Optimization Toolbox).

Phased Array System Toolbox

- backscatterBicyclist

- `clusterDBSCAN`
- `discoverClusters`
- `estimateEpsilon`
- `getNumScatterers`
- `move` of `backscatterBicyclist`
- `ompdecomp`
- `omphybweights`
- `reflect` of `backscatterBicyclist`

Robotics System Toolbox

- `ackermannKinematics`
- `bicycleKinematics`
- `differentialDriveKinematics`
- `jointSpaceMotionModel`
- `taskSpaceMotionModel`
- `unicycleKinematics`

Sensor Fusion and Tracking Toolbox

- `complementaryFilter`
- `ctrect`
- `ctrectcorners`
- `ctrectjac`
- `ctrectmeas`
- `ctrectmeasjac`
- `fuserSourceConfiguration`
- `gmphd`
- `initcagmphd`
- `initctgmphd`
- `initctrectgmphd`
- `initcvgmphd`
- `objectTrack`
- `toStruct`
- `trackFuser`
- `trackOSPAMetric`

Signal Processing Toolbox

These Signal Processing Toolbox™ functions now support C/C++ code generation:

- **Time-Frequency Analysis** — `fsst`, `ifsst`, `tfridge`, `wvd`, and `xwvd`
- **Spectral Analysis of Nonuniformly Sampled Signals** — `plomb`

- **Transforms** — dftmtx and rceps
- **Digital Filtering** — eqtflength, fftfilt, and tf2ss
- **Waveform Generation** — chirp, diric, gmonopuls, and sawtooth
- **Spectral Windows** — chebwin

Statistics and Machine Learning Toolbox

- See Generate C/C++ code for probability distribution functions (requires MATLAB Coder) (Statistics and Machine Learning Toolbox)
- See “saveLearnerForCoder and loadLearnerForCoder Functions: Save and load machine learning models for code generation” (Statistics and Machine Learning Toolbox)

Wavelet Toolbox

- filterbank of shearletSystem
- framebounds of shearletSystem
- isheart2
- mdwtdec
- mdwtrec
- meyeriaux
- numshears
- shearletSystem
- sheart2

WLAN Toolbox

- wlanAPEPLength
- wlanHESIGBCommonBitRecover
- wlanHESIGBUserBitRecover
- wlanPSDULength

Modeling Guidelines

High-Integrity System Modeling guidelines to verify compliance with DO, IEC, EN, and ISO safety standards

This table identifies new and modified modeling guidelines. For a complete list of High-Integrity System Modeling guidelines, including their applicable Model Advisor checks, see Model Advisor Checks for High-Integrity Modeling Guidelines.

High-Integrity Modeling Guideline	Rationale
himl_0011: Data type and size of condition expressions	New guideline that provides recommendations about the use of logical scalars for data type and size in condition expressions.
hisl_0047: Configuration Parameters > Code Generation > Code Style	Removed condition "Select Preserve condition expression in if statement. "
hisl_0049: Configuration Parameters > Code Generation > Identifiers	In R2019b, the configuration parameters pane Code Generation > Symbols is renamed as Code Generation > Identifiers . The title of and support documentation for the guideline are updated to reflect this change.
hisl_0303: Configuration Parameters > Diagnostics > Data Validity > Merge blocks hisl_0304: Configuration Parameters > Diagnostics > Data Validity > Model initialization hisl_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging	<p>These configuration parameters are now available as advanced parameters in the Diagnostics > Data Validitypane:</p> <ul style="list-style-type: none"> • Detect multiple driving blocks executing at the same time step • Underspecified initialization detection • Model Verification block enabling <p>The title of and support documentation for these guidelines are updated to reflect this change.</p>

R2019a

Version: 9.3

New Features

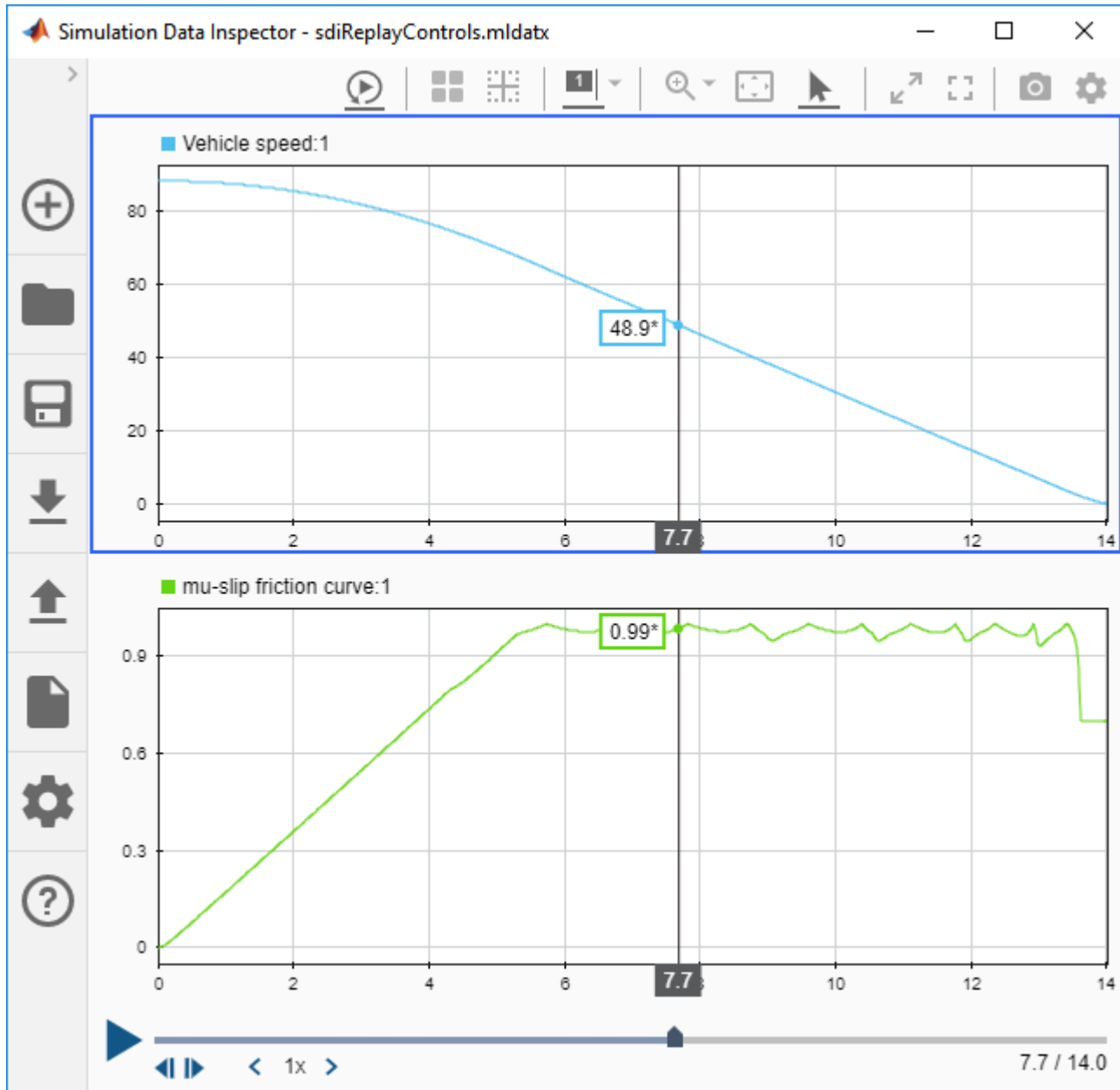
Bug Fixes

Version History

Simulation Analysis and Performance

Replay simulations in the Simulation Data Inspector

You can replay test and simulation data across multiple subplots with synchronized cursors in the Simulation Data Inspector. To enable the replay controls in your view, click the **Show/hide replay controls** button.



You can control the replay speed using the arrows on either side of the speed indicator. You can also click the indicator to enter a specific value. When the replay reaches a region of interest, you can pause it to closely inspect the data. You can move the cursor forward and backward, sample by

sample, using the arrow buttons. When you move the cursors sample by sample, the cursor moves to the next sample in any plotted signal, regardless of whether that signal is on the active subplot.

Signal Editor user interface supports signal draw and insert

The Signal Editor user interface lets you use a mouse or touchscreen to create freehand signal data. For more information, see [Create Freehand Signal Data Using Mouse or Touchscreen](#).

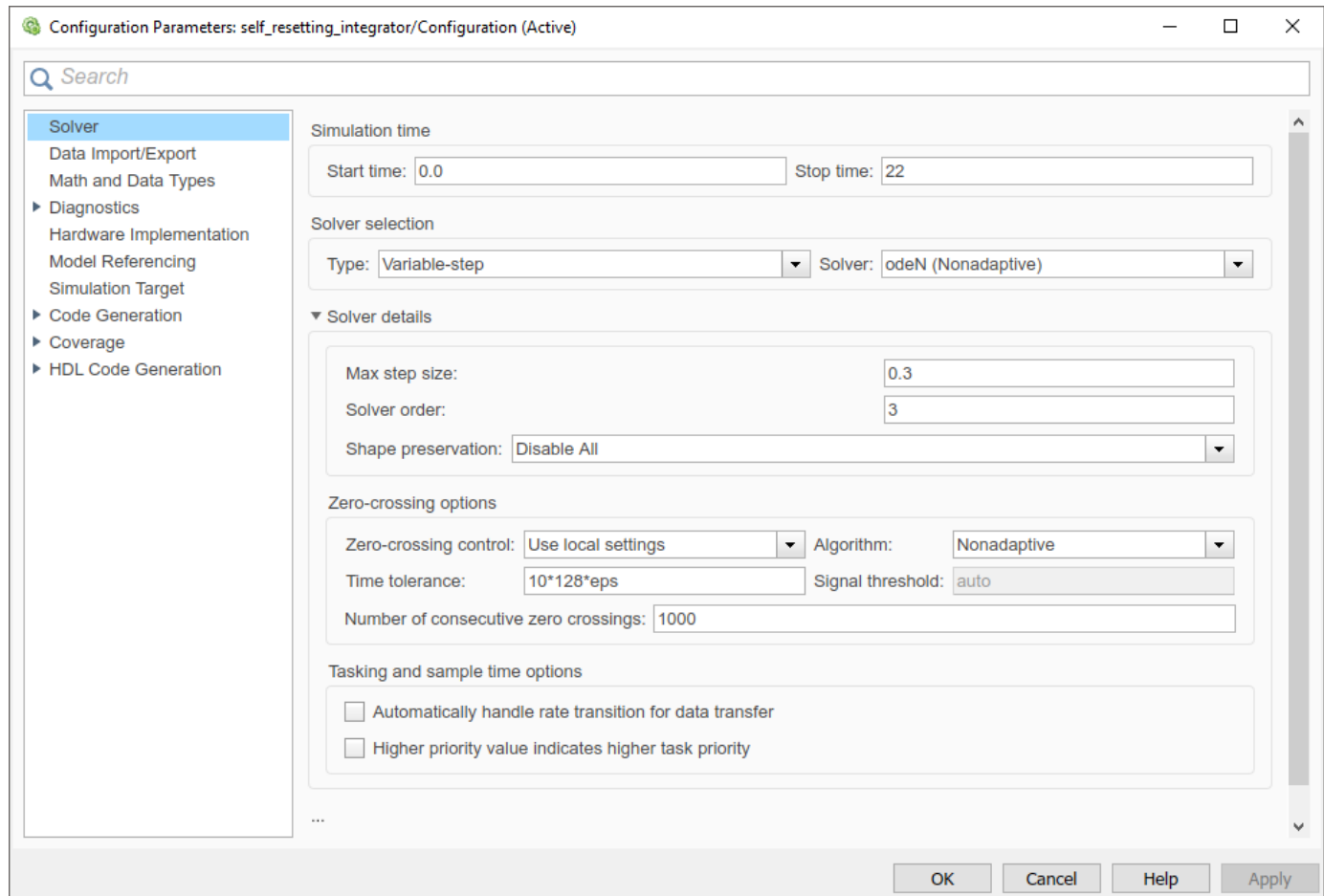
Timing Legend: Interactively identify different rates and their types through highlighting

Timing Legend, previously known as Sample Time Legend enables highlighting the origin and propagation of the sample time rate. In the updated Timing Legend, you can choose to highlight the blocks that introduce the rate or the blocks that contain the rate. Timing Legend also enables you to highlight all the blocks containing one type of rate. For example, select **Discrete** in the Timing Legend to highlight all the blocks in the model that contain discrete rates.

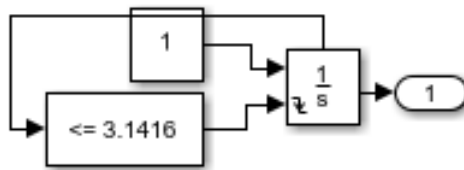
odeN Solver (Nonadaptive): Quickly simulate systems with zero-crossing detection

Simulink provides a variable-step solver that allows you to simulate dynamic models using variable-step (without error-control) integration while preserving accuracy at zero-crossings.

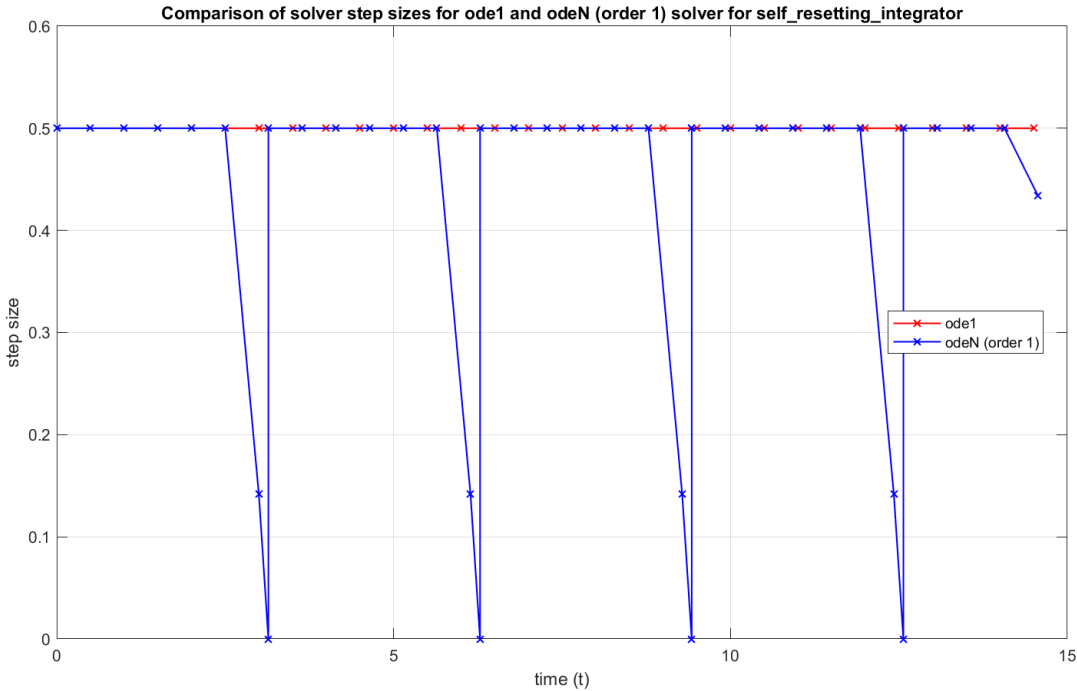
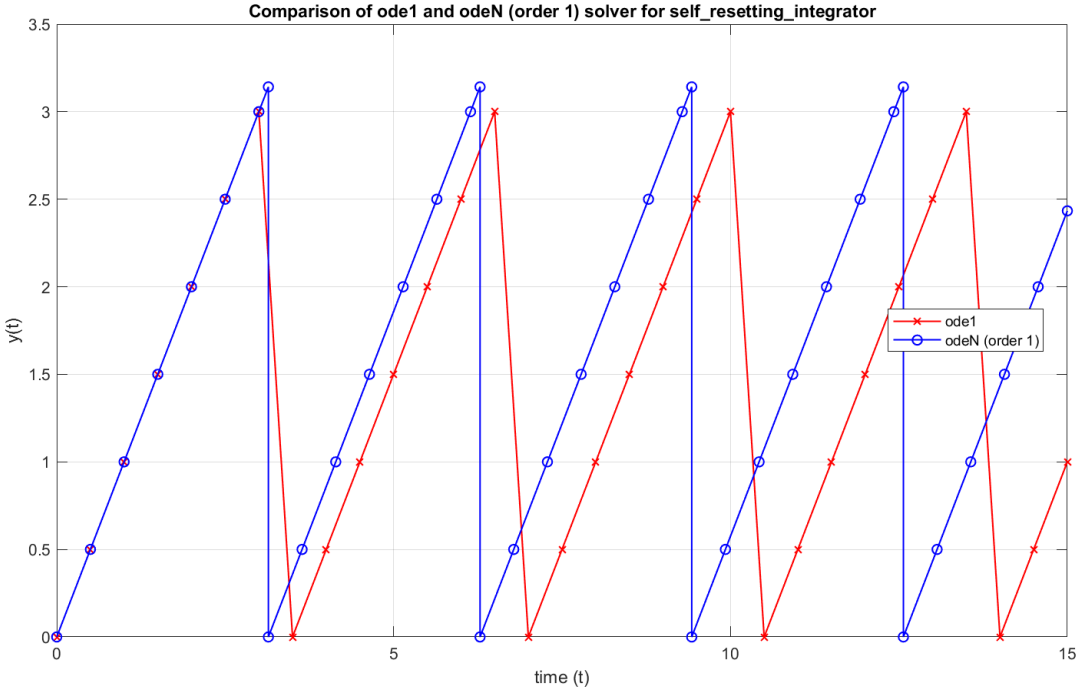
Select the desired odeN (Nonadaptive) solver from the **Solver** menu in **Configuration Parameters > Solver Pane**. You can specify the **Solver order** in the **Solver details** section of the **Solver pane** as well as the **Max step size** and **Shape preservation**.



This figure shows the comparison of the fixed-step ode1 solver and the variable-step odeN (Nonadaptive) solver with **Solver order** 1 after simulating a model of a self resetting integrator.



The integrator is configured to reset when its state $x=\pi$. The next figure shows a stem plot of the same simulation to better visualize the smaller step sizes taken to capture the zero-crossings.



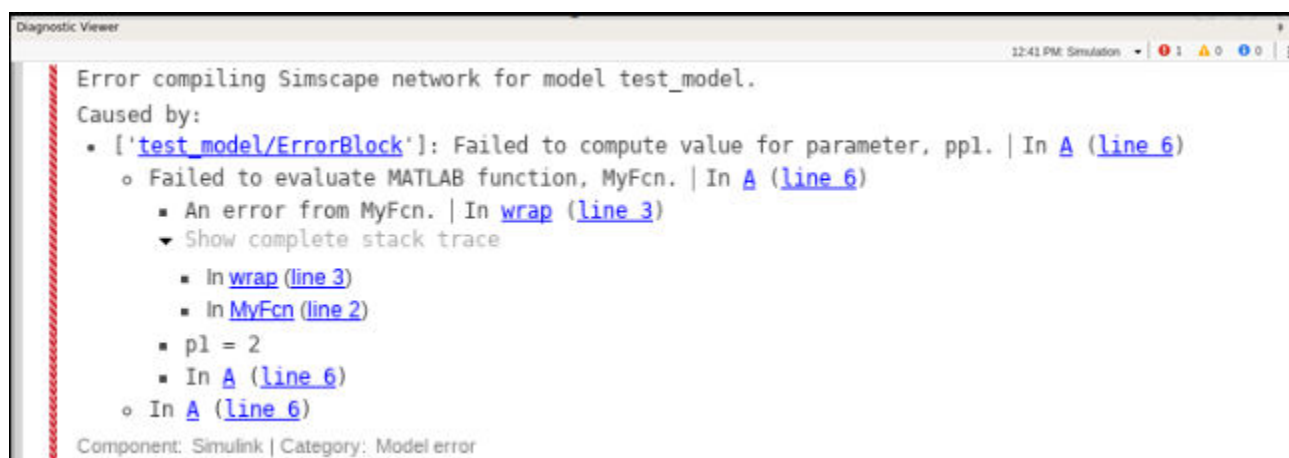
FMU Export: Package a model as an FMU for co-simulation

Packaging a model as a functional mockup unit (FMU) for co-simulation no longer requires a support package.

Wrap your Simulink model and project files into an FMU that has a co-simulation interface. When a third-party tool runs the FMU, the FMU checks out the required MathWorks licenses, starts a local MATLAB session, and runs the model.

Locate source of Simulink diagnostics

From R2019a, the Diagnostic Viewer reports the errors and warnings as a stack that is originating from a .m or .ssc file. Stack reporting enables easier viewing and navigation of diagnostics.



The screenshot shows the Diagnostic Viewer window with the following content:

```















Error compiling Simscape network for model test_model.
Caused by:
  * ['test_model/ErrorBlock']: Failed to compute value for parameter, p1. | In A (line 6)
    o Failed to evaluate MATLAB function, MyFcn. | In A (line 6)
      * An error from MyFcn. | In wrap (line 3)
        v Show complete stack trace
          * In wrap (line 3)
          * In MyFcn (line 2)
          * p1 = 2
          * In A (line 6)
        o In A (line 6)
Component: Simulink | Category: Model error
  
```

For more information, see View Diagnostics.

Simulation Data Inspector comparison enhancements

The comparison pane in the Simulation Data Inspector provides more detailed information about run comparison results and offers enhanced support for comparing large sets of data. A **Results** column summarizes the results of the comparison on the run row and on any data, model, or physical system hierarchy rows.

During comparisons, a progress indicator shows which signals are in the queue for comparison. The **Compare** button becomes a **Cancel** button while the Simulation Data Inspector evaluates a comparison, allowing you to cancel a comparison. When you cancel a comparison, the **Results** column indicates signals that were not compared due to the cancellation and displays results for comparisons completed before the cancellation.

 37 Within tolerance 82 Out of tolerance				
Filter Comparisons				
NAME (BASE)	ABS TOL	REL TOL	MAX DIFF	RESULT
▼ Compare Run 2: ssc_hydraulic_actuator_analog...				✓ 38 ✗ 83
▶  Analog P+I Actuator Controller				✓ 23 ✗ 55
▶  Custom Hydraulic Fluid				✓ 1
▶  HydrRef				✓ 1
▶  Hydraulic Actuator				✓ 7 ✗ 23
▶  Hydraulic Pressure Source				✓ 4 ✗ 1
▶  Load Damper				✓ 1 ✗ 3
<input checked="" type="radio"/> ▶  Load Hard Stop				✓ 1 ✗ 1
<input type="radio"/> ▶  Load Spring				
<input type="radio"/> ▶  MTRef_Cyl				
<input type="radio"/> ▶  MTRef Load				
<input type="radio"/> ▶  Mass				
<input type="radio"/> ▶  Position Sensor				
<input type="radio"/> ▶  Spool Valve				

Show or hide plot grid in the Simulation Data Inspector

You can choose whether to show or hide grid lines on plots in the Simulation Data Inspector. By default, the plots show the grid. To hide the grid, clear the **Show grid** option in the **Time Plot** preferences in the Simulation Data Inspector. The **Show grid** preference applies to all plots in the Simulation Data Inspector.

To modify the **Show grid** preference from the command window, use the `Simulink.sdi.setGridOn` function. You can check the state of the **Show grid** setting by using the `Simulink.sdi.getGridOn` function.

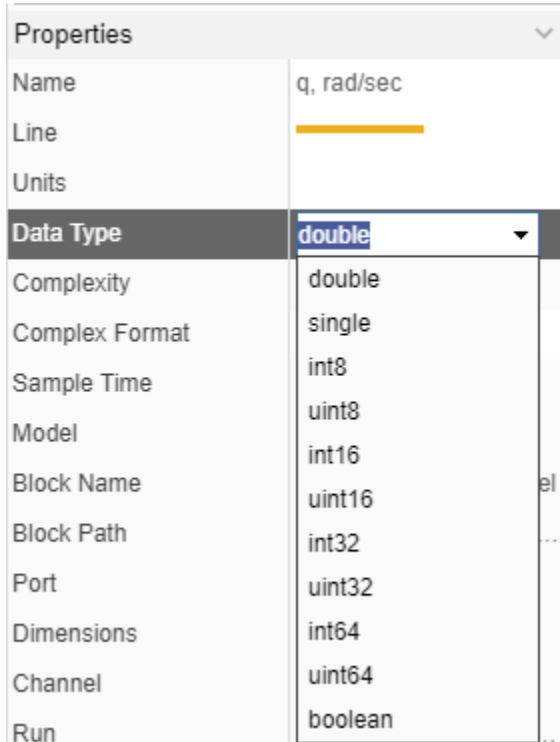
Export data to Microsoft Excel from the Simulation Data Inspector

You can export run and signal data from the Simulation Data Inspector to a Microsoft Excel workbook. In the **Export** dialog box, when you select **Export to: File**, you can select **Excel Workbook** from a drop-down menu of supported file types. When you export to Microsoft Excel, you can specify the file overwrite behavior, metadata to export, and whether to use a single, shared time column for each sheet.

Convert signal data types in the Simulation Data Inspector

You can convert the data type of signals in the Simulation Data Inspector. For data logged in a model, data type conversion in the Simulation Data Inspector does not affect the model. The data type conversion only affects the signal in the Simulation Data Inspector.

To convert the data type for a signal, select the signal and then click the field next to the **Data Type** property in the **Properties** pane. Select the desired data type from the drop-down.



Data type conversion in the Simulation Data Inspector supports all built-in data types. A Fixed-Point Designer license allows you to convert to fixed-point data types.

Process 1-D and 2-D matrix data during simulation

You can use the **Data Access** pane of the **Signal Properties** dialog box to specify a callback function to process logged 1-D and 2-D signal data. Previously, the **Data Access** callback only supported scalar signals. For more information about using data access callbacks, see [Access Data in a MATLAB Function During Simulation](#).

Bus signal logging enhancements

Signal logging fully supports bus signals. You can now log string data inside nonvirtual bus signals, and you can log data for virtual buses that contain one or more nonvirtual buses. When you save **Output** data, root-level Out Bus Element blocks are logged along with root-level Outport blocks.

FMU Import Block: Rapid accelerator support

FMU Import block supports rapid accelerator mode in both co-simulation and model exchange modes.

Save and restore a simulation snapshot using the renamed ModelOperatingPoint object

A ModelOperatingPoint object saves a snapshot of a simulation that you can restore later, and continue simulation from the exact point at which you stopped simulation.

Version History

In previous releases, you saved a simulation snapshot using a SimState object. The SimState object now is stored as a ModelOperatingPoint object. A ModelOperatingPoint object has a same behavior as a SimState object.

Return simulation outputs as a single simulation output by default for new Simulink models

For simulation of new Simulink models, the output is returned as a single output object, Simulink.SimulationOutput. The Simulink.SimulationOutput object encapsulates all the resulting enabled simulation data (signal, state, output, data store logging, toworkspace block logging, and scope). To follow the same behavior while simulating with the sim command, ReturnWorkspaceOutputs is set to On by default.

Version History

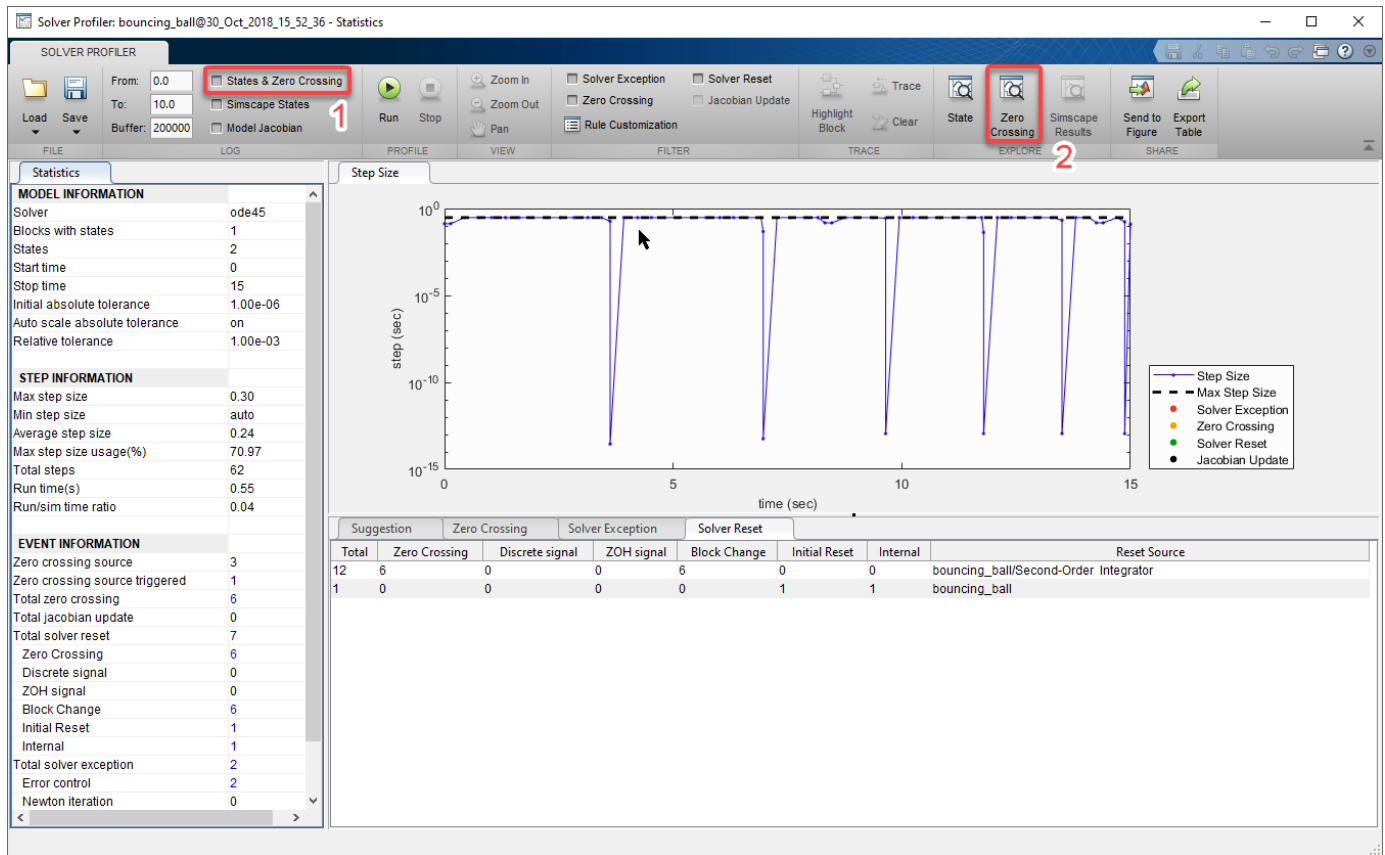
For Simulink models built in previous releases, ReturnWorkspaceOutputs argument remains off. Set the ReturnWorkspaceOutputs argument to on to return the simulation result as a single output object.

Zero-Crossing Explorer for Solver Profiler: Visualize and interactively explore zero-crossing events.

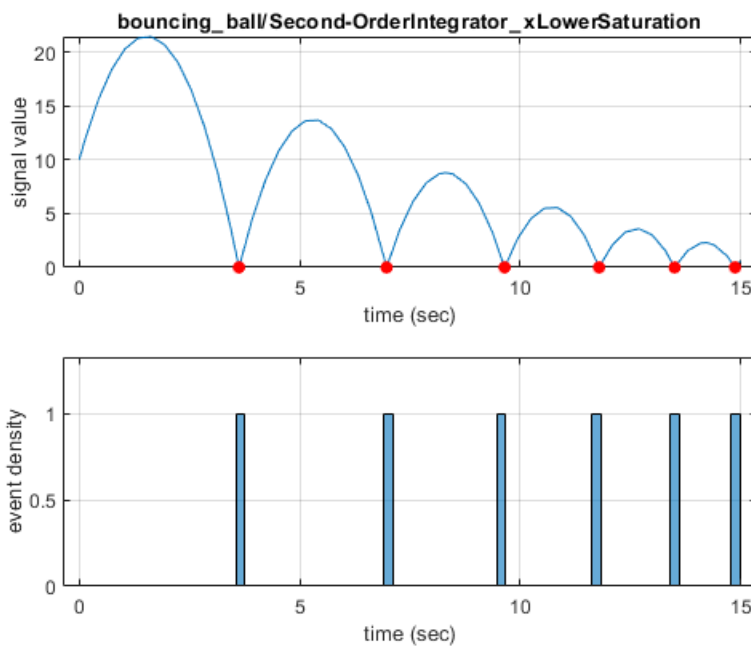
You can visualize zero-crossing signals with respect to a state's dynamics using the Solver Profiler's Zero Crossing Explorer.

Access the Zero Crossing Explorer as follows:

- 1 Profile the solver performance using the Solver Profiler. Ensure the **States & Zero crossing** parameter in the **Log** section of the toolstrip is enabled.
- 2 Once the Solver Profiler has completed profiling the model, select **Zero Crossing** from the **Explore** section of the Solver Profiler's toolstrip.



This figure shows the zero crossing results for a simulation of a bouncing ball.




Unit test constraint for comparing time series data

The `MatchesSignal` constraint compares an actual signal to a signal representing the expected value using the Simulation Data Inspector comparison algorithm. You can specify options for `MatchesSignal` such as interpolation method, synchronization method, alignment criteria, and tolerances, including time tolerance.

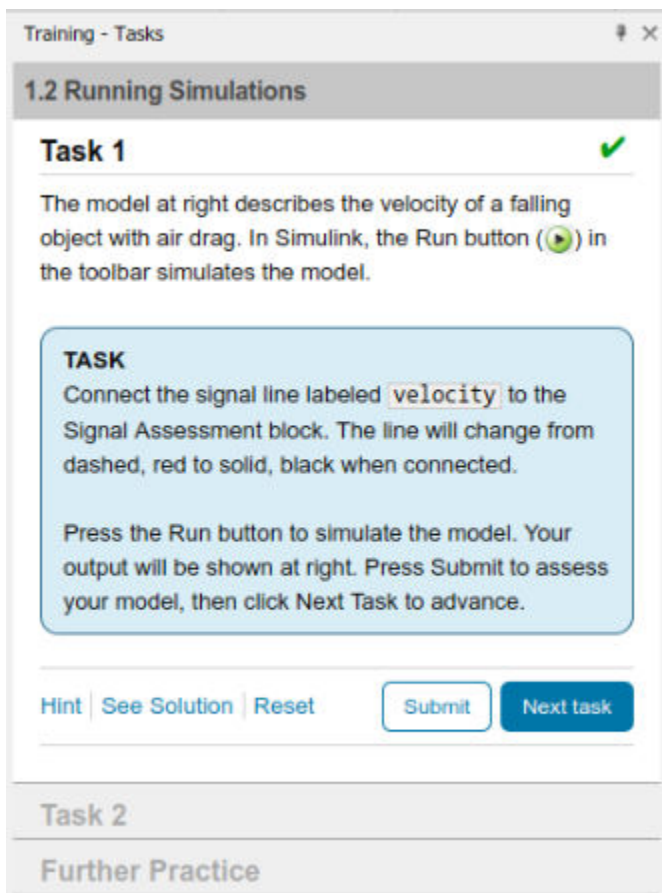
Simulink Editor

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

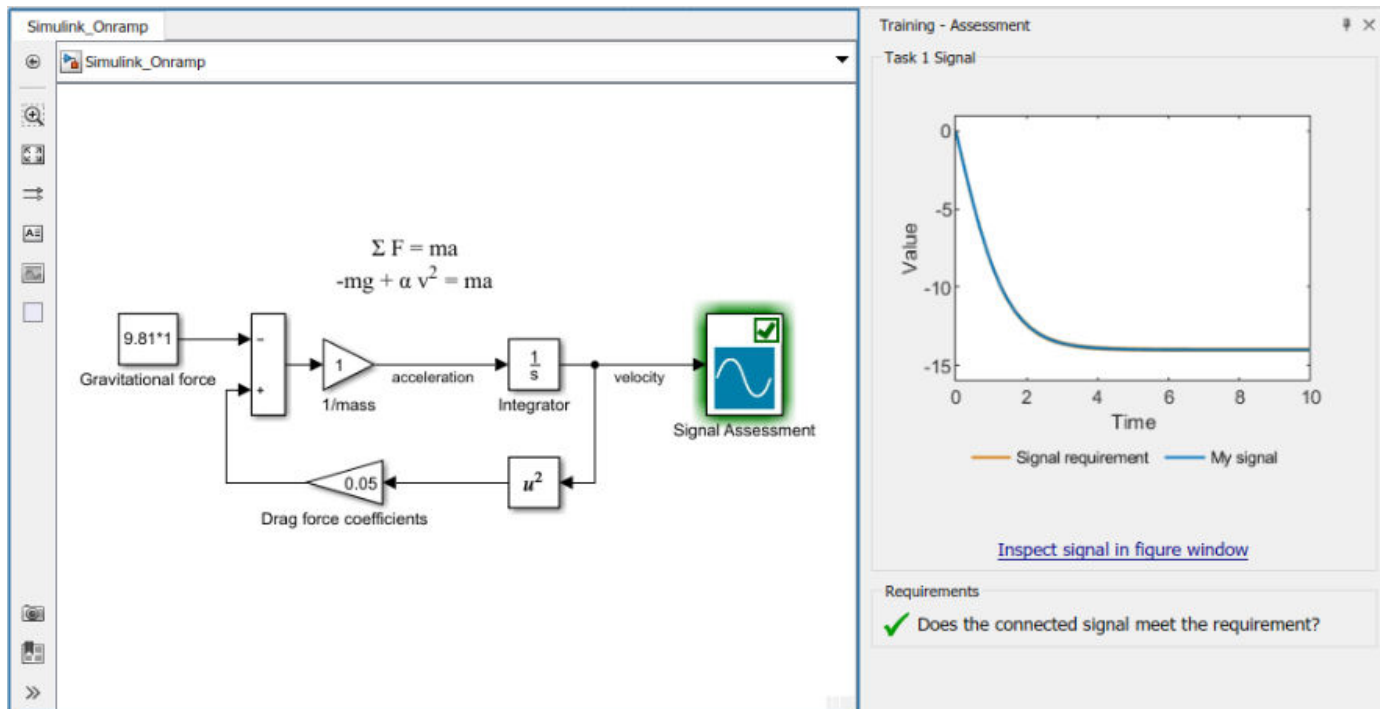
To help you get started quickly with Simulink basics, Simulink Onramp provides a self-paced, interactive tutorial. After completing Simulink Onramp, you will be able to create, modify, and troubleshoot Simulink models that simulate dynamic systems and perform basic signal analysis.

To open Simulink Onramp, on the Simulink Start Page, you can select the  **Simulink Onramp** button under **Learn**. Alternatively, you can select **Help > Simulink > Learn Simulink** or enter `learning.simulink.launchOnramp()` in the MATLAB command prompt.

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



You receive automated assessments and feedback after submitting tasks.



Your progress is saved if you exit the application, so you can complete the training in multiple sessions.

Simulink Onramp covers these topics:

- Simulink blocks and signals
- Math and logic operators for algorithms
- Simulink help and documentation
- MATLAB variables and functions in Simulink
- Dynamic systems in Simulink
- Discrete-time systems
- Continuous-time systems
- Simulation duration

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

- Automotive Performance Modes
- Modeling a Thermostat
- Peregrine Falcon Dive

Arrange Automatically: Improve layout and appearance using a menu command

You can use a menu command to attempt to improve your diagram layout and appearance. Select **Diagram > Arrange > Arrange Automatically**. This menu command can realign, resize, and move blocks and straighten signal lines. For more information, see Improve Model Layout.

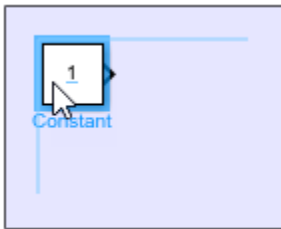
Smart Guides: Evenly space blocks and place blocks in suggested zones

In R2019a, these new smart guides help you position blocks.

- Even-spacing guides suggest where to place blocks so that the vertical or horizontal spacing between blocks is the same.

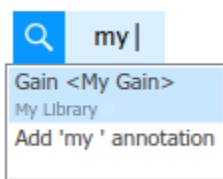


- Inner guides suggest where to place blocks inside Simulink areas and Stateflow boxes.



Quick Insert Keywords: Add library blocks to model using keyword

You can now add one or more keywords to library blocks. Doing so enables users to add the block from the quick insert menu using a keyword instead of the block name. The quick insert menu appears when you double-click the Simulink Editor canvas or double-click at the end of a partially drawn signal line. For more information on adding keywords, see [Blocks for Custom Libraries](#).



Collapsible Panes: Save and restore settings

Starting in R2019a, when you create a model, the Model Browser and Property Inspector are collapsed by default. Models now save which panes are persistent and collapsed. When you load a model, panes are restored according to their saved settings.

Annotations: Specify a transparent background and connect to areas

You can now make annotations transparent by selecting **Format > Background Color > Automatic** from the context menu. You can also connect annotations to areas as well as blocks.

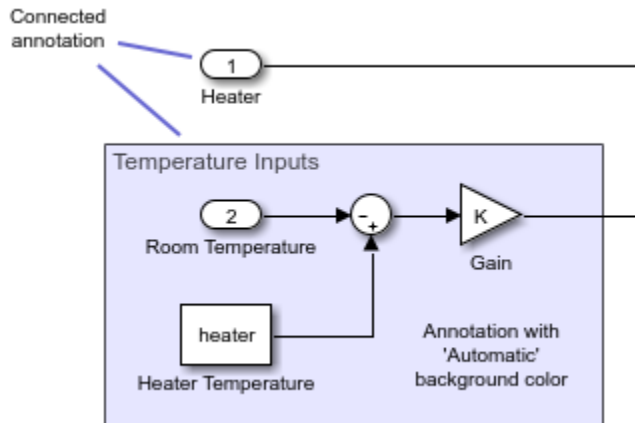


Image Annotations: Drag images to canvas

You can now drag an image from your local file system to the Simulink Editor canvas to create an image-only annotation. Previously, you pasted an image to an image annotation box or inserted it using a file browser. For more information, see [Add an Image Annotation](#).

Model Window Size Increased: 32-bit coordinates supported

Coordinates in the Simulink API, such as those specified by the `Position` parameter, now support a range from -1073740824 to 1073740823. Previously, the maximum absolute value for a coordinate was 32768.

No-Selection Action Bar: Draw to create area or subsystem

To draw a box that you can convert to an area or a subsystem, you can now use the Simulink Editor action bar without a selection. Without selection, start drawing a box, and then use the action bar to specify whether to make the box an area or a subsystem. For more information, see [Box and Label Areas of a Model](#) and [Create Subsystem from Selection](#).

`Simulink.BlockDiagram.expandSubsystem`: Expand subsystems without creating areas

You can now expand a subsystem programmatically without also creating an area annotation that surrounds the newly expanded blocks. Use the `Simulink.BlockDiagram.expandSubsystem` function with the `'CreateArea'` option specified as `'off'`.

```
Simulink.BlockDiagram.expandSubsystem(block, 'CreateArea', 'off');
```

`add_block`: Copy block or subsystem from library without library link

You can now use `add_block` to copy a library block or subsystem without a library link. Use the `'CopyOption'` with `'nolink'`. For more information, see [add_block](#).

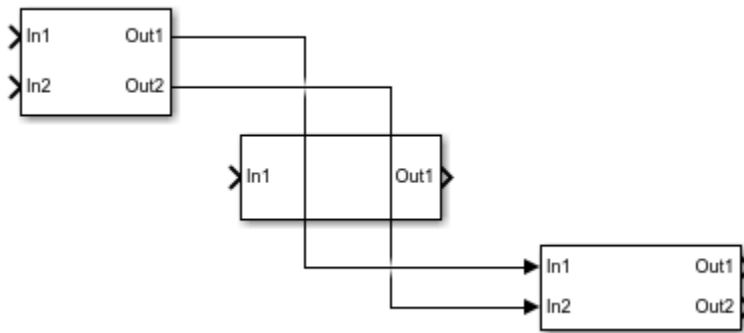
```
h = add_block(source,dest,'MakeNameUnique','on')
```

Simulink.BlockDiagram.routeLine: Route existing lines of a model

You can now use the `Simulink.BlockDiagram.routeLine` function to route the existing lines of a model. Routing existing lines improves line route quality and avoids overlaps of a line with other lines and obstacles in the model.

The `Simulink.BlockDiagram.routeLine` function takes an array of existing line handles as input and routes the corresponding lines. To find the line handles in the model that you want to route, use the `find_system` or `get_param` function.

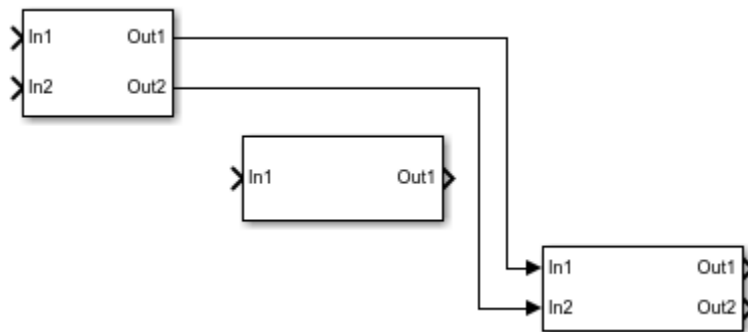
For example, consider this model.



To route the two lines in the top-level system, create an array of line handles by using `find_system`. Then call `Simulink.BlockDiagram.routeLine` with this array as input.

```
lineHandles = find_system(gcs, 'FindAll', 'On', 'SearchDepth', 1, 'Type', 'Line');
Simulink.BlockDiagram.routeLine(lineHandles);
```

The updated model looks like this.



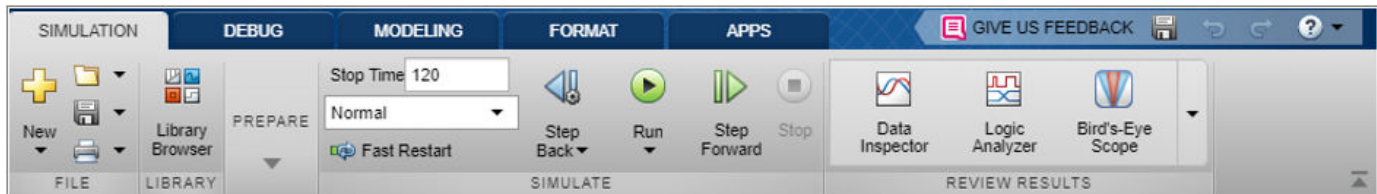
Functions added for displaying errors and warnings

In R2019a, there are two new functions for detecting model design errors and warnings. Use the new `edittime.setDisplayIssues` method to detect whether there are errors and warnings while editing a model. The `edittime.setDisplayIssues` method is equivalent to selecting **Display > Errors & Warnings**. Use the new `edittime.getDisplayIssues` method to determine if the model is set to display errors and warnings.

Simulink Toolstrip Tech Preview replaces menus and toolbars in the Simulink Desktop

In R2019a, you have the option to turn on the Simulink Toolstrip.

The Simulink Toolstrip contains components that were previously available in menus, toolbars, and the **Start** button. To support common user workflow tasks, new tabs called **Modeling**, **Simulation**, **Debug**, **Format**, and **Apps** provide functionality corresponding to each task.



The **Apps** tab provides a gallery of applications from the Simulink family of products. You can also install your own apps, which then appear in the **Apps** gallery.

The toolstrip tabs are contextual — they appear only when you need them. When you click a block in a model, for example, the **Subsystem** tab appears. If the model contains a chart, the **State** tab activates as well. When the block or chart is no longer selected, the contextual tabs disappear.

To activate the Simulink Toolstrip, open **Simulink Preferences**.

- 1 Save your open Simulink models.
- 2 Select **File > Simulink Preferences**.
- 3 Under **Editor Preferences**, select **Replace menus/toolbars with Simulink Toolstrip**. Click **Apply**.
- 4 Simulink closes all open models. When you reopen Simulink, the toolstrip replaces the menus and toolbars in any new or saved models.

To deactivate the Simulink Toolstrip, open **Simulink Preferences**.

- 1 Save your open Simulink models.
- 2 Under the **Modeling** tab, select **Environment > Simulink Preferences**.
- 3 Under **Editor Preferences**, select **Replace menus and toolbars with the Simulink Toolstrip (Tech Preview)**. Click **Apply**.
- 4 Simulink closes all open models. When you reopen Simulink, the menu replaces the Simulink Toolstrip in any new or saved models.

When you activate or deactivate the toolstrip, Simulink saves your preferences for all models that you create or open.

Version History

R2019a includes a tech preview of the Simulink Toolstrip, and you may encounter performance issues when you enable the toolstrip. Documentation does not reflect the addition of the Simulink Toolstrip and toolstrip customization is not available.

Component-Based Modeling

Schedule Editor: Explicitly schedule the execution of your model components

In a model that contains many components, figuring out the schedule of components can be time consuming. The Schedule Editor allows you to easily view and edit the schedules in a graphical format.

The Schedule Editor shows a graph of the components in the model (known as partitions), the data communication between them, and a table of the execution order of those partitions. You can edit the schedule of partitions without disturbing the model structure. You can partition a subsystem block, a model block, and an export-function model. To learn more, see [Schedule Editor](#). To start exploring the Schedule Editor, see [Schedule an Export-Function Model Using the Schedule Editor](#) example.

The screenshot displays the Schedule Editor window. The main area shows a graph of partitions (components) and their execution order. The partitions are represented by colored boxes with their names and rates. The execution order is listed in a table on the right.

EXECUTION ORDER

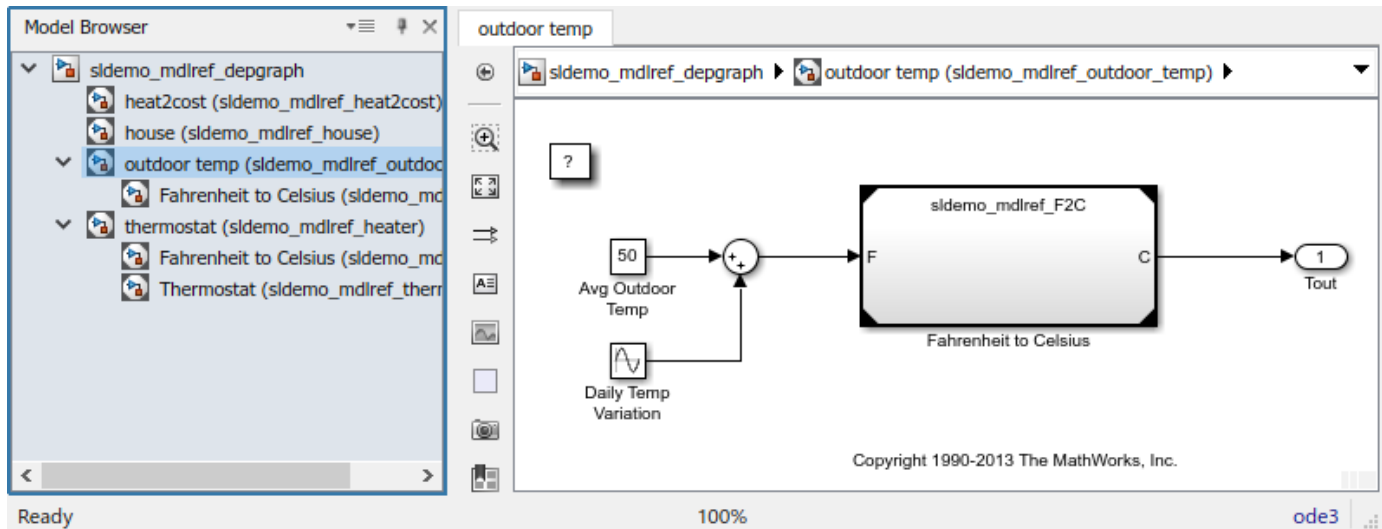
Order	Name	Rate
1	Cont	0
2	ThrottleComposition.TPSPrimaryRun5ms	0.001
3	ThrottleComposition.ActuatorRun5ms	0.005
4	ThrottleComposition.TPSSecondaryRun5ms	0.005
5	ThrottleComposition.MonitorRun5ms	0.005
6	ThrottleComposition.ControllerRun5ms	0.005
7	D2	0.005
8	D3	0.01
9	ThrottleComposition.APPSnrRun10ms	0.01

PROPERTY INSPECTOR

Property	Value
Name	ThrottleComposition.ControllerRun5ms
Rate	0.005
Type	Explicit periodic partition

Model Reference: Open referenced models in the same window as their parent models

Starting in R2019a, referenced models open in the same window as their parent models. Opening the referenced model in the same window reduces window clutter, provides easier navigation through model hierarchies, and improves signal tracing. For information on navigation options, see [Navigate Model Hierarchies](#).



Keyboard shortcut **Ctrl+S** now saves all modified models in the model hierarchy. To save only the active referenced model, use **Ctrl+Shift+S**.

Scope windows display simulation results for the most recently opened top model. Playback controls in scope blocks and viewers simulate the model containing that block or viewer. For more information, see *Scopes in Referenced Models*.

Functionality related to simulation or code generation, such as **Update Diagram**, applies to the entire model hierarchy. These actions are unaffected by whether the active model is a top or referenced model.

Perspective views apply only to the active model, which can be the top model or a referenced model. Functionality related to model properties or parameters, such as **Refresh Blocks**, also applies only to the active model.

To create a `Simulink.BlockPath` object that is context sensitive to model hierarchy, use the new `gcbp` function. The `hilite_system` function now supports `Simulink.BlockPath` objects.

Protected Models: Use models protected in previous releases for normal mode simulation

Starting with models protected in R2018b, you can use protected models in their original release and later releases for normal mode simulation and read-only view. For example, in R2019a, you can simulate models protected in R2018b.

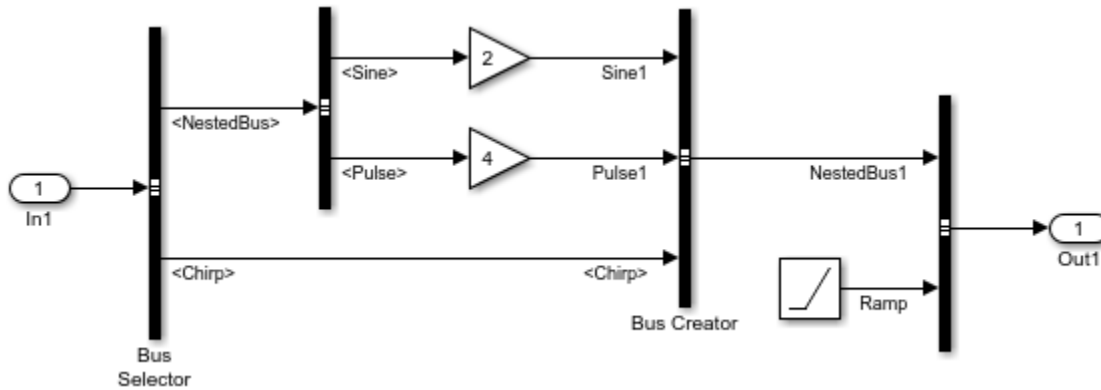
Buses: Simplify model interfaces with bus element port blocks

You can now use bus element port blocks at model interfaces. Bus element port blocks:

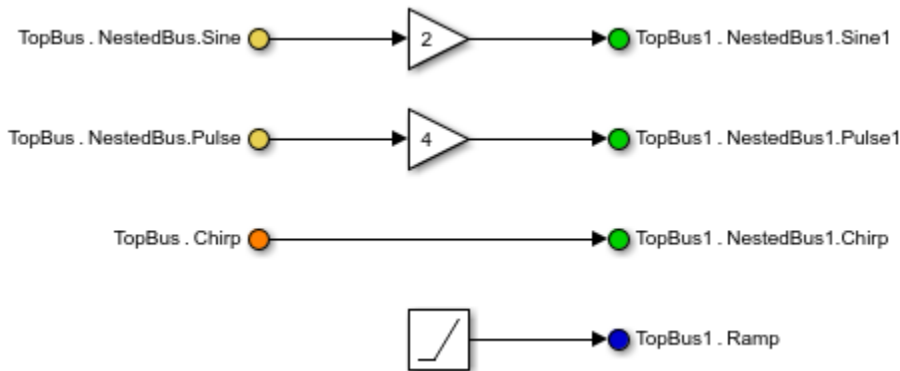
- Allow buses to cross a model boundary without a bus object.
- Allow you to write to all or some of the bus elements that the bus object defines when using a bus object. Unassigned bus elements use the ground value.
- Reduce signal line complexity and clutter in a block diagram.

- Make it easier to change the interface incrementally.
- Allow access to a bus element closer to the point of usage, avoiding the use of a Bus Selector and Goto block configuration.

For example, this model uses Inport, Bus Selector, Bus Creator, and Outport blocks.



Here is an equivalent model that uses In Bus Element and Out Bus Element blocks.



You can load simulation input data for a group of root-level In Bus Element blocks using a structure of timeseries, timetable, or SimulationDatastore objects. For an example, see Load Input Data for a Bus Using In Bus Element Blocks.

When you save output data to the workspace or a file, bus data defined by groups of root-level Out Bus Element blocks gets logged along with root-level Outport block data.

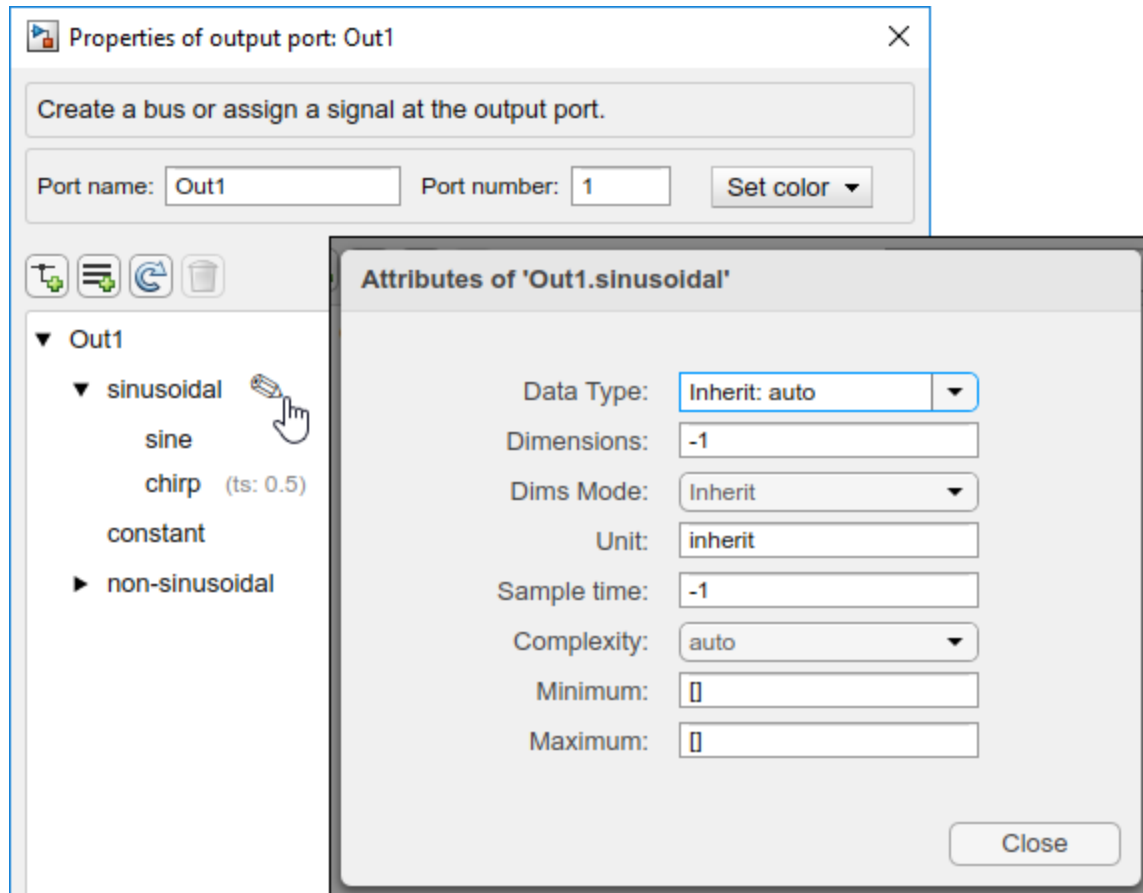
The choice of bus ports or traditional input and output ports does not affect standalone simulation or code generation.

Buses: Specify multirate buses and signal attributes at bus ports

You can now specify attributes for elements of a bus created using bus element port blocks. Bus ports support partial specification and use default values for all unspecified attributes. You do not need to specify a bus object for bus ports at subsystem or model interfaces.

Bus ports allow you to specify different sample time values for elements of a bus. If a bus object defines the bus, specifying **Sample time** at the bus port overwrites the bus object sample time definition. All other signal attributes must use the specification from the bus object.

To specify attributes, click the pencil button or the specified attributes.



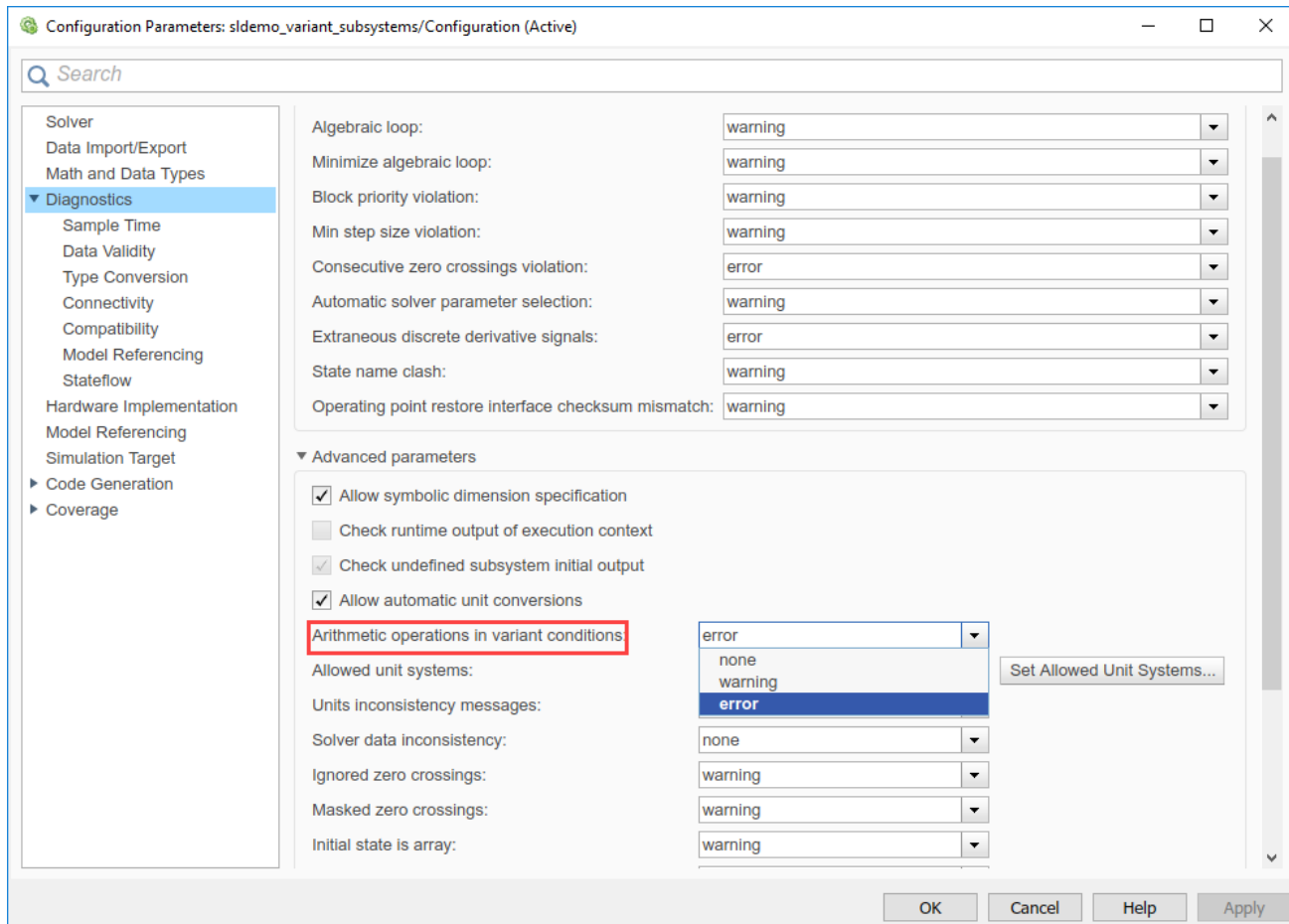
Variant Configuration Analysis Tool: Difference tool to compare variant configurations

From R2019a, you can use Variant Configuration Analysis Tool in Variant Manager to perform these functions:

- Compare different variant configurations for a model
- Get information on model references and libraries for each variant configuration

Control usage of arithmetic operations in Variant conditions

From R2019a, you can use the **Arithmetic operations in variant conditions** diagnostic to specify the diagnostic action to take place when arithmetic operations (+, -, *, idivide, rem) are found in variant conditions specified within Variant blocks. Simulink displays an error or a warning or takes no action based on the diagnostic selection you make. To select the diagnostic action, navigate to **Model Configuration Parameters > Diagnostics > Arithmetic operators in Variant Conditions** and then set the value to none or warning or error, based on your requirement.



By default, error is selected. When you set the value to none or warning, you can use arithmetic operations in variant conditions.

For example, if you use the condition $V * W == 10$ in a Variant Source block and request that the block produces preprocessor conditions in the Simulink Coder generated code. This results in generated C code containing `"#if V*W == 10"`. Simulink uses int32 types for V and W, whereas the integer types used by the compiler are implementation dependent. So, for large values of V and W, there could be a difference in behavior between simulation and code generation. So, it is recommended to use the default value error. If your model is currently using arithmetic operations, you must remove them and not change the diagnostic.

Global data on signals and states are guarded

Starting in R2019a, global signals and states are guarded in the generated code with their corresponding variant conditions in Simulink models that contain Variant Source or Variant Sink blocks.


```
/* Exported block signals */  
#if V == 1 || V == 2  
  
real_T myOutput;          /* '<Root>/Gain3' */  
  
#endif
```

Code generation for multi-instance referenced models with variant Simulink Functions

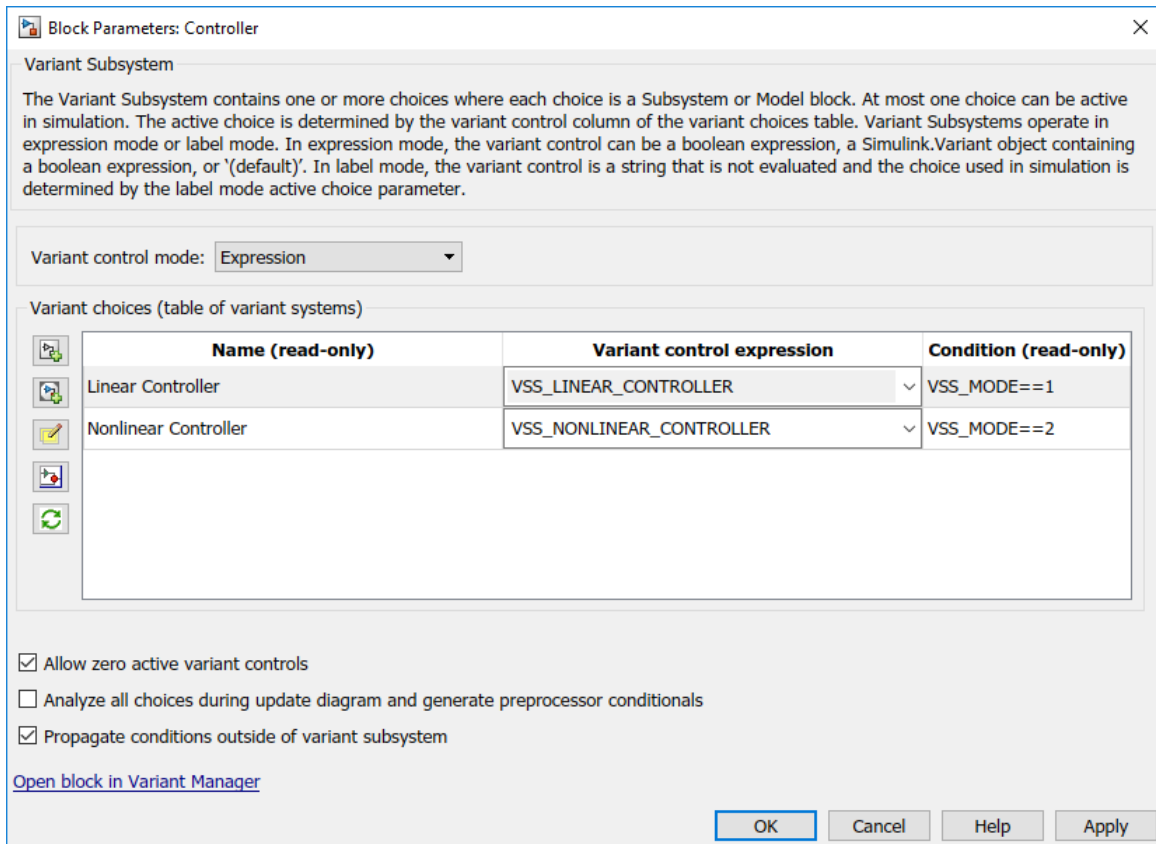
Before R2019a, multi-instance referenced models with variant Simulink Functions were not supported for code generation.

From R2019a, Simulink supports using multi-instance referenced models with variant Simulink Functions for code generation.

Propagate variant conditions with no active variant controls in Variant Subsystem

Before R2019a, propagation of variant conditions outside a Variant Subsystem block was supported only when **Allow zero active variant controls** option was disabled.

From R2019a, you can choose to propagate variant conditions outside a Variant Subsystem block for zero or one active variant control specified. The generated code for such modeling pattern has the outputs guarded resulting in memory optimization.

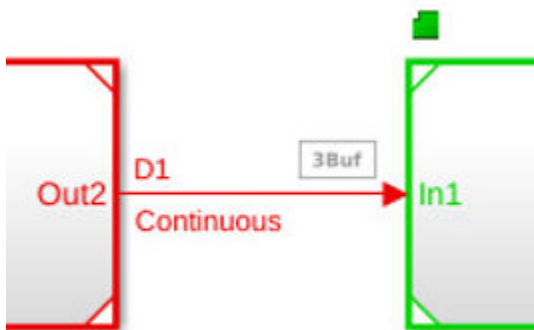


Rate transition improvements in models configured for concurrent execution

Rate Transition blocks are now supported at the top level of models using explicit partitioning for concurrent execution. In past releases, only automatically inserted rate transitions were supported.

To enable the software to automatically insert rate transitions where needed, in the **Solver** pane of the Configuration Parameters, under **Tasking and sample time options**, select **Automatically handle rate transition for data transfer**.

When you update the model diagram, a 3buf label appears on the canvas to indicate where the rate transitions are inserted. This badge appears on models using both explicit and implicit partitioning.



Version History

Starting in R2019a, update code that references the `AutoInsertRateTranBlk` property of the `Simulink.GlobalDataTransfer` class. For example, replace instances of

```
dt = get_param(gcs, 'DataTransfer');
dt.AutoInsertRateTranBlk = 1;
```

with

```
set_param(gcs, 'AutoInsertRateTranBlk', 'on');
```

Functionality being removed or changed

Specifying variant models in Model blocks is no longer supported

Warns

Starting in R2019a, specifying variant models in Model blocks is no longer supported. The Model Variants block, which was a Model block preconfigured to specify variant models, is also no longer supported. Loading a model that contains these blocks converts them to Variant Subsystem blocks that contain Model blocks that represent the variant choices.

Variant Subsystem blocks provide these advantages:

- You can mix Model blocks and Subsystem blocks as variant choices.
- You can specify variants that have different numbers of input and output ports.

The Variant Subsystem block parameter **Analyze all choices during update diagram and generate preprocessor conditionals** behaves differently from the Model block parameter **Generate preprocessor conditionals**. Instead of causing simulation and update diagram to compile the active variant only, the Variant Subsystem block parameter compiles all variants.

If you have scripts that use Model block parameters for variants, you must update them to use Variant Subsystem block parameters.

Convert Configurable Subsystem blocks to Variant Subsystem blocks

Warns

The Configurable Subsystem block will be removed in a future release. The Configurable Subsystem blocks in existing models must be converted to Variant Subsystem blocks. Variant Subsystems offer more capabilities than Configurable Subsystems with these advantages:

- Model blocks and Subsystem blocks can be mixed as variant systems
- Flexible input or Output ports support same or different number of input and output ports

To convert a Configurable Subsystem block to a Variant Subsystem block, right-click the Configurable Subsystem block and select **Subsystem & Model Reference > Convert to > Variant Subsystem**. For more information, see [Convert Configurable Subsystem to Variant Subsystem](#).

During conversion, Simulink performs these operations:

- Replaces the Subsystem block with a Variant Subsystem block, preserving ports and connections.
- Adds the original subsystem as a variant choice in the Variant Subsystem block.

- Preserves links to libraries. For linked subsystems, Simulink adds the linked subsystem as a variant choice.

The Override variant conditions and use the following variant option is removed

Warns

From R2019a, the **Override variant conditions and use the following variant** option no longer exists. To get the same functionality, use the `Label` option in Variant control mode instead.

The label mode allows you to specify the name-based variant controls (Label mode active choice). In label mode, the variant control is not evaluated and the choice used in simulation is determined by the `Label mode active choice` parameter.

Additionally, to choose the active variant based on the evaluation of variant conditions, select `Expression` as the variant control mode. In expression mode, the variant control can be a Boolean expression, a `Simulink.Variant` object containing a Boolean expression, or `'(default)'`.

Simulink models that were created prior to R2019a with **Override variant conditions and use following variant** option will continue to work when you update the model to a newer version.

Variant Subsystem block parameter name change

Warns

These Variant Subsystem block parameters have been renamed:

Old Name	New Name
<code>ActiveVariant</code>	<code>CompiledActiveChoiceControl</code>
<code>ActiveVariantBlock</code>	<code>CompiledActiveChoiceBlock</code>

Simulink models that were created prior to R2019a and use the above mentioned old parameter names will continue to work when you update the model to a newer version.

Updated arguments for function `slbuild`

Still runs

In R2019a, some arguments used with the `slbuild` function have been updated. Legacy arguments are supported for backward compatibility.

To take advantage of the updated arguments:

- Use `'StandaloneCoderTarget'` instead of `'StandaloneRTWTarget'`
- Use `'ModelReferenceCoderTarget'` instead of `'ModelReferenceRTWTarget'`
- Use `'ModelReferenceCoderTargetOnly'` instead of `'ModelReferenceRTWTargetOnly'`

Updated argument value for function `Simulink.SubSystem.convertToModelReference`

Still runs

In R2019a, an argument value used with the `Simulink.SubSystem.convertToModelReference` function has been updated. The legacy argument value is supported for backward compatibility.

To take advantage of the updated `'BuildTarget'` argument value, use `'Coder'` instead of `'RTW'`.

Project and File Management

Projects: Use projects in MATLAB

In R2019a, you can use projects in MATLAB, with or without Simulink. Therefore, you now can share projects with users who do not have Simulink.

For information on basic project workflows in MATLAB, see [Projects \(MATLAB\)](#).

Projects continue to provide additional tools to help with Simulink workflows. For example:

- Opening models and running customizations on startup
- Checking for shadowed model files
- Dependency analysis of models, subsystems, block libraries and individual blocks, data files, requirements, and generated code
- Automatic refactoring help for models, libraries, library links, model references, model callbacks, S-functions, buses and bus elements
- Comparing and merging differences in models.

To learn more about project workflows in Simulink, see [Project Management](#).

Model Comparison: Get more accurate Simulink block matches when comparing models

In R2019a, model comparison uses an improved algorithm for clearer Simulink block matches. The model comparison algorithm is improved to account for block connectivity. Comparison reports are more intuitive and contain fewer mismatches.

To learn more about comparing and merging models, see [Model Comparison](#).

Project Files: Find and add files to the project in bulk

In R2019a, you can click **Add Files** to easily find and add all files that are not in the project. This can be helpful for adding many files at once and any time you want to check for new files not yet added to the project. You no longer need to locate files to add manually by browsing the all files view. To learn more, see [Add Files to the Project](#).

Project Dependency Analysis: Navigate required files with improved highlighting and filtering

In R2019a, the dependency analysis graph in a project clearly highlights selected files and their impacted files. It is now easier to see the files and their dependencies in the context of the other project files. Selected files and their dependencies are highlighted with strong contrasting colour and thick connection lines. Other files are dimmed with white backgrounds. If you filter files, you can now click the arrow to undo the filtering and return to previous views. You can filter files using context menu items on selected files such as **Find All Dependencies**, or by typing in the search box. The undo functionality for filtering helps you efficiently navigate the dependency graph. To learn more, see [Perform Impact Analysis](#).

Project Startup: Control Simulink starting and running customizations when opening projects

In R2019a, when you open a project, you can specify whether Simulink starts and whether to run customizations. If you have a hierarchy of referenced projects, you can specify for each project if it needs to run `sl_refresh_customizations`. The customizations refresh once after opening or closing a hierarchy of referenced projects. Use this feature to reduce multiple calls to `sl_refresh_customizations` and the need to use project startup and shutdown files. To learn more, see Automate Startup Tasks.

Project Archives: Share projects in a single file and check for required products

In R2019a, you can share a project in a single file that you can open directly from the file system, and it checks for required products. Projects have a new archive file format `mlproj`. To create a new project from an archive, double-click the `mlproj` file in the Current Folder, Windows Explorer, or Apple Finder. If you do not have a product that the project requires, you see a warning with a link to the Add-On Explorer to get missing products. For more information, see Create a New Project from an Archived Project.

Older projects use zip files instead of `mlproj` files and both continue to work.

Functionality being removed or changed

Simulink project API is not recommended

Still runs

Starting in R2019a, instead of `simulinkproject` and related functions, use the `currentProject` or `openProject` functions in MATLAB. The Simulink project API will continue to be supported, but, after R2019a, new features might be available only if you use the new MATLAB project API. There are no plans to remove the Simulink project API at this time.

The new MATLAB project API is part of the R2019a functionality enabling you to use projects in MATLAB, with or without Simulink. You can now share projects with users who do not have Simulink.

Data Management

Simplified workflow for specifying values for model arguments in a nested model reference hierarchy

Model arguments are parameters in the model workspace whose value can be different for each instance of the model. Previously, promoting model arguments through multiple levels of a model hierarchy required several configuration steps at each level. In R2019a, intermediate models in the model reference hierarchy can allow parent models to override the value of model arguments without having to create additional model workspace parameters.

To configure model arguments for your hierarchy:

- 1 Use the Model Explorer to define parameters in the model workspace of the referenced model. The following workspace variables can be configured as model arguments:
 - Objects of class `Simulink.Parameter` or its subclasses.
 - Objects of class `Simulink.LookupTable` or its subclasses.
 - MATLAB variables that can be assigned to the Value property of a `Simulink.Parameter` object. For example: numeric MATLAB variables, `fi` objects, and enumerated values of a Simulink enumerated data type.
- 2 Use the Model Data Editor, Property Inspector, or Model Explorer to select the **Argument** check box.
- 3 Specify the instance-specific value of the parameter on the Model block. A model workspace parameter configured as a model argument is exposed as a tunable parameter on the Model block in the parent model.
- 4 In intermediate models, decide if you want to override the value of the parameter at the next level of the model hierarchy. To override the value, select the **Argument** check box on the Model block.

For more information, see [Parameterize Instances of a Reusable Referenced Model](#).

Componentized management of global data in a model reference hierarchy

R2019a provides a more streamlined approach to componentized management of global data in a model reference hierarchy with these updates to the data dictionary:

- Top model data dictionaries no longer have to link to submodel dictionaries.
- Duplicate data definitions can exist in a model reference hierarchy under these conditions:
 - Each model in the hierarchy can see only one definition.
 - Definitions must be the same across models in the hierarchy.

For an example, see [Partition Data for Model Reference Hierarchy Using Data Dictionaries](#).

Root Inport Mapper Updates

The Root Inport Mapper has these changes:

The Preview tab has been removed. To visualize signals, use other visualization tools, such as:

- Signal Editor
- **Simulation Data Inspector**

You can also use the `Simulink.SimulationData.Dataset` or `Simulink.SimulationData.DatasetRef` plot method to import data into Signal Editor or Simulation Data Inspector, from which you can plot the data.

Signal Editor user interface supports improved fixed-point data type proposals

The Signal Editor user interface now provides histograms of your proposed fixed-point data. Use these histograms to help you select the fixed-point data type for your signal data. For more information, see [Create Signals with MATLAB Expressions and Variables](#).

Override parameter values in a configuration reference

Before R2019a, to change a parameter value in a configuration reference, you opened and edited the referenced configuration set. In R2019a, you can override a parameter in the reference and set a value that is different from the value in the standalone configuration. Right-click the parameter and select **Override** from the context menu. You can edit the value without affecting the standalone configuration set or other configuration references. You can also restore the referenced value again through the context menu of the overridden parameter. For more information, see [Change Parameter Value in a Configuration Reference](#).

Simplified configurations section in the Model Explorer

Enhancements to the Model Explorer reduce complexity and make it easier to perform actions on elements in your model. A Configurations node has replaced the configuration objects that previously appeared in the **Model Hierarchy** pane. Access the configuration sets and configuration references through the **Contents** pane when you select the Configurations node.

Version History

Previously, when you selected a configuration node in the **Model Hierarchy** pane, the configuration panes appeared in the **Contents** pane. In R2019a, you can no longer access configuration panes through the Model Explorer. To copy and paste configuration panes, use the command-line interface to copy and paste the associated configuration components. For more information, see [Copy Configuration Set Components](#).

Also, when you selected a pane, the **Dialog** pane displayed its parameters. In R2019a, you can no longer access configuration parameters through the Model Explorer. Instead, access configuration parameters through the Configuration Parameters dialog box. In the **Contents** pane, right-click the configuration object and select **Open**. Alternatively, from the Simulink Editor, select the Configuration Parameters button to open the dialog box.

Previously, you could use context menus and drag-and-drop operations to copy, move, and delete model configurations among the model nodes in the **Model Hierarchy** pane. In R2019a, perform these operations using the configuration objects in the **Contents** pane and the Configurations nodes in the **Model Hierarchy** pane.

View Simulink data dictionary version information

In R2019a, the **Details** pane of the MATLAB editor shows the Simulink version in which a data dictionary is saved. For more information, see Dictionary Usage for Models Created with Different Versions of Simulink.

Enable model to access data directly from base workspace and data dictionary

In R2019a, managing data outside your model has become more flexible.

- Previously, you could configure a model to access data from the base workspace or a data dictionary. In R2019a, you can configure models to access both. In the Model Properties dialog box, on the **Data** tab, you can:
 - Configure the **Enable access to base workspace** setting.
 - Link to a data dictionary.
- The process of migrating your data from the base workspace to a data dictionary is independent of the process of linking your model to a dictionary. To start the migration, from the **Data** in the Model Properties dialog box, click **Migrate Data**.

For more information, see Migrate Models to Use Simulink Data Dictionary.

Version History

Before R2019a, a model could have direct access to the base workspace or a data dictionary. In R2019a, this limitation no longer exists. A new model still has access to the base workspace by default, but does not automatically lose access when it is linked to a data dictionary. If you have code that is based on the assumption that the model has access to either the base workspace or a data dictionary, this code must be modified to account for models with access to both.

Other model configuration issues to consider with this change:

- For models created in previous releases, **Enable access to base workspace** is selected only if the model is not currently linked to a data dictionary.
- You cannot set **Data Dictionary** to '' unless you select **Enable access to base workspace** .

Enhanced user interface for edit-time checks

Edit-time notifications in both the Simulink Editor canvas and block dialog boxes use the same interface as the Diagnostic Viewer. If an error or warning has a predefined fix, the notification in the updated interface can provide you with the same suggested actions, with corresponding **Fix** buttons, available to you at compile time in the Diagnostic Viewer.

For more information, see Model Editing Environment.

Create missing data stores during edit and compile time

Previously, the Diagnostic Viewer reported an error for a missing Data Store Memory block or global data store definition at compile time without providing suggested actions. In R2019a, an edit-time

check can also report this error. Both the Diagnostic Viewer and the edit-time check provide a fix for these suggested actions:

- Create the missing data store by either adding a Data Store Memory block or creating a global signal object.
- Load a file into the base workspace. This fix is available only if the model has access to the base workspace.

Edit-time fix provided for renamed variables in a data dictionary

If a model uses a variable in a data dictionary, an edit-time check reports an error when that variable is renamed. The error notification provides a fix to update the reference to the new variable name. If multiple models have a reference to the renamed variable, you must update each reference individually.

For information on managing data dictionaries, see [What Is a Data Dictionary?](#)

Enhanced name conflict management in Model Explorer

In Model Explorer, when you copy and paste (or drag and drop) items from one data source to another, you can manage the resolution of name conflicts. In the dialog box, choose an action for each item that conflicts with an existing item:

- **Skip** — Default action when the conflicting items have the same value.
- **Replace** — Default action when the conflicting items have different values.
- **Keep both** — Preserve both copies. The name given to the copied item follows the naming rules that are used in the MATLAB workspace.

If the source and destination are the same, the software resolves the name conflict by following the naming rules that are used in the MATLAB workspace.

For more information about using Model Explorer, see [Search and Edit Using Model Explorer](#).

Support for uint64 and int64 as built-in Simulink data types

Previously, Simulink supported 64-bit integers as 64-bit fixed-point data types (`fix64` and `ufix64`). In R2019a, specify 64-bit integers in Simulink as the built-in data types `int64` and `uint64` on:

- Block data type parameters
- Mask data type parameters
- Data type properties of data objects, alias type, and bus elements

View and edit these built-in data types in interfaces such as block dialog boxes, Property Inspector, Model Data Editor, Model Explorer, Mask Editor, and Data Type Assistant.

For more information on built-in data types, see [Data Types Supported by Simulink](#).

New data type propagation rules for Sum, Gain, and Product blocks

There are now new output data type choices for the Sum, Gain, and Product blocks. These new data type propagation rules give you more control over the range and scaling of the output.

- **Inherit: Keep MSB** - This rule selects an output data type that maintains the full range of the operation and then reduces the precision of the output value to a size appropriate for the target hardware. This rule will never produce an overflow.

This rule is available for the Sum, Product, and Gain blocks.

- **Inherit: Keep LSB** - This rule selects an output data type that maintains the precision of the operation but reduces the range if the full type does not fit on the target hardware. This rule can produce overflows.

This rule is available for the Sum block.

- **Inherit: Match Scaling** - This rule attempts to maintain the scaling of the output data type. This rule can produce overflows.

This rule is available for the Product and Gain blocks.

Support for bus element ports in Model Data Editor

In the Model Data Editor, bus element ports appear as rows on the **Inports/Outports** tab with these features:

- The editor displays block properties as read-only.
- Properties stored on the signal line are still writeable.
- The **Source** column does not display the underlying block name, but displays the same concatenated string displayed on the canvas. This string is a hyperlink that you can click to open the block dialog box.

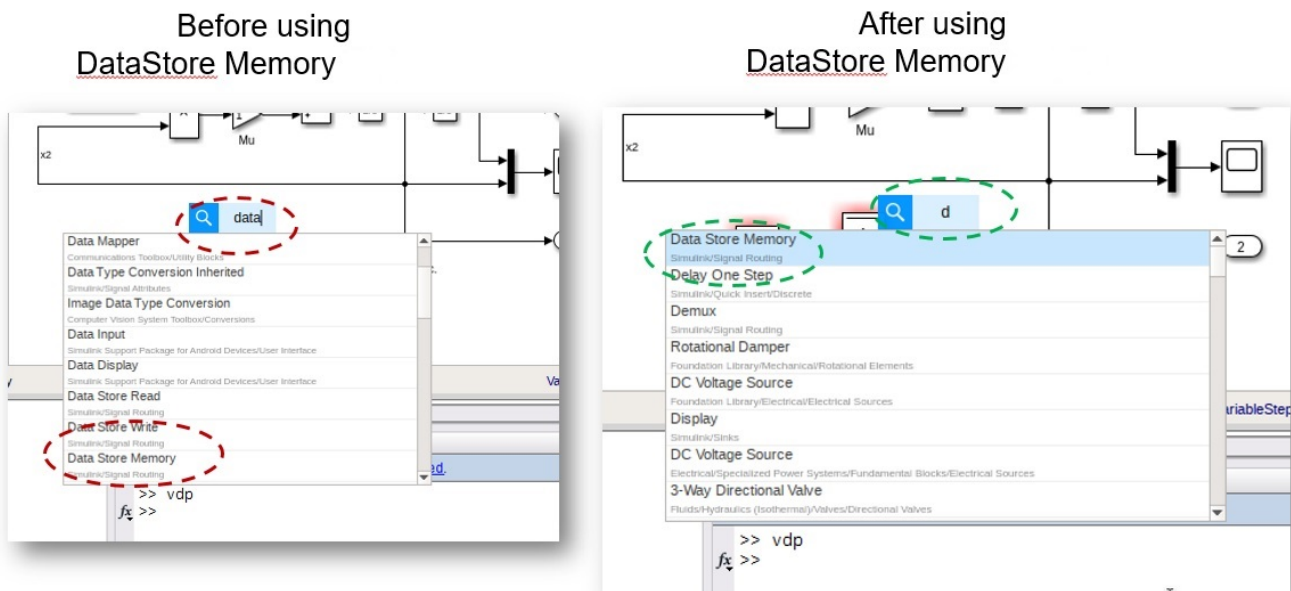
Block Enhancements

Call Simulink Functions from MATLAB System block

MATLAB System block can call Simulink functions through the specified System object. To call a Simulink function in a MATLAB System block, register the Simulink function names using `getSimulinkFunctionNamesImpl`. When the System object has direct feedthrough, invoke the functions in `stepImpl` method; otherwise, invoke the functions in the `updateImpl` and `outputImpl` methods of the System object. See [Call Simulink Functions from MATLAB System Block](#) to learn more.

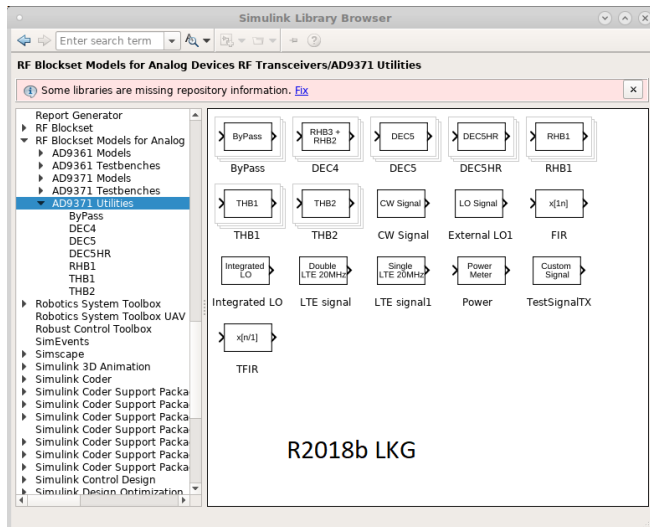
Improved results with Quick Block Insert

Quick Block Insert results are now ranked dynamically based on the usage patterns of individual users.

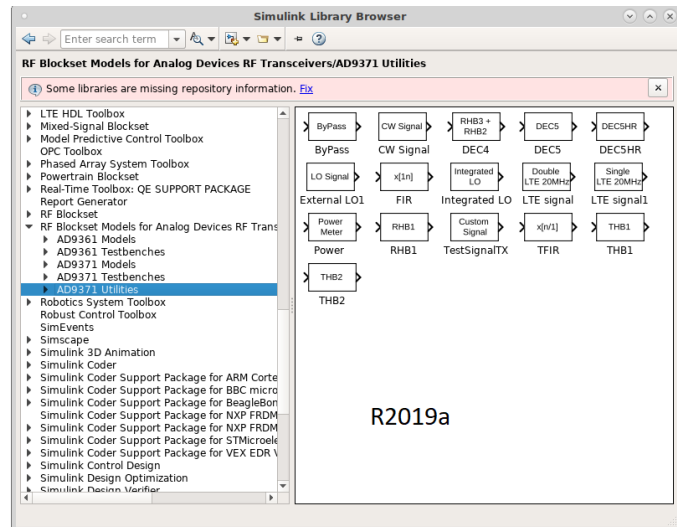


Effective identification of leaf and container blocks

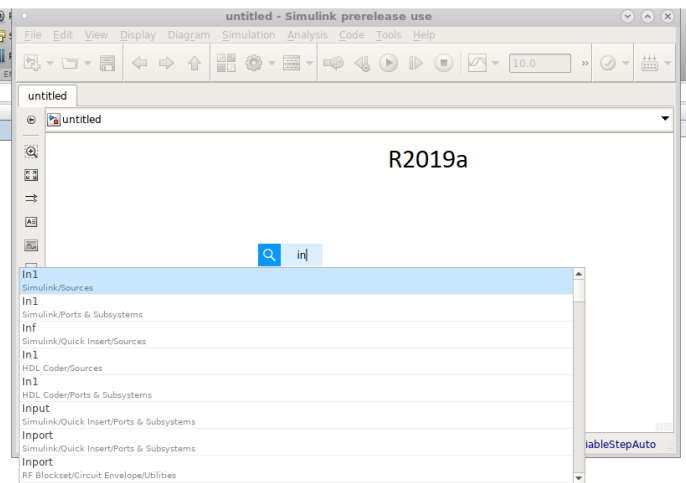
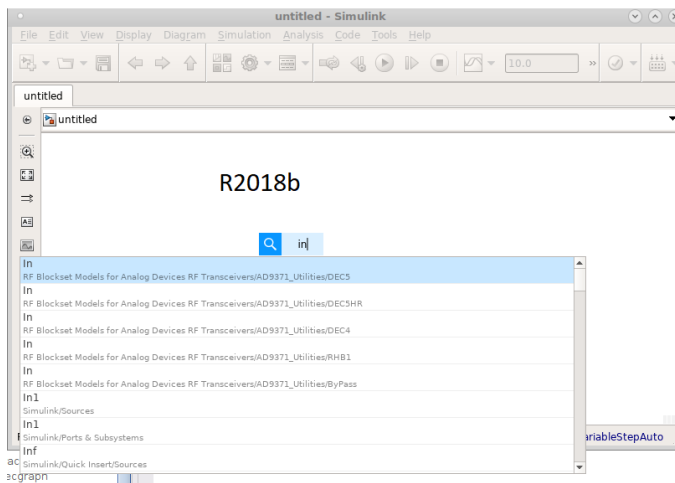
The logic for identifying leaf and container blocks in the Library Browser has been improved. This improvement helps to prevent misclassification of blocks.



R2018b LKG



R2019a



Mask Parameter Constraints: Create custom error messages

From R2019a, while creating a mask parameter constraint, you can choose to add a customized error message to be displayed for a validation failure. The reporting of validation failure is also improved.

Additionally, mask parameter constraint now supports string data type.

For more information, see [Add Constraints to Mask Parameters](#).

set_param command on mask parameter executes callback code

Before R2019a, using the `set_param` command on a mask parameter to change its value did not execute the associated mask callback code.

From R2019a, when you use the `set_param` command on a mask parameter, the callback code associated with the mask parameter is executed.

For more information, see Mask Callback Code.

Mask parameter callbacks execute in temporary workspace

Before R2019a, execution of the mask parameter callbacks occurred in the base workspace. This resulted in dirtying of the base workspace. For example, if a variable `var` was used in the mask callback code, during callback execution, `var` would be created or replaced in the base workspace.

From R2019a, the execution of the mask parameter callbacks happens in a temporary workspace. The variables defined in the mask callback code can only be accessed in the scope of the mask parameter callback. To access these variables outside of the scope of mask parameter callback, the variables must be defined in the base workspace using the `assignin` command. For example, `assignin('base','var','3')`.

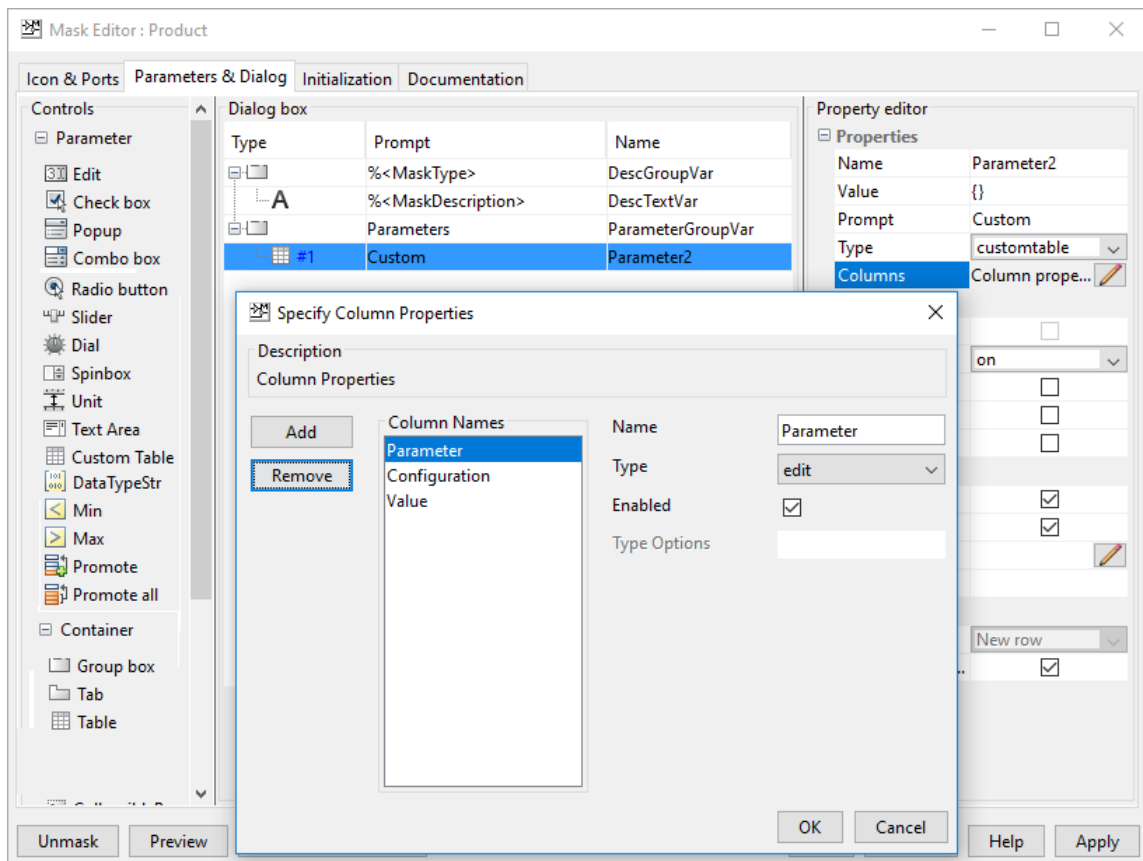
For more information, see Mask Callback Code.

Modification of mask parameters is restricted from callback

Mask parameter addition, deletion, or modification is now restricted from callback.

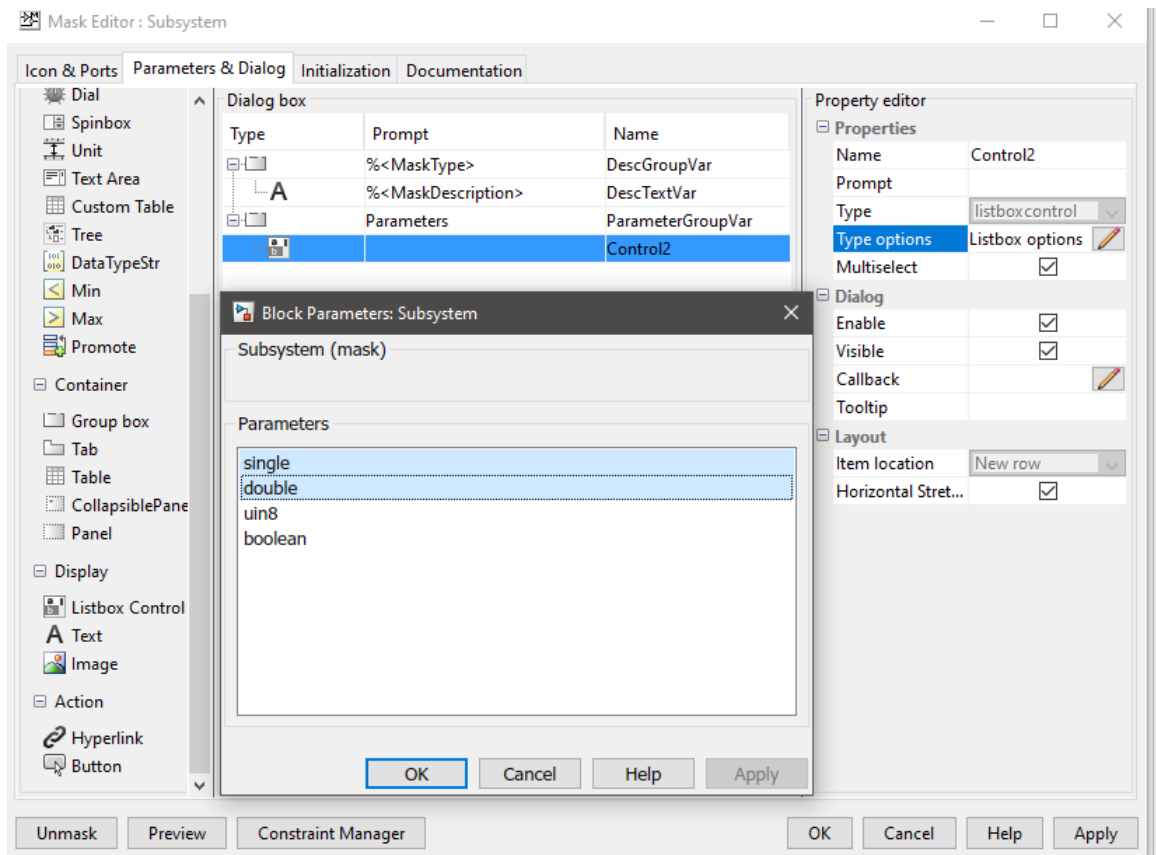
Simulink Mask: Enhance usability with additional parameters

- **Custom Table:** Design a customized table on a block mask.



For more information, see [Customize Tables for Masked Blocks and Control Custom Tables Programmatically](#).

- **Listbox Control:** Create a list on a mask to display the acceptable parameter values.



For more information, see Mask Editor Overview.

Trigonometric Function and Sqrt block supports symbolic dimensions

The Trigonometric Function (all supported functions), Sqrt, Signed Sqrt, and Reciprocal Sqrt blocks now support symbolic dimensions.

Direct Lookup Table (n-D) block updates

The Direct Lookup Table (n-D) block now supports:

- Scaled double data type for index ports. In previous releases, this block returned an error if it received a signal with a data type of scaled double.
- Fixed-point data types for the index port into n -dimensional tables. These fixed-point data types must have bias equal to 0, slope equal to 1.0, fractional length equal to 0, and word length less than or equal to 128.

Lookup table object updates

To help you create lookup table objects more quickly, when you create a lookup table object from a lookup table block that has table and/or breakpoint information, the corresponding parameters of the new lookup table object are now automatically populated with the block information. You can use these initial values, or change them as you want.

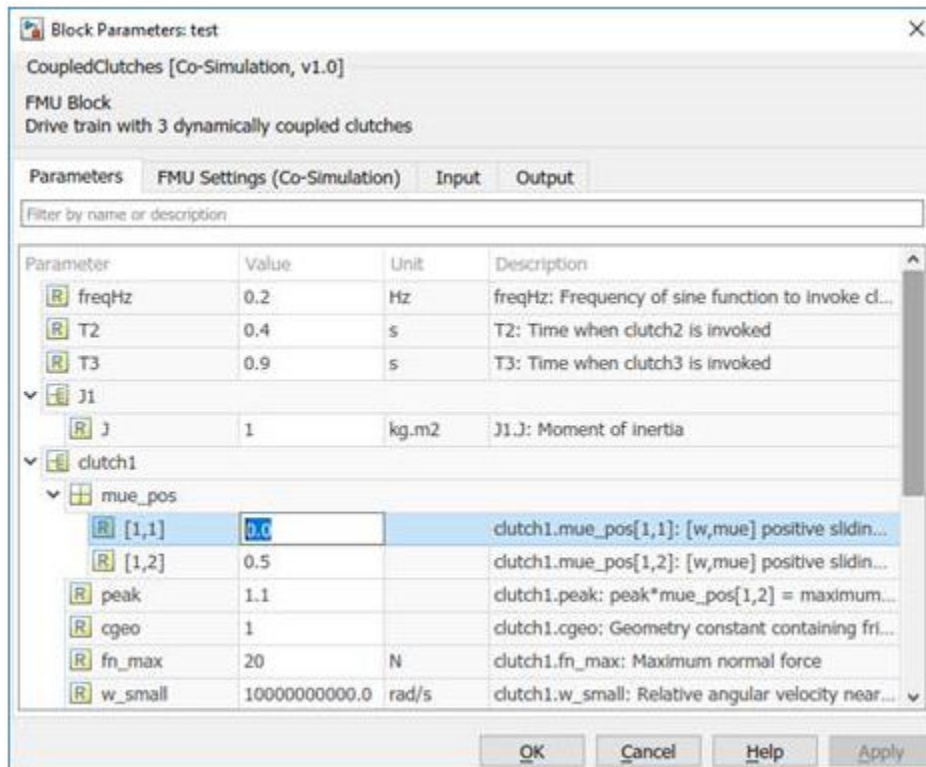
In previous releases, an empty table was created, and you needed to manually enter the table and breakpoint information.

Display block updates

The Display block **Format** has been renamed to **Numeric display format**. This change indicates that the **Numeric display format** parameter affects only numeric input for the Display block. This parameter does not affect the display of strings.

FMU Block Updates

The FMU block lists parameters in a tree view. This improves readability when the parameter is a structure or an array.



Programmatically add and configure Dashboard blocks

You can use functions such as `add_block` and `set_param` to create and configure Dashboard blocks in your model.

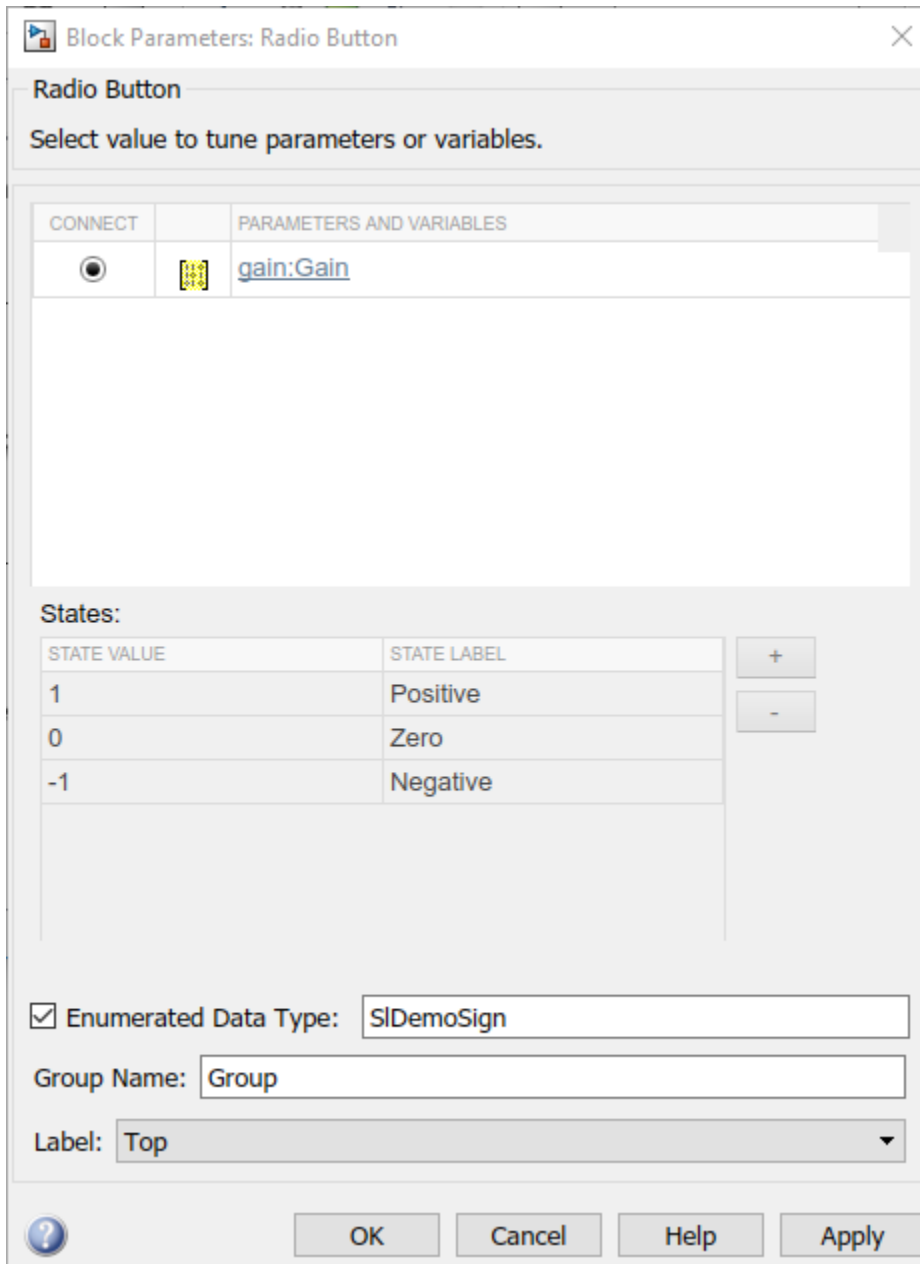
Display Dashboard block enhancements

The Display block in the Dashboard library has improved support for enumerated data types. In addition, the Display block now supports string data and can display numeric data in all formats supported by the MATLAB `format` command.

short	3.1416	shortEng	3.1416e+000	string	Hello!
long	3.141592653589793	longEng	3.14159265358979e+000		
shortE	3.1416e+00	bank	3.14		
longE	3.141592653589793e+00	plus	+		
shortG	3.1416	hex	400921FB54442D18	enum	Positive
longG	3.14159265358979	rat	355/113		

Enhanced enumerated data type support for the Radio Button block

You can use an enumeration class to configure the values and labels for the states of a Radio Button block. To configure the **States** for a Radio Button block with an enumerated data type, select the **Enumerated Data Type** parameter. Enter the name of the enumeration class you want to use in the text box. In this example, the enumeration class `SI Demo Sign` defines the state values and labels. When you use an enumeration class to configure the **States** for the block, you cannot manually edit, add, or remove states from the **States** table.



Parameter Writer block

The Parameter Writer block changes block parameter values in a referenced model by writing to instance parameters belonging to the Model block.

Use the Parameter Writer block with the Initialize Function and Reset Function blocks to respond to events. Set parameters based on environment sensor values and tune algorithms during execution. For more information, see Parameter Writer block reference.

Input/Output Scope for C Caller Blocks

Use the C Caller block to map an input passed by a pointer in your C functions. With this scope, you can call a C function accesses and updates a signal passed by a pointer and output the result from the C Caller block without a wrapper function. Simulink ports created in C Caller block using InputOutput scope have the same name for input and output ports. Reusing the buffer for input and output ports may optimize the memory use depending on the signal size and block layout.

Signal Editor block updates

To enable the Signal Editor block to label output signal lines, set the new `PreserveSignalName` command-line property to 'on'. For example:

```
set_param(gcf, 'PreserveSignalName', 'on')
```

signalBuilderToSignalEditor function updates

The `signalBuilderToSignalEditor` function now more efficiently converts Signal Builder block group names to scenario names, including Unicode® text. For example:

- Unicode group names are prefixed with `Test_Case_`.
- Group names are converted with leading and trailing white spaces removed.
- Group names are converted with trailing underscores removed.
- Remaining numbers maintained in the scenario name.

Signal Builder block update

The Signal Builder block is not recommended. Use the Signal Editor block instead.

Connection to Hardware

Perform external mode simulation on Arduino using XCP over serial, TCP/IP, and Wi-Fi connections

The Simulink Support Package for Arduino Hardware supports external mode simulation using Universal Measurement and Calibration Protocol (XCP) over serial, TCP/IP, and Wi-Fi® connections. XCP-based external mode enables parameter tuning and signal monitoring using Simulation Data Inspector and Dashboard blocks.

Perform external mode simulation on Raspberry Pi using XCP over TCP/IP

The Simulink Support Package for Raspberry Pi Hardware supports external mode simulation using Universal Measurement and Calibration Protocol (XCP) over TCP/IP. XCP-based external mode enables parameter tuning and signal monitoring using Simulation Data Inspector and Dashboard blocks.

Trigger a downstream function-call subsystem using Arduino External Interrupt block

The Simulink Support Package for Arduino Hardware supports the External Interrupt block. Using this block, you can create an Interrupt Service Routine (ISR) to invoke a downstream Function-Call Subsystem associated with the block. Invoking the function-call subsystem executes the output methods of the blocks within the subsystem in a sorted order.

Measure and control motor parameters using Arduino MKR motor carrier blocks

The Simulink Support Package for Arduino Hardware supports the MKR motor carrier blocks shown in the table. Using these blocks, you can measure and control the parameters of the motor connected to an MKR motor carrier on Arduino hardware.

Blocks	Usage
Servo Write	Sets the shaft angle of the servo motor connected to the carrier.
M1M2DCMotors	Controls the speed and direction of the DC motor connected to port number M1 or M2 of the carrier.
M3M4DCMotors	Controls the speed and direction of the DC motor connected to port number M3 or M4 of the carrier.
Encoder	Reads the position and rotational speed of the motor connected to the carrier.
Battery Read	Reads the voltage of the battery connected to the carrier.

MATLAB Function Blocks

Sparse matrix inputs for more functions in MATLAB Function blocks

These functions now support sparse matrix inputs in a MATLAB Function block:

- chol
- tril
- triu

Expanded support for the vecdim input argument in MATLAB Function blocks

MATLAB Function blocks now support the vecdim input argument for these functions:

- all
- any
- max
- mean
- median
- min
- mode
- prod
- std
- sum
- var

MATLAB Function block support for class properties with string scalar initial values

In R2019a, in a MATLAB Function block, you can include a function that uses a class property whose initial value is a string scalar. For example:

```
classdef myClass
    properties
        prop = "myString";
    end
end

function out = useClass
    obj = myClass;
    out = obj.prop;
end
```

Behavior change for character vector or string scalar input to ismethod in MATLAB Function blocks

Starting in R2019a, in a MATLAB Function block, specifying the first input (obj) to `ismethod` as a character vector or string scalar results in an error.

Version History

In previous releases, if you specified the first input to `ismethod` as a character vector or string scalar, the MATLAB Function block treated the value as the name of a class.

Code generation for more MATLAB functions

- `arrayfun`
- `accumarray`
- `bounds`
- `categorical`
- `categories`
- `cellstr`
- `cospi`
- `dec2base`
- `discretize`
- `hasFrame`
- `ind2rgb`
- `iscalendarduration`
- `iscategory`
- `isdatetime`
- `isduration`
- `isordinal`
- `isprotected`
- `isundefined`
- `normalize`
- `pause`
- `read` of `VideoReader`
- `readFrame` of `VideoReader`
- `tic`
- `toc`
- `qrupdate`
- `rat`
- `sinpi`
- `VideoReader`

Code generation for more 5G Toolbox functions

- `getTransportBlock`
- `nrDLSCH`
- `nrDLSCHDecoder`
- `nrLowPAPRS`
- `nrPUCCH0`
- `nrPUCCH1`
- `nrPUCCH2`
- `nrPUCCH3`
- `nrPUCCH4`
- `nrPUCCHHoppingInfo`
- `nrPUCCHPRBS`
- `nrPUSCH`
- `nrPUSCHCodebook`
- `nrPUSCHDecode`
- `nrPUSCHDescramble`
- `nrPUSCHPRBS`
- `nrPUSCHScramble`
- `nrTransformPrecode`
- `nrTransformDeprecode`
- `nrUCIDecode`
- `nrUCIEncode`
- `nrULSCH`
- `nrULSCHDecoder`
- `nrULSCHInfo`
- `resetSoftBuffer`
- `setTransportBlock`

Code generation for more Audio Toolbox functions

- `gtcc`
- `harmonicRatio`
- `imdct`
- `kbdwin`
- `mdct`
- `melSpectrogram`
- `spectralCentroid`
- `spectralCrest`
- `spectralDecrease`

- spectralEntropy
- spectralFlatness
- spectralFlux
- spectralKurtosis
- spectralRolloffPoint
- spectralSkewness
- spectralSlope
- spectralSpread
- erb2hz
- bark2hz
- mel2hz
- hz2erb
- hz2bark
- hz2mel
- octaveFilterBank
- gammatoneFilterBank

Code generation for more Automated Driving Toolbox functions

These Automated Driving Toolbox™ path planning functions and objects now support code generation:

- vehicleDimensions
- inflationCollisionChecker
- vehicleCostmap
- checkFree
- checkOccupied
- getCosts
- setCosts
- pathPlannerRRT
- plan
- driving.Path
- interpolate
- driving.DubinsPathSegment
- driving.ReedsSheppPathSegment
- checkPathValidity
- smoothPathSpline

Code generation for more Communications Toolbox functions

- algintrlv

- `algdeintrlv`
- `comm.DecisionFeedbackEqualizer`
- `comm.DPD`
- `comm.DPDCoefficientEstimator`
- `comm.LinearEqualizer`
- `intrlv`
- `deintrlv`
- `matintrlv`
- `matdeintrlv`
- `helscanintrlv`
- `helscandintrlv`
- `pammod`
- `genqammod`
- `pamdemod`

Code generation for more Computer Vision Toolbox functions

- `pointCloud`
- `findNearestNeighbors`
- `findNeighborsInRadius`
- `findPointsInROI`
- `removeInvalidPoints`
- `select`
- `pcdownsample`
- `pcfitcylinder`
- `pcfitplane`
- `pcfitsphere`
- `pctransform`
- `pcregistercpd`
- `pcmerge`
- `pcdenoise`
- `pcnormals`
- `pcsegdist`
- `segmentLidarData`
- `detectORBFeatures`
- `disparityBM`
- `disparitySGM`
- `opticalFlow`
- `estimateFlow`
- `reset`

Code generation for more DSP System Toolbox functions

- `dsp.STFT`
- `dsp.ISTFT`
- `dsp.FourthOrderSectionFilter`

Code generation for more Image Processing Toolbox functions

- `affine3d`
- `inpaintCoherent`
- `rgb2lightness`

Code generation for more Phased Array System Toolbox functions

- `azelcut2pat`
- `clone`
- `move`
- `rcscylinder`
- `rscdisc`
- `rcstruncone`
- `release`
- `reflect`
- `reset`
- `rotpat`

Code generation for more Robotics System Toolbox functions

- `robotics.LidarSLAM` class and methods `addScan`, `copy`, `removeLoopClosures`, and `scansAndPoses`
- `robotics.PoseGraph` object, `robotics.PoseGraph3D` object and functions `addRelativePose`, `edges`, `edgeConstraints`, and `findEdgeID`, `nodes`, and `removeEdges`
- `optimizePoseGraph`
- `robotics.DubinsConnection` object and `connect` function
- `robotics.DubinsPathSegment` object and `interpolate` function
- `robotics.ReedsSheppConnection` object
- `robotics.ReedsSheppPathSegment` object
- `uavOrbitFollower` System object
- `bsplinepolytraj`
- `cubicpolytraj`
- `quinticpolytraj`
- `rottraj`
- `transformtraj`

- trapveltraj

Code generation for more Sensor Fusion and Tracking Toolbox

- ahrs10filter
- append of ggiwphd
- AsyncMARGGPSFuser
- altimeterSensor
- clone of ggiwphd
- correct of ahrs10filter
- correct of AsyncMARGGPSFuser
- correct of ErrorStateIMUGPSFuser
- correct of ggiwphd
- correctjpda
- correctUndetected of ggiwphd
- ErrorStateIMUGPSFuser
- extractState of ggiwphd
- fuseaccel of AsyncMARGGPSFuser
- fusealtimeter of ahrs10filter
- fusegps of AsyncMARGGPSFuser
- fusegps of ErrorStateIMUGPSFuser
- fusegyro of AsyncMARGGPSFuser
- fusemag of ahrs10filter
- fusemag of AsyncMARGGPSFuser
- fusemvo of ErrorStateIMUGPSFuser
- jpdaEvents
- extractState of ggiwphd
- ggiwphd
- initcaggiwphd
- initctggiwphd
- initcvggiwphd
- labeledDensity of ggiwphd
- likelihood of ggiwphd
- merge of ggiwphd
- partitionDetections
- pose of ahrs10filter
- pose of AsyncMARGGPSFuser
- pose of ErrorStateIMUGPSFuser
- predict of ahrs10filter
- predict of AsyncMARGGPSFuser

- predict of ErrorStateIMUGPSFuser
- predict of ggiwphd
- prune of ggiwphd
- randrot
- reset of ahrl0filter
- reset of AsyncMARGGPSFuser
- reset of ErrorStateIMUGPSFuser
- scale of ggiwphd
- stateinfo of ahrl0filter
- stateinfo of AsyncMARGGPSFuser
- stateinfo of ErrorStateIMUGPSFuser
- trackerJPDA
- trackerPHD
- trackingSensorConfiguration

Code Generation for SerDes Toolbox Functions

- serdes.AGC
- serdes.CDR
- serdes.CTLE
- serdes.DFECDR
- serdes.FFE
- serdes.PassThrough
- serdes.SaturatingAmplifier
- serdes.VGA

Code generation for more Signal Processing Toolbox functions

The following Signal Processing Toolbox functions now support C/C++ code generation:

- **Filter Design and Filtering:**

buttapp, filtfilt, filtord, fir1, fir1s, kaiserord, and sos2tf

- **Spectral Analysis:**

cpsd, czt, goertzel, mscohere, periodogram, and pwelch

- **Spectral Windows:**

barthannwin, bartlett, blackman, blackmanharris, bohmanwin, flattopwin, gausswin, nuttallwin, parzenwin, rectwin, taylorwin, triang, and tukeywin now accept variable input.

- **Waveform Generation:**

gauspuls, pulstran, rectpuls, square, and tripuls

- **Linear Predictive Coding:**

lsf2poly, poly2ac, poly2lsf, poly2rc, rc2ac, rc2poly, and rlevinson

Code generation for more Statistics and Machine Learning Toolbox functions

- You can generate C/C++ code that predicts responses by using trained naive Bayes models.
 - `predict` — Classify observations, estimate posterior probabilities, or compute misclassification costs by applying a naive Bayes classification model to new data.
- You can specify a custom binary loss function for the `predict` function of the `CompactClassificationECOC`.
- The following functions support code generation:
 - `ksdensity` — Find kernel smoothing function estimates for univariate and bivariate data.
 - `mvksdensity` — Find kernel smoothing function estimates for multivariate data.
 - `ecdf` — Estimate empirical cumulative distribution function values.

Code generation for more Wavelet Toolbox functions

- `cwtfilterbank`
- `wt`
- `cwtfreqbounds`

Code generation for more WLAN Toolbox functions

MAC Frame Parser

- `wlanAMPDUDeaggregate`
- `wlanMPDUDecode`
- `displayIEs` of `wlanMACManagementConfig`

IEEE 802.11ax Signal Recovery

- `wlanHERecoveryConfig`
- `getSIGBLength` of `wlanHERecoveryConfig`
- `interpretHESIGABits` of `wlanHERecoveryConfig`
- `wlanHESIGABitRecover`
- `wlanLSIGBitRecover`

OFDM Demodulation and Information

- `wlanHEDemodulate`
- `wlanDMGOFDMDemodulate`
- `wlanSIGDemodulate`
- `wlanHEOFDMInfo`

- wlanVHTOFDMInfo
- wlanDMGOFDMInfo
- wlanHTOFDMInfo
- wlanNonHTOFDMInfo
- wlanSIGOFDMInfo

Reference Symbols

- wlanReferenceSymbols
- wlanClosestReferenceSymbol

802.11ay Channel Model

- wlanTGayChannel
- info of wlanTGayChannel
- showEnvironment of wlanTGayChannel
- wlanURAConfig

S-Functions

Thread-safe error status handling

From R2019a, you can use these functions for thread-safe handling of errors:

- `ssSetLocalErrorStatus` — Report an error
- `ssGetLocalErrorStatus` — Identify the last error

SimState S-Function APIs renamed

The `SimState` object has been renamed to `ModelOperatingPoint` object. The S-Function API have been updated to reflect this name change.

Version History

`SimState` in the S-Function API has been renamed to `OperatingPoint`. The old API continues to work, however, starting in R2019a, it is recommended to use the new API.

- C-MEX S-Function

Old	New
<code>ssSetSimStateCompliance</code>	<code>ssSetOperatingPointCompliance</code>
<code>ssSetSimStateVisibility</code>	<code>ssSetOperatingPointVisibility</code>
<code>mdlGetSimState</code>	<code>mdlGetOperatingPoint</code>
<code>mdlSetSimState</code>	<code>mdlGetOperatingPoint</code>
<code>MDL_SIM_STATE</code>	<code>MDL_OPERATING_POINT</code>

- MATLAB S-Function

Old	New
<code>GetSimState</code>	<code>GetOperatingPoint</code>
<code>SetSimState</code>	<code>SetOperatingPoint</code>

Modeling Guidelines

High-Integrity System Modeling guidelines to verify compliance with DO, IEC, EN, and ISO safety standards

This table identifies removed and modified modeling guidelines. For a complete list of High-Integrity System Modeling guidelines, including their applicable Model Advisor checks, see Model Advisor Checks for High-Integrity Modeling Guidelines.

High-Integrity Modeling Guideline	Rationale
hisl_0005: Usage of Product blocks	Removed condition "Set the model configuration parameter Diagnostics > Data Validity > Signals > Division by singular matrix to error". The condition is covered in hisl_0314: Configuration Parameters > Diagnostics > Data Validity > Signals.
hisl_0019: Usage of bitwise operations	<p>Guideline expanded to include the following blocks:</p> <ul style="list-style-type: none"> • Bit Clear • Bit Set • Bitwise Operator • Extract Bits • Shift Arithmetic <p>These blocks are checked in Model Advisor check Check for bitwise operations in Simulink blocks.</p>
hisl_0021: Consistent vector indexing method	<p>Enhanced condition "Consistent vector indexing" to include a list of blocks/functions that support:</p> <ul style="list-style-type: none"> • Configurable conditioning • One-based indexing • Zero-based indexing <p>These blocks/functions are checked in Model Advisor check Check for inconsistent vector indexing methods.</p>
hisl_0060: Configuration parameters that improve MISRA C:2012 compliance	<p>Added configuration parameters Include Comments and MATLAB user comments to the guideline.</p> <p>These parameter settings are checked in Model Advisor check Check configuration parameters for MISRA C:2012.</p>

R2018b

Version: 9.2

New Features

Bug Fixes

Version History

Simulation Analysis and Performance

Batch Simulations: Offload execution of simulations to run in the background by using batchsim

`batchsim` allows you to run batch simulations, in serial or in parallel. To run batch simulations in parallel, specify the size of the parallel pool. `batchsim` offloads simulations to the compute cluster, giving you the ability to carry out other tasks while the batch job is processing, or close the client MATLAB and access the batch job later.

Execution Domain Specification: Improve your solver performance and code by specifying the domains of subsystems within your model

To improve solver performance and execution, you can constrain a subsystem to be in the discrete time domain. Domain specification lets you increase reusability of subsystems as components and increase reusability of the generated code. See Specify Execution Domain for more information.

Custom Gauge Block: Create a gauge with a fully customizable appearance


Using the Custom Gauge block, you can create a gauge with a fully customized appearance that represents the connected signal as it would appear on a real dashboard.

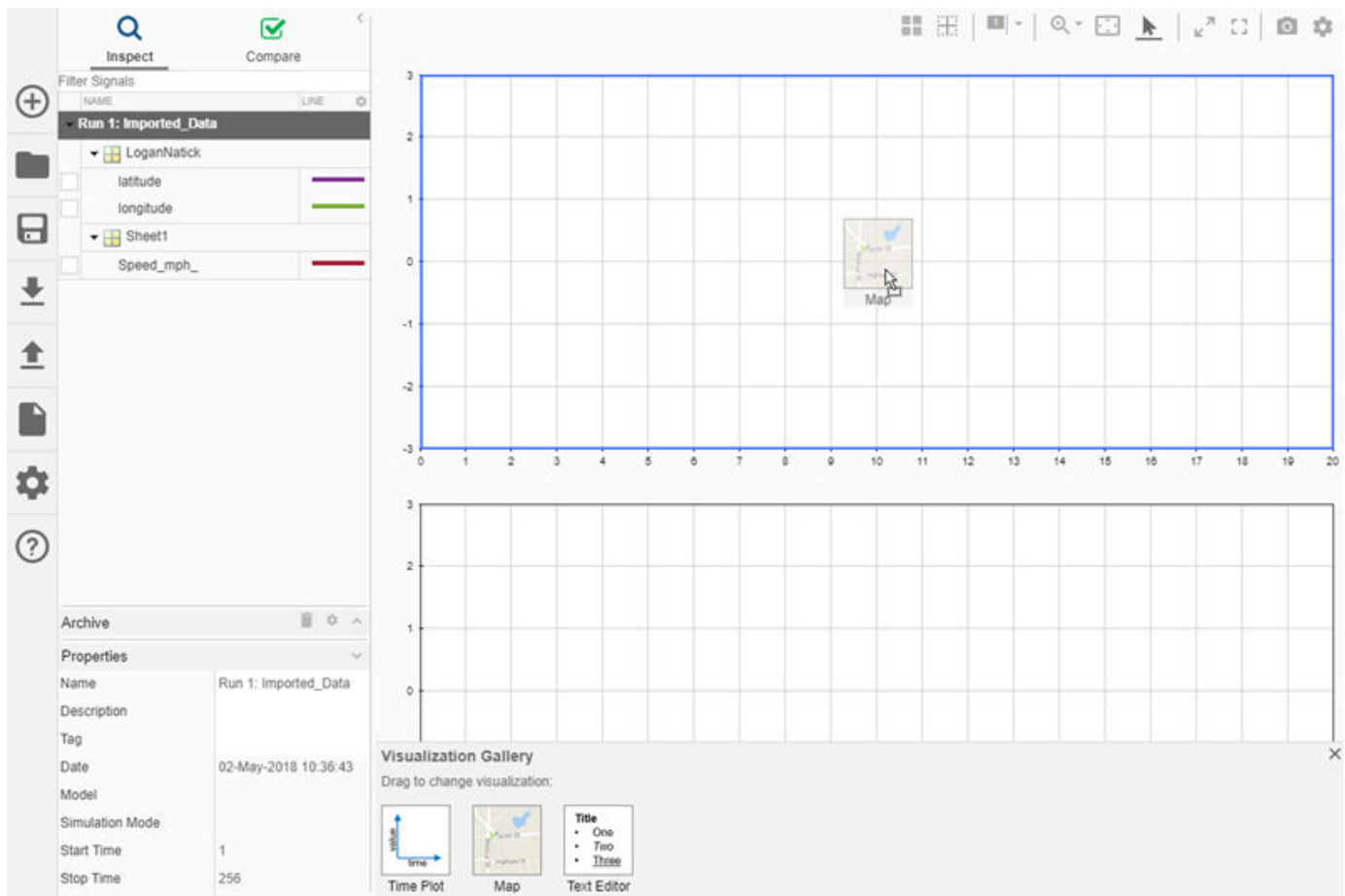


In addition to the customizable appearance, the Custom Gauge block has a set of parameters similar to the other gauge Dashboard blocks, including options to set the value range and tick mark interval.

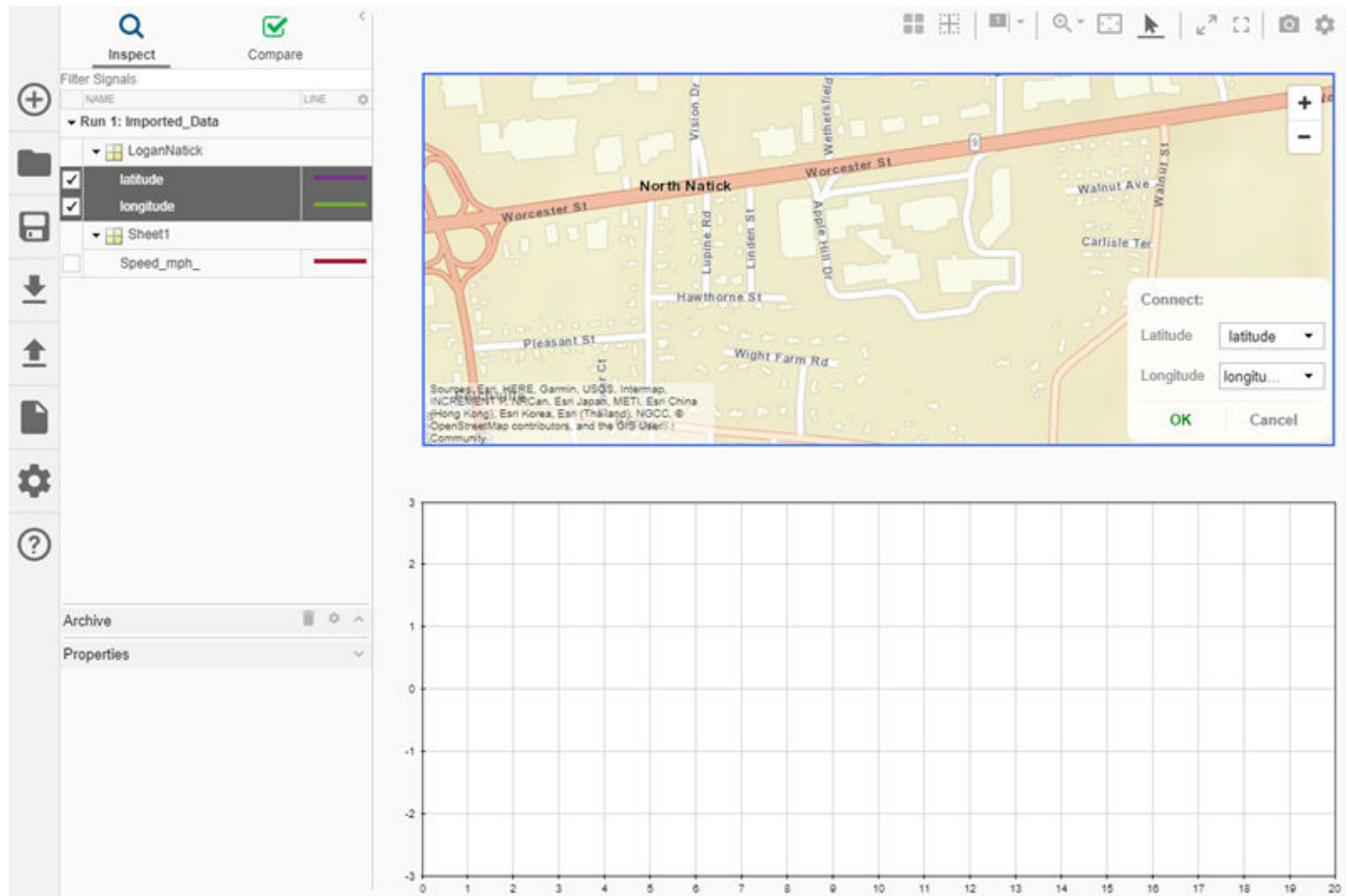
Maps in the Simulation Data Inspector: View map data that is synchronized with signal data in the Simulation Data Inspector

The Simulation Data Inspector **Visualization Gallery** includes a map visualization. You can plot longitude and latitude data to view a route. When you have signal data to view with your map data, the Simulation Data Inspector synchronizes the map data and signal data so you can inspect sensor signals along the route.

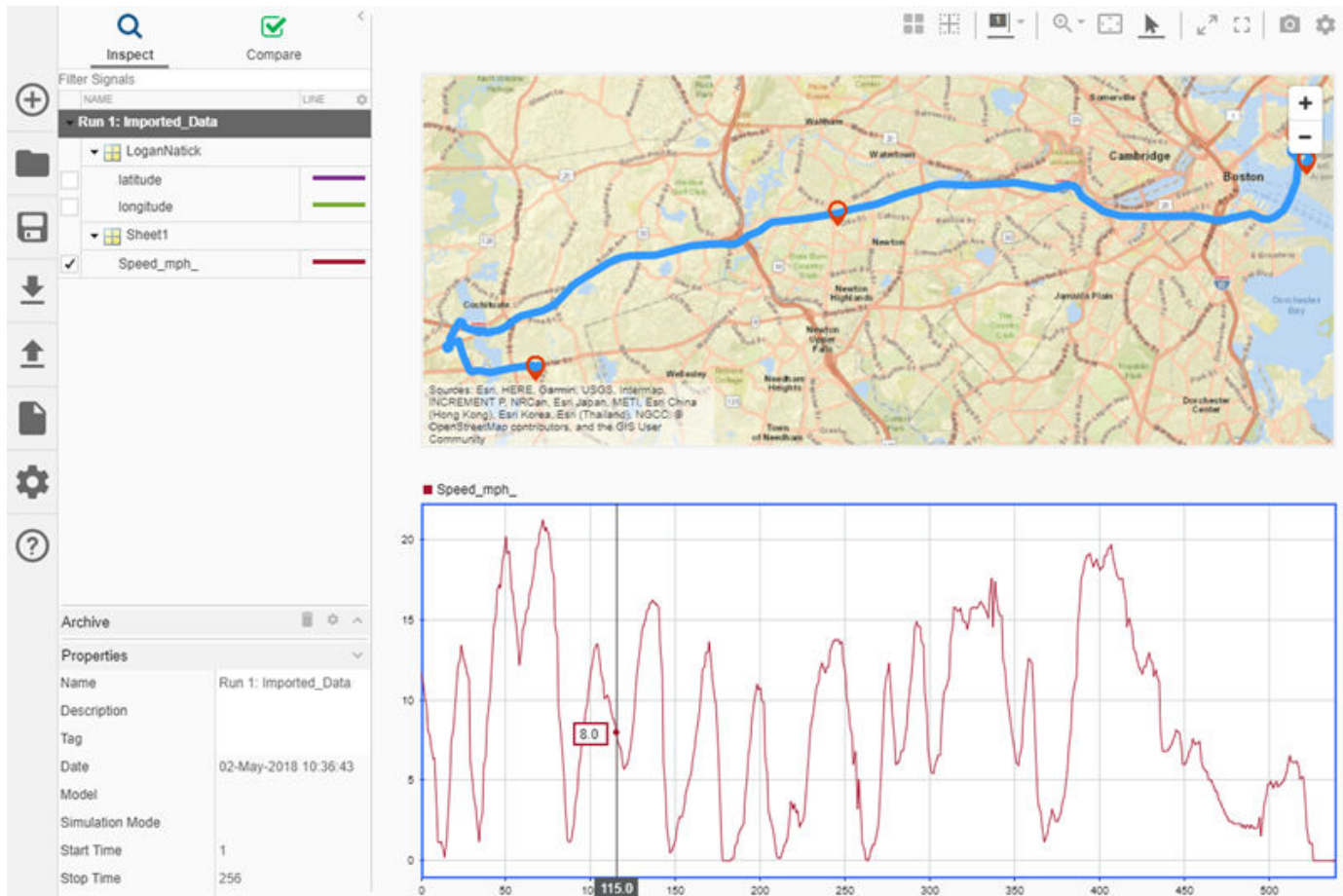
To add a map to your plot layout, drag and drop the map option from the **Visualization Gallery** onto the plot where you want the map. Open the **Visualization Gallery** from the **Layout** menu .



To specify longitude and latitude data for your map, drag and drop your signals onto the map and specify which signal corresponds to latitude and which to longitude.



To inspect the temporal characteristics of your map data, plot a related time-dependent signal and add a cursor. As you move the cursor, a marker on the map shows the corresponding location along the route for that time.



Descriptor State-Space Block: Author linear time invariant DAE equations directly

You can easily create continuous-time implicit state-space models using the new Descriptor State-Space block.

The behavior of the system is described by

$$E\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

where E is the mass matrix of the system. Descriptor State-Space systems are a state-space representation of differential algebraic equations. For more information on DAEs, see Model Differential Algebraic Equations.

FMU Export: Package a model as an FMU for co-simulation

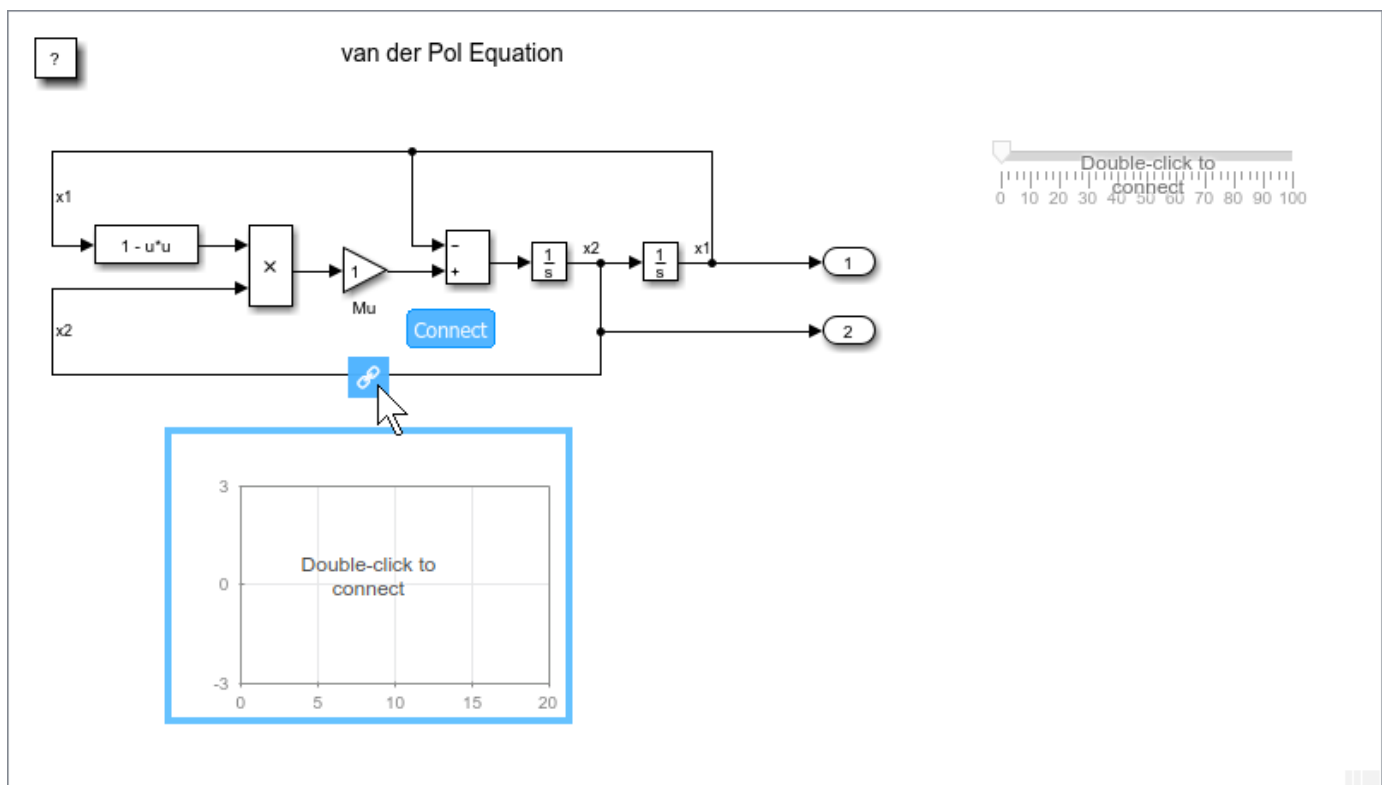
Wrap your Simulink model and project files into a Functional Mock-Up Unit (FMU) with a Co-Simulation 2.0 interface. When a third-party tool runs the FMU, it checks out required licenses and starts a local installation of Simulink to start the model. This function requires the Tool-Coupling Co-

Simulation FMU Export for Simulink support package. To download and install the support package, use the Add-On Explorer.

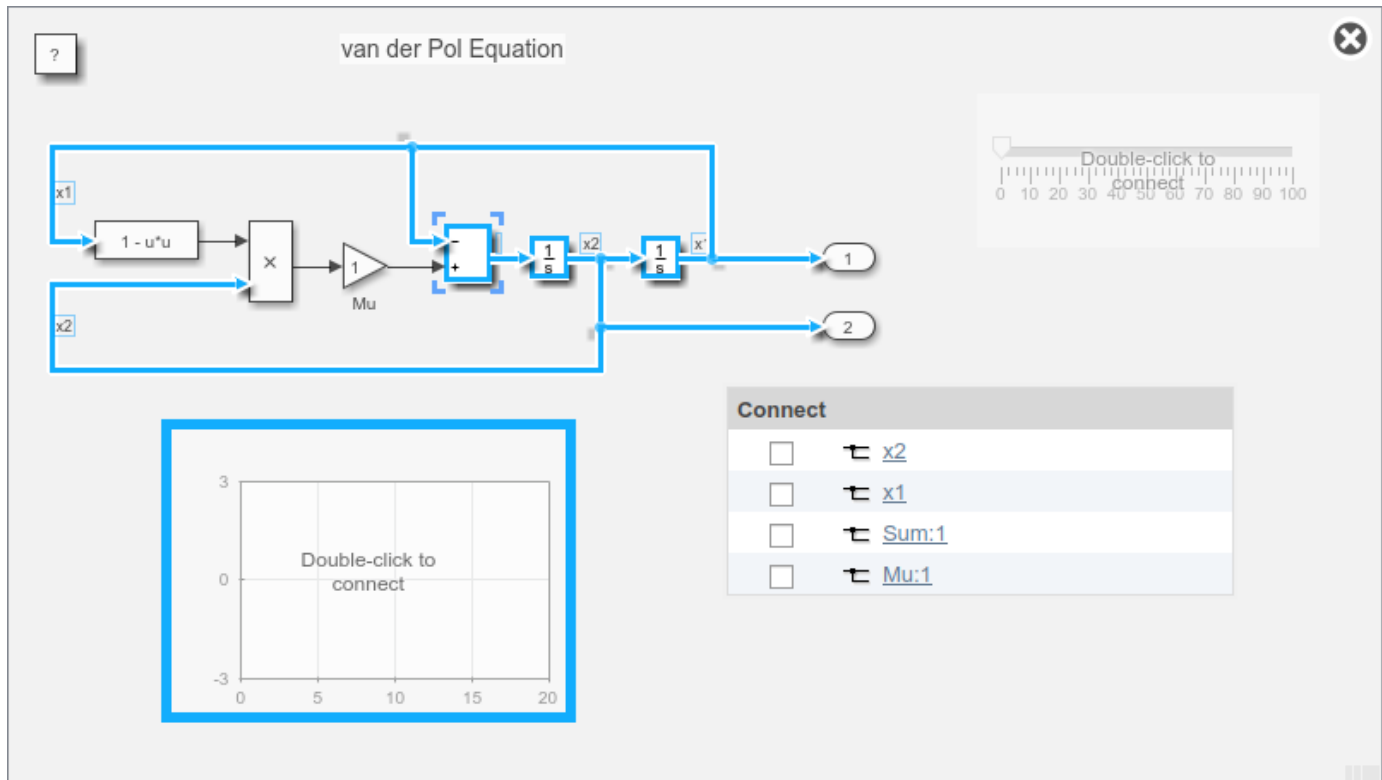
Easily connect multiple Dashboard blocks to signals in your model with connect mode

A new connect mode facilitates connecting multiple Dashboard blocks to signals in your model. While in connect mode, the process to connect a Dashboard block to a signal requires fewer clicks. After you have connected your signals of interest, you can exit connect mode and continue building your model or run a simulation.

To enter connect mode, click the **Connect** button that appears above your unconnected Dashboard block when you pause on it.




In connect mode, when you select one or more signals or blocks a list of available parameters and signals available for connection appears. Select a signal or parameter from the list to connect the selected Dashboard block. Then, you can select the next Dashboard block you want to connect, make a selection of signals and blocks in your model, and choose a signal or parameter to connect.

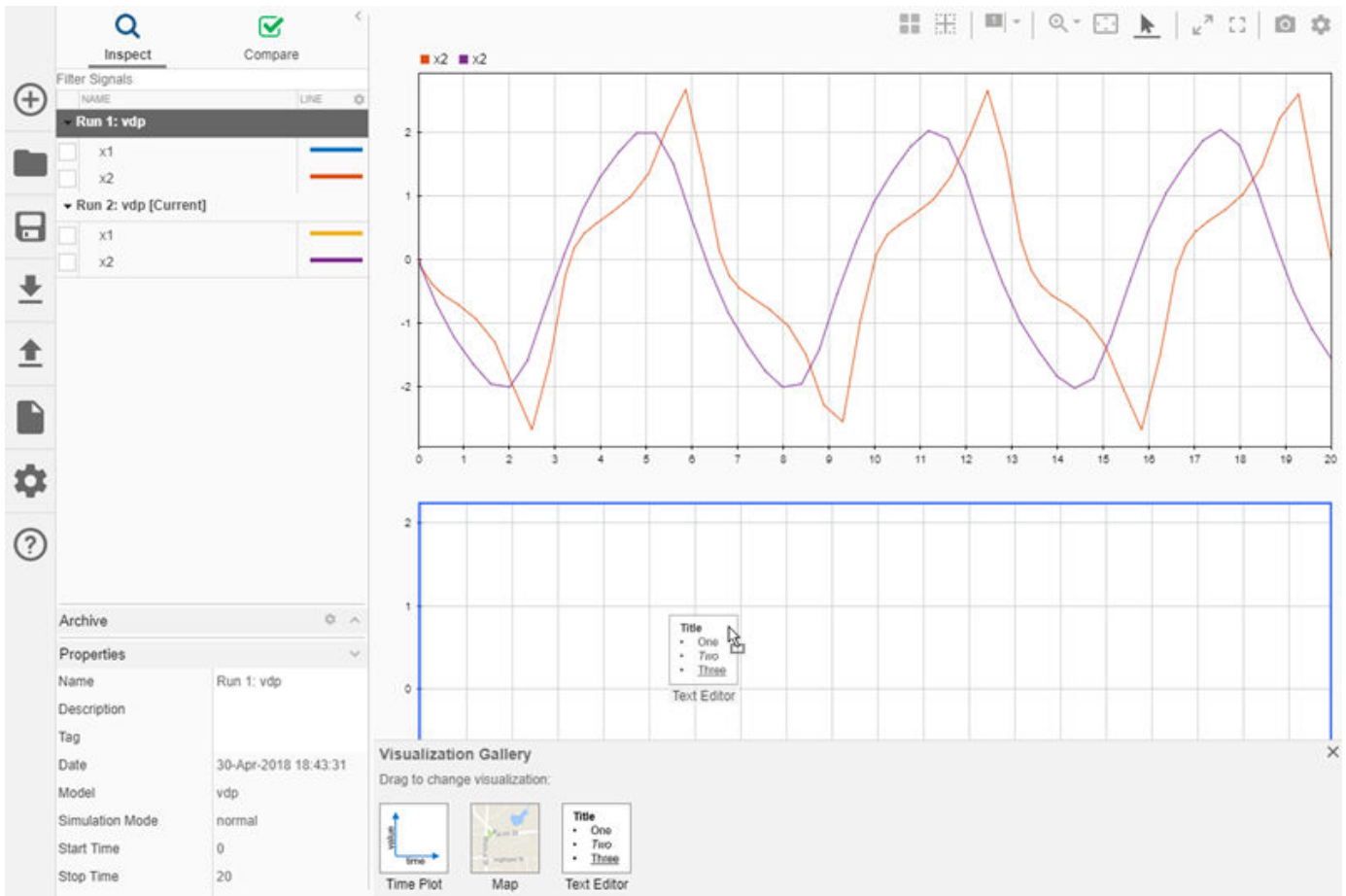


When you finish connecting the Dashboard blocks in your model, click the **Exit** button in the upper-right of the canvas to exit connect mode.

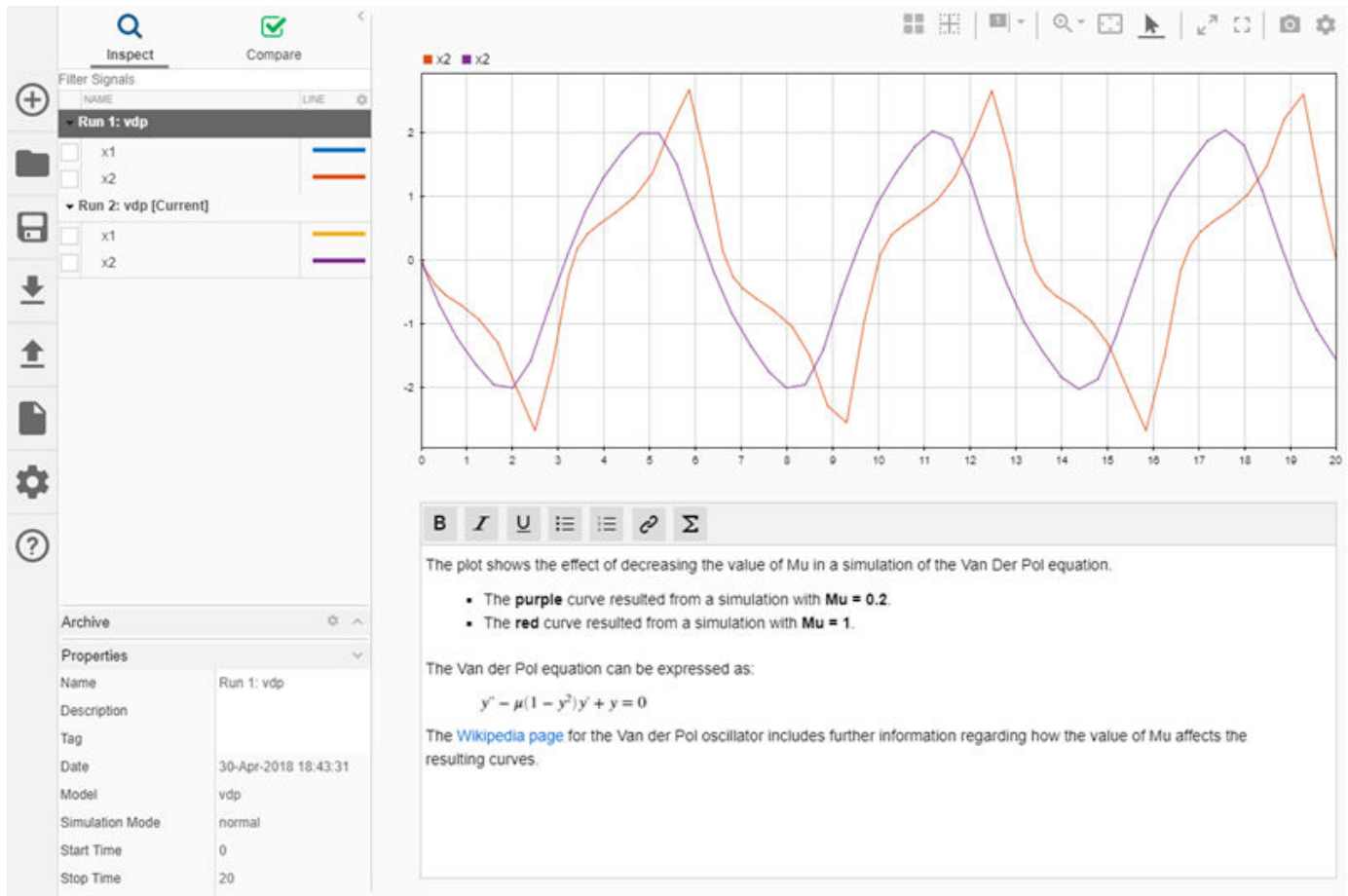
Include notes and analysis in visualizations using a text editor in the Simulation Data Inspector

The **Visualization Gallery** added in R2018b includes a text editor that you can drag and drop into your layout to add notes, analysis, and significant findings to sessions, views, and reports created with the Simulation Data Inspector. The text editor has formatting options, including hyperlinks and equations formatted with LaTeX.

You can access the **Visualization Gallery** from the **Layout** menu . To add a text editor to your layout, drag and drop the text editor option from the **Visualization Gallery** to the subplot location where you want to include your notes.

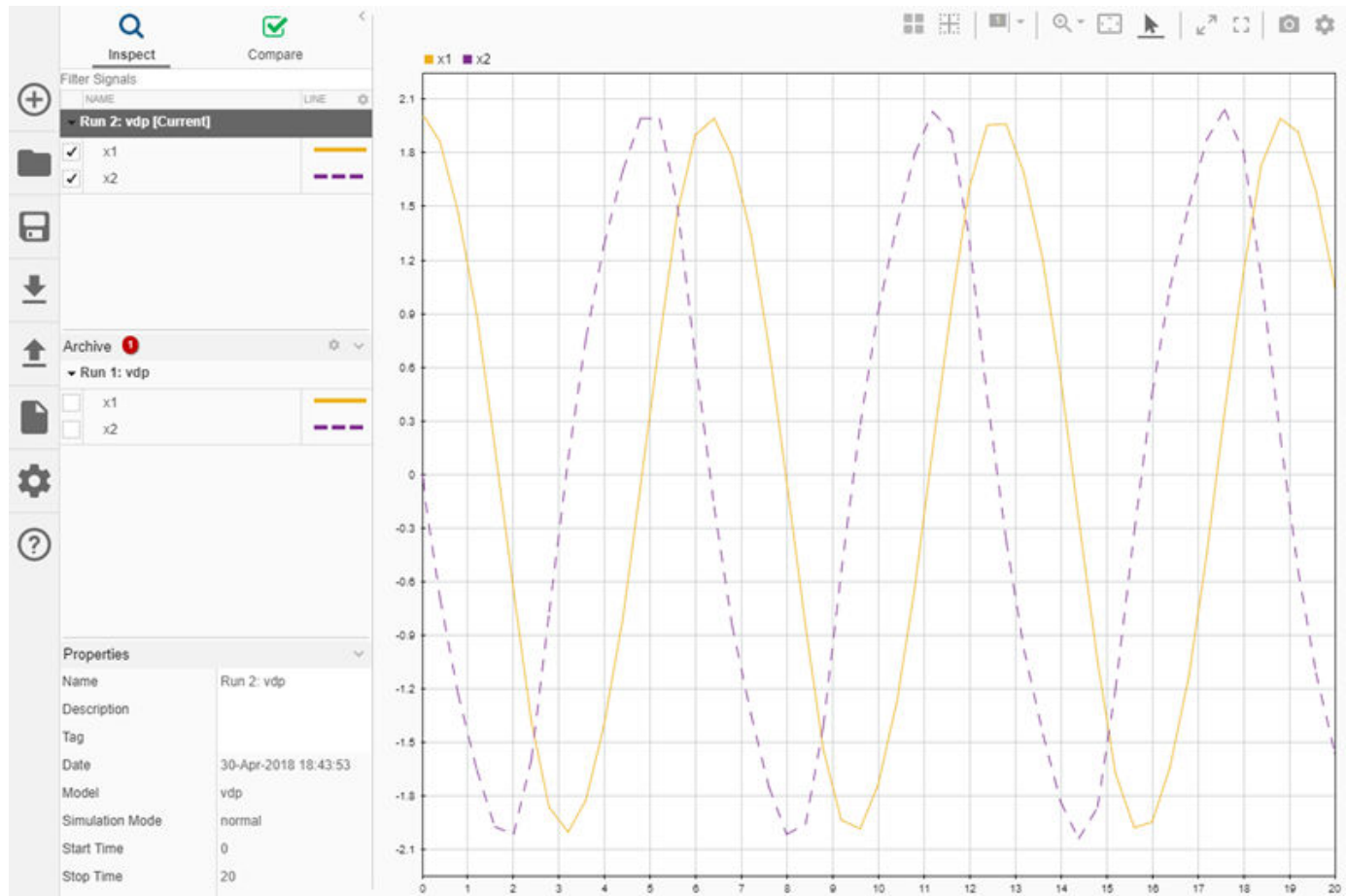


Then, you can add meaningful notes, context, and equations to the text editor.



Iterate model design with the Simulation Data Inspector archive

The archive in the Simulation Data Inspector provides a place to store runs outside the work area so your session can remain focused. By default, the Simulation Data Inspector auto-archives simulation runs. With auto-archive enabled, when you simulate a model, the Simulation Data Inspector moves the previous run from the work area to the archive. In the process of moving the run, the Simulation Data Inspector also transfers signal check box settings for your plots and signal line styles and colors so the plots automatically update to show the corresponding results for the new simulation run.



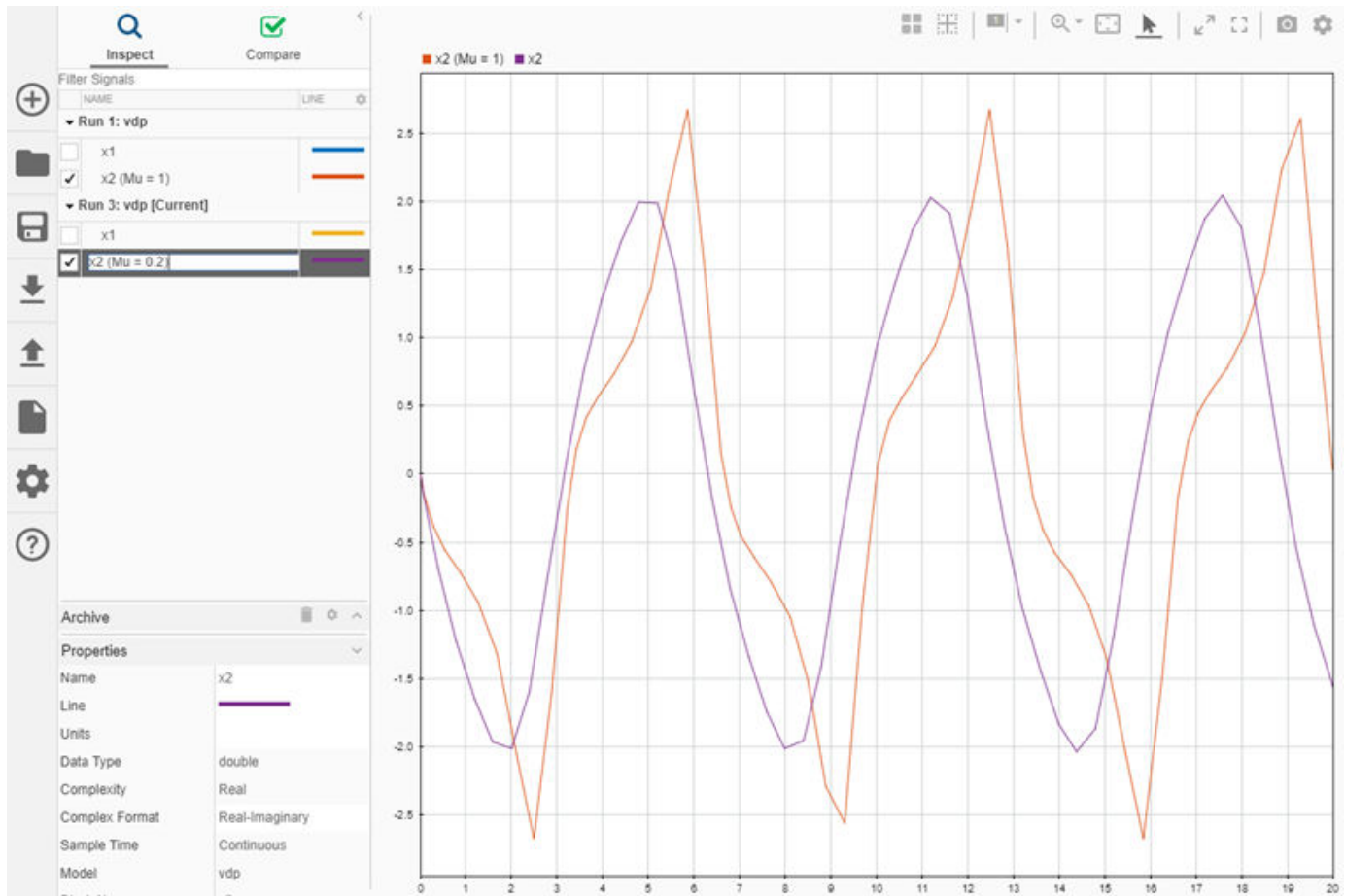
When you are interested in more than your most recent run, you can drag a run from the archive and drop it in the working area. You can plot signals from the archive and from the working area.

By default, the Simulation Data Inspector auto-archives without a limitation on the number of runs stored in the archive. To limit the use of disk space, you can configure a limit for the number of runs stored in the Simulation Data Inspector archive. You can also disable the auto-archiving behavior and manually manage where your runs are stored.

Rename signals in the Simulation Data Inspector

You can rename nonbus signals in the Simulation Data Inspector to identify a signal more clearly. For example, when you log a signal in your model without a name, the default name assigned by the Simulation Data Inspector can be nondescript, such as `Gain:1`.

The plot legends associate the signal name with the line color. Renaming signals in the Simulation Data Inspector can improve the legend, for example, when you plot the same signal from different runs.



Renaming a signal in the Simulation Data Inspector does not affect the source model or file for the data. The names you assign in the Simulation Data Inspector are saved in sessions and views and used in the base workspace variables and MAT-files created when you export data.

Import Excel files into the Simulation Data Inspector


You can import Microsoft Excel XLS and XLSX files into the Simulation Data Inspector to view model input data and visualize test data with simulation data.

Inspect data in the Simulation Data Inspector by viewing the window full screen and maximizing subplots

The new full screen button maximizes the Simulation Data Inspector window on your monitor and collapses the navigation pane, allowing the plotting area to occupy as much space as possible. When your layout contains multiple subplots, you can use the maximize button on the Simulation Data Inspector toolbar to maximize the active visualization within the plotting area for close inspection.

The full screen and maximize buttons are on the toolbar at the top of the Simulation Data Inspector window.



The maximize button is on the left, and the full screen button is on the right. When you activate either option, the button changes to indicate that the option is active. 

Click the full screen or maximize button a second time to return to regular viewing in the Simulation Data Inspector.

Dynamic Solver Switching: Adaptively switch solver during run-time to improve simulation performance

You can configure Simulink to change the solver during the simulation of your model. By default, Simulink automatically changes the solver according to the dynamics of your model. You can also specify parameter change events and specific times in the simulation for solver switching.

Step through the referenced model with Simulink Debugger

Simulink Debugger now allows you to step inside a normal mode model reference. Stepping inside a model reference enables you to display information on block states, block inputs and outputs, and block method execution. You can now pinpoint issues in your model reference to specific blocks, parameters, or interconnections.

Diagnostic Viewer: Register custom compiler to report errors and warning

Diagnostic Viewer can recognize errors and warnings from builds generated by custom compilers. You can specify compiler-specific patterns to make Diagnostic Viewer recognize them.

```
tool.setDirective ('WarningPattern','warning #(\d+):'); //Specifies warning patterns
tool.setDirective ('ErrorPattern','error:'); //Specifies error pattern
tool.setDirective ('FileNamePattern','[^\s]*\w+\.(\c|h)'); //Specifies file name pattern
tool.setDirective ('LineNumberPattern','\(\d+\)'); //Specifies line number pattern
```

For more information, see Register Custom Toolchain and Build Executable (Simulink Coder).

Functionality being removed or changed

Simulink.sdi.setRunOverwrite function has been removed

Errors

The `Simulink.sdi.setRunOverwrite` function is removed in R2018b with the introduction of the Simulation Data Inspector archive. You can configure the Overwrite Run mode behavior using the new `Simulink.sdi.setAutoArchiveMode` and `Simulink.sdi.setArchiveRunLimit` functions. Enable the Simulation Data Inspector to automatically archive simulation runs, and set the archive limit to 0.

```
Simulink.sdi.setAutoArchiveMode(true);
Simulink.sdi.setArchiveRunLimit(0);
```

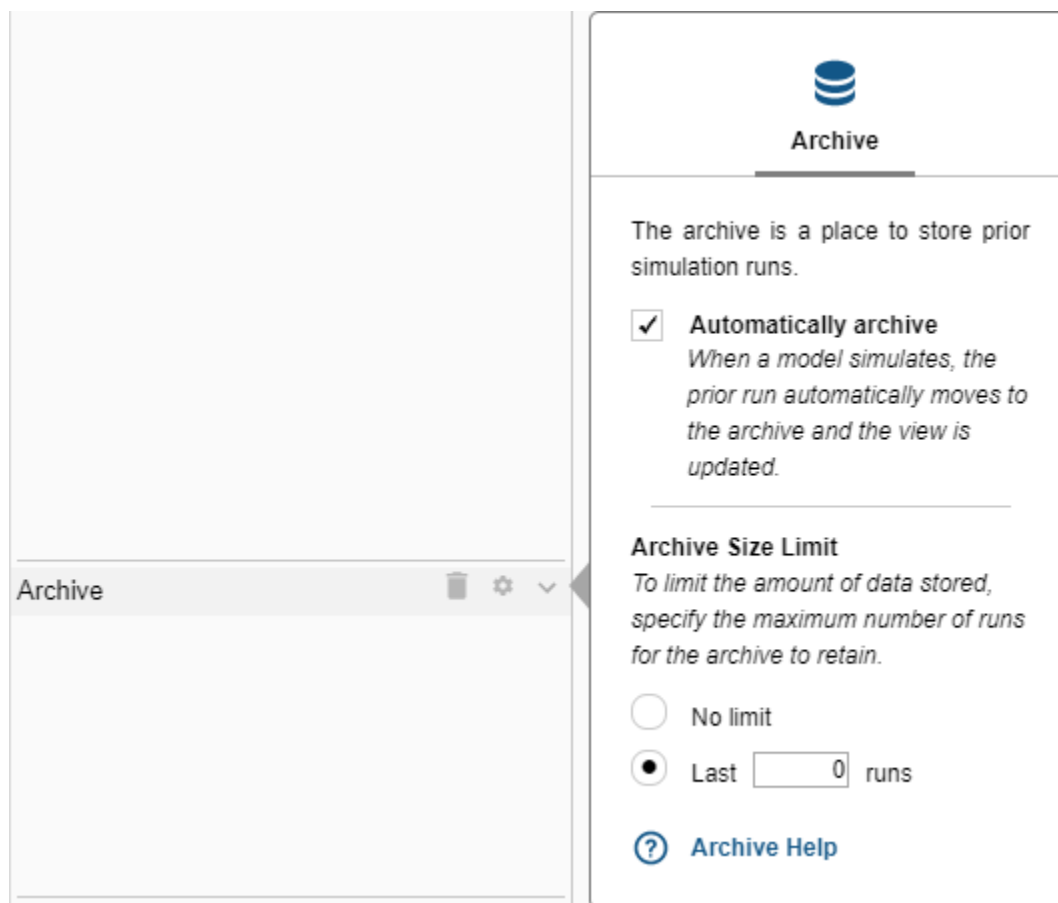
With these settings, the Simulation Data Inspector retains the data for the current simulation run and discards the data for the previous simulation run. The view in the Simulation Data Inspector also updates to show signals in the current run, with consistent line styles and colors.

You can disable each aspect of the behavior separately. To retain prior run data, set the archive limit to a nonzero value. If you want no limit on the number of runs stored in the archive, you can specify -1 as the archive limit. When you disable the **Automatically archive** setting, the Simulation Data Inspector does not move the prior run to the archive or update the view to show signals from the current run.

The Simulation Data Inspector archive replaces Overwrite Run functionality

Behavior change

Overwrite Run mode is removed from the Simulation Data Inspector in R2018b with the introduction of the Simulation Data Inspector archive. You can configure the Simulation Data Inspector to behave as it would in Overwrite Run mode using the archive settings. Enable **Automatically archive** and set the **Archive Size Limit** to 0 on the archive settings pane.



With these settings, the Simulation Data Inspector retains the data for the current simulation run and discards the data for the previous simulation run. The view in the Simulation Data Inspector also updates to show signals in the current run, with consistent line styles and colors.

You can configure the Simulation Data Inspector archive settings to update the view with data from the current simulation and retain the data for previous simulations. Specify a nonzero run limit or select **No limit** for the **Archive Size Limit** and leave **Automatically archive** checked. With the **Automatically archive** setting enabled, the Simulation Data Inspector moves the previous simulation run to the archive and updates the view to show signals from the current simulation run. For more information about using the Simulation Data Inspector archive, see:

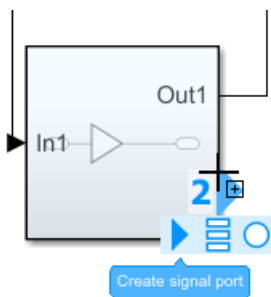
- Iterate Model Design Using the Simulation Data Inspector
- The Simulation Data Inspector Archive

Simulink Editor

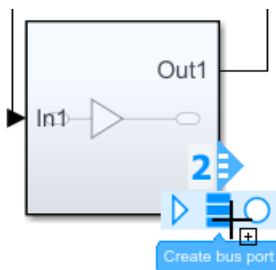
Automatic Port Creation: Create a port by clicking or dragging the block outline

In R2018b, you can add ports by clicking where you want to add the port. After you click the port, you can select the type of port you want to create: signal, bus port, or connection.

Click to add the port. The new port number appears in blue. When you place your cursor over the new port, a menu appears. Select the type of port you want to create. Only the types of ports that you can create in that context appear as options.



Move the cursor over the menu to change the port type that you want to create, and then drag to add the signal or connection.



For more information on interactive connections, see [Branch a Connection](#).

Predictive Quick Insert: Connect a recommended block to an existing block in your model, sorted by frequency of use

As you build your model, you can double-click the end of new signal lines to search for and select a block from a menu to connect to the signal. In R2018b, the menu suggests blocks to insert based on context. For example, if you start to draw a signal from a Gain block and double-click the end of the new signal line, the menu includes math blocks such as Sum. You can select a listed block to insert or start typing to search for and insert any block.

Simulink comes with default suggestions. To learn about enhancing the list of suggested blocks based on your model designs, see [Improve Quick Block Insert Results](#).

Edit on Block Icon: Modify block parameters without opening a dialog box

You can now edit block parameter values on the block icon in the Simulink Editor. If you hover on a selected block, the editable parameter value appears underlined.



Click the underlined text to edit the value. Enter the text and press **Enter** or click away. To cancel the edit, press **Esc**.

Better context awareness for SimEvents blocks

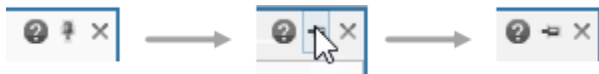
While working with SimEvents, the quick block insert search results are based on SimEvents context.

Quick Block Insert: Dynamically sort quick block insert search results

You can get the quick block insert search results ranked dynamically based on your recent block usage history. For more information, see [Branch a Connection](#).

Collapsible Panels: Manage UI elements by hiding panels

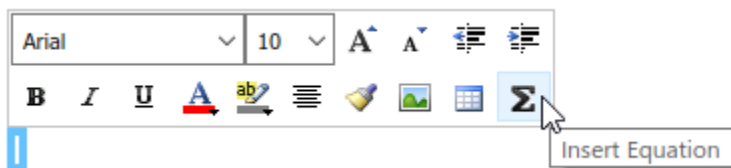
You can control whether panels such as the Model Data Editor and Property Inspector are persistent or collapsed in the Simulink Editor. By default, panels are persistent when opened. To allow a panel to collapse when inactive, click the pushpin.



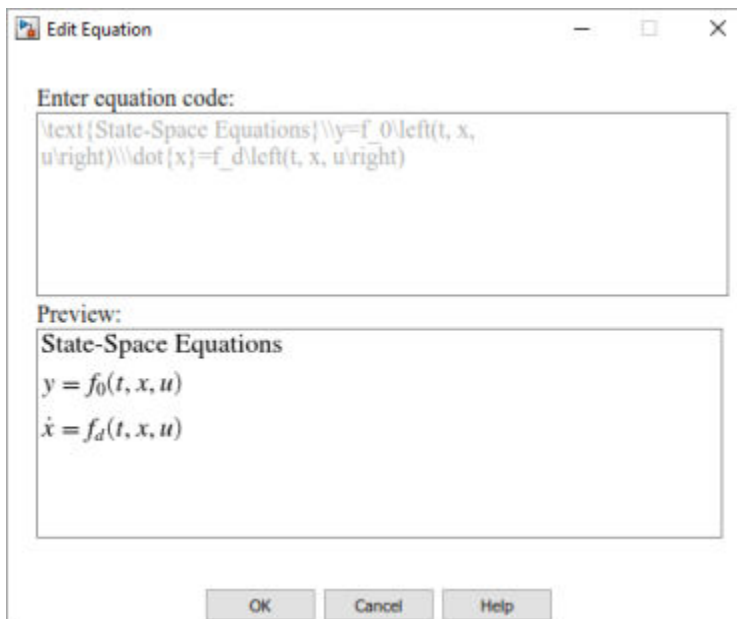
To access a collapsed panel, click the corresponding tab. To make a panel persistent again, double-click its tab or click the pushpin button.

Equations in Annotations: Insert LaTeX and MathML equations in your annotations

You can now add equations to your annotation by clicking the Insert Equation button in the annotation formatting toolbar.



In the Edit Equation dialog box, enter LaTeX or MathML code to generate equations.



To edit equation code, double-click the equation in the annotation. Similar to text in annotations, you can format equations in annotations by using the formatting toolbar. You can change the font color, text highlight color, font size, and location of equations in annotations.

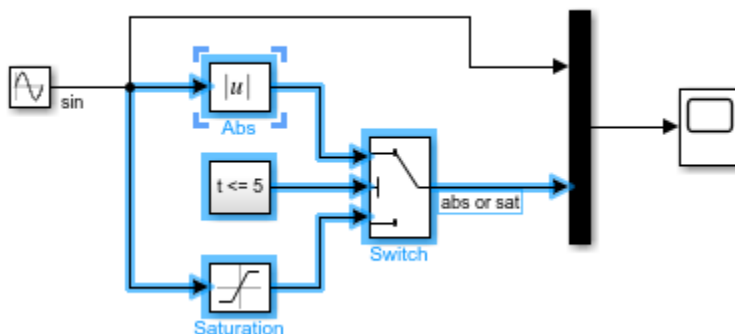
Group Block Rotation: Rotate or flip selected blocks as a group

In R2018b, you can rotate a selection of blocks in a model in 90-degree increments as a group. You can also flip a selection of blocks from left to right. Rotating as a group helps you to reorient blocks for better readability or better use of the model canvas.

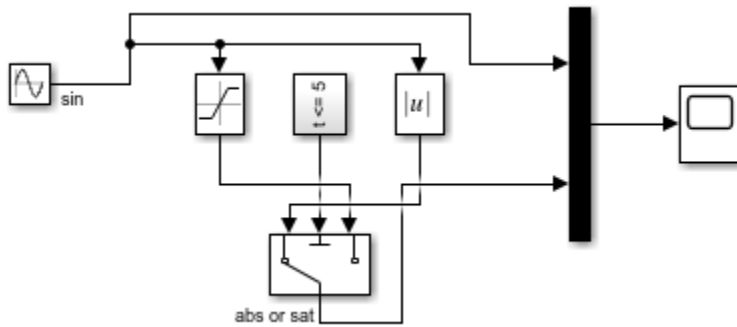
In previous releases, selecting more than one block rotated or flipped each block.

To rotate the group or flip it vertically, select the blocks and then select a command from the **Diagram > Rotate & Flip** menu.

For example, rotate the selected blocks in this model clockwise.



The blocks rotate together, maintaining their relationship to each other.



The rotation or flip can cause signal lines to have more bends than in the original model. You can improve the readability of the model by editing the signal lines.

For more information, see [Flip or Rotate Blocks](#)

Version History

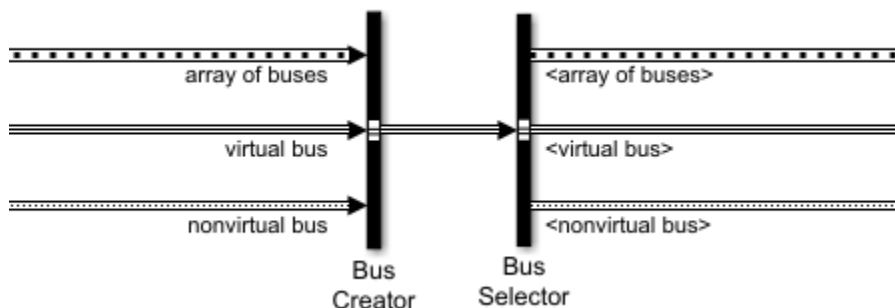
In previous releases, with multiple blocks selected each block rotated or flipped. In R2018b, to rotate or flip multiple blocks, select and rotate or flip each block.

Component-Based Modeling

Buses: Create buses from Simscape physical connection lines and retain arrays of buses and nonvirtual buses within virtual buses

For Simscape users, the new Simscape Bus block in the Utilities library lets you bundle conserving connections into a Simscape Bus line. You can also use this block to access one or more connections from an existing Simscape Bus line. Simscape Bus lines cannot be used with Bus Creator, Bus Selector, or Bus Assignment blocks.

For Simulink users, virtual buses now support arrays of buses. Nonvirtual buses and arrays of buses now retain their signal type inside virtual buses.



Version History

If a model relies on having a nested nonvirtual bus become virtual, you must update the model. To retain the old behavior, you can:

- Convert a selected nonvirtual bus to a virtual bus. For example, to provide a vector signal input to a Gain block, you can insert a Signal Conversion block between the Bus Selector block and Gain block.
- Update the block that receives a nested nonvirtual bus to accept nonvirtual buses. For example, the Outport block requires a vector signal input when the **Initial output** parameter specifies a nonzero scalar value. You can change **Initial output** to specify zero, [], or a finite numeric MATLAB structure.

Simulink Variants: Develop versatile variant models with improved support for buses, IRT blocks, and functions

From R2018b, Simulink Variants contains these improvements:

- Variant signal within buses with Composite ports — Simulink Variant models now support Composite ports as inports and outports with buses. You can have different Variant conditions for each Composite port. For more information, see Variant Elements within Buses.
- Propagate conditions out of a Model block that has Reset Event ports — Simulink now supports propagating Variant conditions in Model blocks that have Reset Event ports. This results in optimized existence of blocks connected to the Reset Event ports. Models with inactive variant

Reset Event functions (in reference models) also support Variant condition propagation. For more information, see [Variant Condition Propagation with Initialize, Reset, and Terminate Blocks](#).

- Export function within Variant Subsystems — Simulink now supports using Export Function models inside Variant Subsystem blocks as Variant choices. You can also use multientry-point export functions for code generation. For more information, see [Variant Systems with Conditional Systems](#).
- Use `InitFcn` to define Variant Control Variables — You can use `InitFcn` callback to define variant control variables in Variant blocks and Variant Models. For more information on `InitFcn`, see [Initialization Function](#).

Simulink.Bus.createObject supports more object types

`Simulink.Bus.createObject` now accepts structures that contain MATLAB timetable and `matlab.io.datastore.SimulationDatastore` objects.

Mask Constraints: Create and share mask parameter constraints

You can use the Constraint Manager available in the Mask Editor to manage constraints. Constraints ensure that mask parameters are within a desired specification, removing the need for additional code. You can associate a constraint with **Edit** type mask parameters.

From R2018b, you can save a mask constraint in a MAT-file and share the constraint among different blocks across models or libraries. For more information, see [Add Constraints to Mask Parameters](#).

Mask Editor: Extend mask usability with additional parameters and options

From R2018b, you can use these parameters and options in the Mask Editor to enhance usability:

- Logarithmic Dial and Slider:- Before R2018b, you could add the Dial and Slider parameters in the mask dialog box and tune the parameter value in the linear scale. From R2018b, these values can also be tuned in logarithmic scale.
- Text Area:- You can now use the Text Area parameter to add custom text or MATLAB code in the mask dialog box.

For more information, see [Mask Editor Overview](#).

Label Mode: Use Label mode to select Variant controls for modeling Variant blocks

Before R2018b, you could use the **Override variant conditions and use following variant** parameter to select the active variant based on the value specified in the Variant parameter.

From R2018b, the `Label` mode allows you to choose name-based Variant controls (Label Mode Active choice) for modeling Variant Subsystem, Variant Source, and Variant Sink blocks. In `Label` mode, you do not need to create Variant controls in the global workspace. Label mode allows you to explicitly select variant control options. For more information, see [Introduction to Variant Controls](#).

Variant Reducer: Enhanced functionalities for better usability

From R2018b, you can use these enhanced options in the Variant Reducer:

- **Specify variant control values** — While reducing a model, use the **Specify variant control values** option to create multiple variable groups corresponding to different configurations. For more information, see [Reduce Models Containing Variant Blocks](#).
- **Specify variant configurations** — The **Specify variant configurations** option now displays values of variant controls variables for each configuration. For more information, see [Reduce Models Containing Variant Blocks](#).

Simulink Variants: Empty Simulink Variant objects not allowed in Simulink Variants

From R2018b, Simulink.Variant objects without a condition are not allowed in Variant blocks. For more information, see [Introduction to Variant Controls](#).

Compact Code Packaging for Variants: Generate lightweight code for Simulink Variants

By default, files like `model_private.h`, `model.c`, `model.h`, `model_types.h`, and so on, are generated during code generation of a Simulink Variant model.

You can choose to generate compact code by eliminating the generation of `model_private.h` and `model_types.h` files. For more information, see [Prepare Variant-Containing Model for Code Generation](#).

Functionality being removed or changed

Convert Model Blocks That Contain Model Variants to Variant Subsystem Blocks

Warns

In R2018b, the Block Parameters dialog box for Model blocks no longer contains an **Enable Variants** button. In a future release, Model blocks that contain variant models will convert to Variant Subsystem blocks that reference the variant models. Variant Subsystem blocks provide these advantages:

- Allow you to mix Model blocks and Subsystem blocks as variant systems
- Support flexible I/O, so that variants can have a different number of input and output ports

To convert a Model block that contains variant models to a Variant Subsystem block that contains Model blocks referencing the variant models, use one of these approaches:

- Right-click the Model block and select **Subsystems & Model Reference > Convert to > Variant Subsystem**.
- Use the `Simulink.VariantManager.convertToVariant` function. Specify the Model block path or block handle.

If you convert a Model block to a Variant Subsystem block, the Model block parameter **Generate preprocessor conditionals** behaves differently from the Variant Subsystem block parameter **Analyze all choices during update diagram and generate preprocessor conditionals**. For

Model blocks that contain model variants, enabling the parameter causes simulation and update diagram to compile the active variant only. For Variant Subsystem blocks, enabling the parameter compiles all the variants, which can make simulation and updates slower.

Converting model variants to subsystem variants can require you to update scripts that use the `Variants` command-line parameter.

Project and File Management

Project References: Explore the full project reference hierarchy and associated files directly from your Simulink Project

View all project references in your Simulink Project in a deep hierarchy. You can view project and modified files for a selected reference. The new references toolstrip improves workflow efficiency.

For details, see [Componentization of Large Projects](#).

Project-Wide Bus Renaming: Automatically update all bus references across Simulink Project when you rename a bus or bus element

When in a Simulink project, renaming buses and bus elements using the Simulink Bus Editor prompts you to update all usages in the project. For more information, see [Automatic Updates When Renaming, Deleting, or Removing Files](#).

Model Comparison: Use custom filters to simplify and focus model comparisons

Define custom filters in Simulink model comparison reports to simplify reports and focus on specified elements. You can import and export filters for sharing. For more information, see [Filter Comparison Reports](#).

Generated Code Traceability: View code files generated from models in the dependency graph

In R2018b, you can see what code is generated by each model directly in the dependency graph.

- The dependency analysis shows all generated files with connections to their parent model.
- Any generated files not included in the project are indicated.
- Use the graph to find what code needs to be regenerated if you modify a model.

For more information, see [Run Dependency Analysis](#).

Git Stashes: Store uncommitted changes for later use

In R2018b, you can use Git stashes directly from MATLAB. You can create a Git stash to store uncommitted changes for later use. For details, see [Use Git Stashes](#).

Source Control: Simplified workflow for creating projects from Git or SVN

The Simulink start page now includes templates for Git and SVN, enabling simpler project creation. For details, see [Clone Git Repository](#) or [Check Out SVN Repository](#).

Data Management

Block Parameter Autocomplete: Improve the speed and accuracy of block parameter editing by selecting from suggested variable or function names as you type

Simulink helps you edit a block parameter quickly and accurately by providing a list of variables to select from. In a block dialog box or in the Property Inspector, Simulink suggests variables based on the current text typed in the edit field. The suggestions include relevant variables or objects from every workspace (base, model, and mask), data dictionary, and referenced dictionary visible to the editable block parameter. Autocomplete is available for:

- Variables
- Fields of structures and objects
- Functions on the MATLAB path

The suggestion list includes the icon, value, and source of the variable.

For more information, see Set Block Parameters.

Undefined Variable Detection: Get more information about missing variables while editing your model

If the variables that a model needs are not available, edit-time checks for block parameters and signal lines provide error cues near the affected blocks. In R2018b, clicking the error cue provides more information about a missing variable. The check can detect these conditions and provide the corresponding fixes:

- The variable was deleted from a workspace or data dictionary. If the variable is deleted while the model is open, you can undo the deletion.
- The variable was renamed in a data dictionary. If the model is open or if the Simulink cache file (.slxc) is available, you can copy the renamed variable.

When you update the block diagram, these same options are available from the Diagnostic Viewer.

Model Workspace API: Read and modify variable properties in the model workspace

Read and modify variable properties in the model workspace by using the functions `getVariablePart` and `setVariablePart`. Function inputs can be of type:

- Structure array
- Cell array
- Vector or matrix
- Class

For setting and getting variable properties in the model workspace, consider using `setVariablePart` and `getVariablePart` instead of `evalin` because:

- The functions do not create new variables or cause unintended results.
- `getVariablePart` does not dirty the model.

Model Data Editor Enhancements: Perform actions on referenced data objects from a context menu

Perform common actions on referenced data objects without leaving the Model Data Editor. In the Model Data Editor, when you right-click a row that contains a referenced object from a workspace or data dictionary, a context menu provides these actions:

- **Open** — Opens the property dialog box for the data object.
- **Explore** — Opens the Model Explorer, then selects:
 - In the **Model Hierarchy** pane, the workspace or data dictionary that contains the data object.
 - In the **Contents** pane, the data object.
- **Find Where Used** — Finds where the data object is used.
- **Rename All** — Renames the data object and the references to the object.

Model Explorer Enhancements: Navigate your model using simplified interface

Enhancements to the Model Explorer reduce complexity and make it easier to perform actions on elements in your model:

- **Toolbars:**
 - Search bar hidden by default
 - Toolbar buttons hidden when not in context
 - Library Browser button removed from toolbar
- **Model Hierarchy** pane: Embedded Coder Dictionary and Simulink Design Verifier nodes removed.
- **Contents** pane:
 - New column view, **Files**. The view provides the file management properties **Dirty**, **LastSaved**, **Created**, and **FileName**.
 - New filter options **Subsystems/Model References** and **Charts**. These options are turned off by default. Masked subsystems remain in the content list.
 - Existing filter option, **Named Signals/Connections**, turned off by default in R2018b.
 - If you select the Simulink Root node in the **Model Hierarchy** pane, new context menu options for models are **Open**, **Save**, and **Close**. **Save** and **Close** options can be used in batch mode.
 - If you select a model node in the **Model Hierarchy**, several items are no longer on the contents list: **Model Workspace**, **Embedded Coder Dictionary**, and **Simulink Design Verifier results**.

For more information on using the Model Explorer, see [Search and Edit Using Model Explorer](#).

Root Inport Mapper updates

Root Inport Mapper has these updates:

- The Options menu has a new option, **Notify of Missing Signals**, which identifies unassigned root inports and detects incomplete input data sets.
- The mapping mode you select from the toolstrip, such as **Block Name** or **Port Order**, is now maintained between Root Inport Mapper sessions and models. This capability frees you from having to select the map mode each time you want to map signal data to root inports.
- In previous releases, you could not use configuration references to reference configuration sets while compiling the model (**Options > Update Model** selected). You had to use the Model Explorer to active a configuration set first. Starting in R2018b, you can now use configuration references to reference configuration sets while compiling the model.

Signal Editor user interface updates

The Signal Editor user interface now supports fixed-point data.

In the **Author and Insert** and **Author and Replace** dialog boxes

- Select signal data types from the new Data Type option
- Create, replace, and edit fixed-point signals with MATLAB expressions.
- Receive proposed fixed-point data types based on specified MATLAB expressions.

The Signal Editor user interface now plots the real and imaginary parts of a signal in one plot. In previous releases, the interface plotted the real and imaginary parts in two separate plots.

For more information, see [Create and Edit Signal Data](#).

Underspecified dimension diagnostics

Simulink has better detection algorithms to detect when signal dimensions cannot be inferred at compile time. As a result, you will see more diagnostics when signal dimensions are not fully specified.

To control how Simulink shows you these diagnostics, use the Underspecified dimensions configuration parameter.

Propagated Signals: Display all propagated signal labels

Currently, in the Signal Properties dialog box, you can display propagated signal labels for a signal by selecting **Show propagated signals**. In R2018b, continue to display propagated signal labels for individual signals, or display the propagated signal labels for all signals in a model. From the **Display** menu in the Simulink Editor, select **Signals & Ports > Propagated Signal Labels**.

For more information, see [Signal Label Propagation](#).

Enumerated Data Types: Use functions to find, inspect, and delete enumeration classes

Find, inspect, and delete enumeration classes defined by `Simulink.defineIntEnumType` by using these functions:

- `Simulink.findIntEnumType` — Find a specific enumeration class by name or find all enumeration classes. Use the returned `meta.class` object to query attributes of the enumeration class.
- `Simulink.clearIntEnumType` — Delete a specific enumeration class by name or delete all enumeration classes. The function generates a warning if the class name is invalid or if a class cannot be deleted because instances of the class exist.

These functions do not find or delete enumeration classes that are defined by `classdef` files on the MATLAB path.

MATLAB variables: View dimensions and complexity in Simulink tools

Previously, Model Explorer, Model Data Editor, and Property Inspector displayed information about dimensions and complexity for Simulink data objects. In R2018b, these tools also display dimensions and complexity information for MATLAB variables.

Simulink Data Dictionary Version Handling: Use and export model data saved in previous releases with models created in later releases

R2018b introduces Simulink data dictionary version handling. You can:

- Link a model to a data dictionary that was saved in a previous version of Simulink (for example, you can link a model that you develop in R2018b with a dictionary saved in R2018a).
- Continue developing a model, which has a data dictionary, that was created with a version of Simulink that is older than the current version (for example, if you started developing a model that uses a data dictionary in R2018a, you can open and continue development of that model in R2018b).
- Export (save) a data dictionary for use in models created with a previous version of Simulink.

For more information, see [Dictionary Usage for Models Created with Different Versions of Simulink](#).

Legacy Code Tool: Integrate code with 64-bit variables

In R2018b, Legacy Code Tool can integrate C functions that use long 64-bit integer types.

Configuration Parameters: View referenced configuration with a single click

For a model using a referenced configuration, you can view the configuration set more quickly. On the model editor toolbar, if you click the Model Configuration Parameters icon, the Configuration Reference dialog box displays a read-only view of the referenced configuration. In the dialog box you can browse, search, and get context-sensitive help for parameters in the same way you do in the Configuration Parameters dialog box.

From the toolbar on the Configuration Reference dialog box:

- View the name and location of the referenced configuration.
- Select a different referenced configuration to view.
- Open the configuration set for edit in the Configuration Parameters dialog box.
- Refresh the view after configuration changes.

For more information, see [Manage a Configuration Reference](#).

Model Data Editor: View referenced variables on the Parameters tab

In the Model Data Editor, the Parameters tab displays any variable that is used by a runtime tunable block parameter. In R2018b, this includes:

- Structures and cell arrays whose members are referenced by a tunable block parameter
- Variables of type string (such as those used by the String Constant block)

Version History

Some variables are no longer available in the Model Data Editor, including:

- Variables used by non-tunable block parameters. For example, the minimum and maximum parameters on a Gain block or the `Sample time` on a Constant block.
- Variant control variables
- Variables used for symbolic dimensions

For information on Model Data Editor limitations, see [Model Data Editor Limitations](#).

Log Datastore Memory blocks while your model is configured for multi-tasking

You can now log Datastore Memory block data when your model is configured for multi-tasking.

Load string data with the From Workspace block

The From Workspace block now supports loading string data.

Block Enhancements

Row-Major Array Layout: Simplify integration with external C/C++ code for Lookup Table and other blocks

Most Simulink blocks, including the lookup table blocks, now support row-major array layouts for code generation. In previous releases, Simulink blocks supported only column-major array layouts for code generation. Row-major layout can improve performance for certain algorithms and ease integration with other code that also uses row-major layout.

Array layout behavior is controlled by these model configuration parameters:

- **Code Generation > Interface > Array layout** — Specify the array layout of model data in the generated code.
- **Math and Data Types > Use algorithms optimized for row-major array layout** — Enable row-major algorithms, which perform best with respect to speed and memory usage when operating on data with row-major memory layout.

For more information on row-major support, see “Row-Major Array Layout: Simplify integration with external C/C++ code for Lookup Table and other blocks” (Simulink Coder).

New algorithms for lookup table blocks

As part of the row-major support, the Direct Lookup Table (n-D), Interpolation Using Prelookup, and n-D Lookup Table blocks have new algorithms that work with row-major arrays.

The column-major and row-major algorithms may differ semantically in output calculations. For example, assume a Direct Lookup Table (n-D) where a vector is selected as output from a table (**Inputs select this object from table** parameter set to Vector). The elements of the selected vector are contiguous in the table storage memory. This table shows the column-major and row-major algorithm depending on the table dimension:

Table Dimension	Column-Major Algorithm	Row-Major Algorithm
2-D table	Column vector is selected	Row vector is selected
3-D and higher table	Output vector is selected from the first dimension of the table	Output vector is selected from the last dimension of the table

C Caller Block: Call external C functions directly from the model

Use the C Caller block to integrate your C code into Simulink. The C Caller block supports simple algorithmic integration for seamless simulation and code generation projects.

To use the C Caller block, click **Library Browser** on the toolbar, and find the C Caller block under **Simulink > User Defined Functions**. You can specify your external C code from **Configuration Parameters > Simulation Target**. Select your resolved C functions to call in the **Block Parameters** dialog box. You can access your external C code specifications from the **Block Parameters** dialog box by clicking on the **Configuration Parameters** button.

For more information on the C Caller block, see Integrate C Code Using C Caller Blocks.

Signal Editor Block Enhancement: Set active scenarios and signals by index

The Signal Editor block has these updates to simplify the script workflow in iterating over all the scenarios and signals:

- The existing `ActiveScenario` and `ActiveSignal` properties now accept index values. Previously, they accepted only character vectors.
- The new `NumberOfScenarios` and `NumberOfSignals` properties return a character vector representing the total number of scenarios and signals known to the block. Use the `str2double` function to convert the character vector to a double.
- The Signal Editor block now accepts string signals.

String block updates

- String blocks now support virtual and nonvirtual buses.
- Stateflow now supports strings in C action language charts. For more information, see [Manage Textual Information by Using Strings \(Stateflow\)](#).

Prelookup and Interpolation Using Prelookup blocks support virtual buses

The Prelookup and Interpolation Using Prelookup blocks now support virtual buses. You can now combine two signals into a virtual bus

Direct Lookup Table block supports symbolic dimension

The Direct Lookup Table block now supports symbolic dimensions.

Interpolation Using Prelookup block selection port support of enumerated data

The selection port (*sm*) of the Interpolation Using Prelookup block now accepts enumerated data types.


State owner blocks: Read and write block state for Second-Order Integrator and Outport blocks when running initialize, reset, and terminate functions

Supported state owner blocks for the State Reader and State Writer blocks now include the Second-Order Integrator and Outport blocks. See [State Reader and State Writer block reference](#).

Use State Reader and State Writer blocks with Initialize Function, Reset Function, Terminate Function, and Function-Call Subsystem blocks.

Add signals to Floating Scope blocks and Signal Viewers with connect mode

Use connect mode to select which signals to connect to a Floating Scope or Scope Viewer.

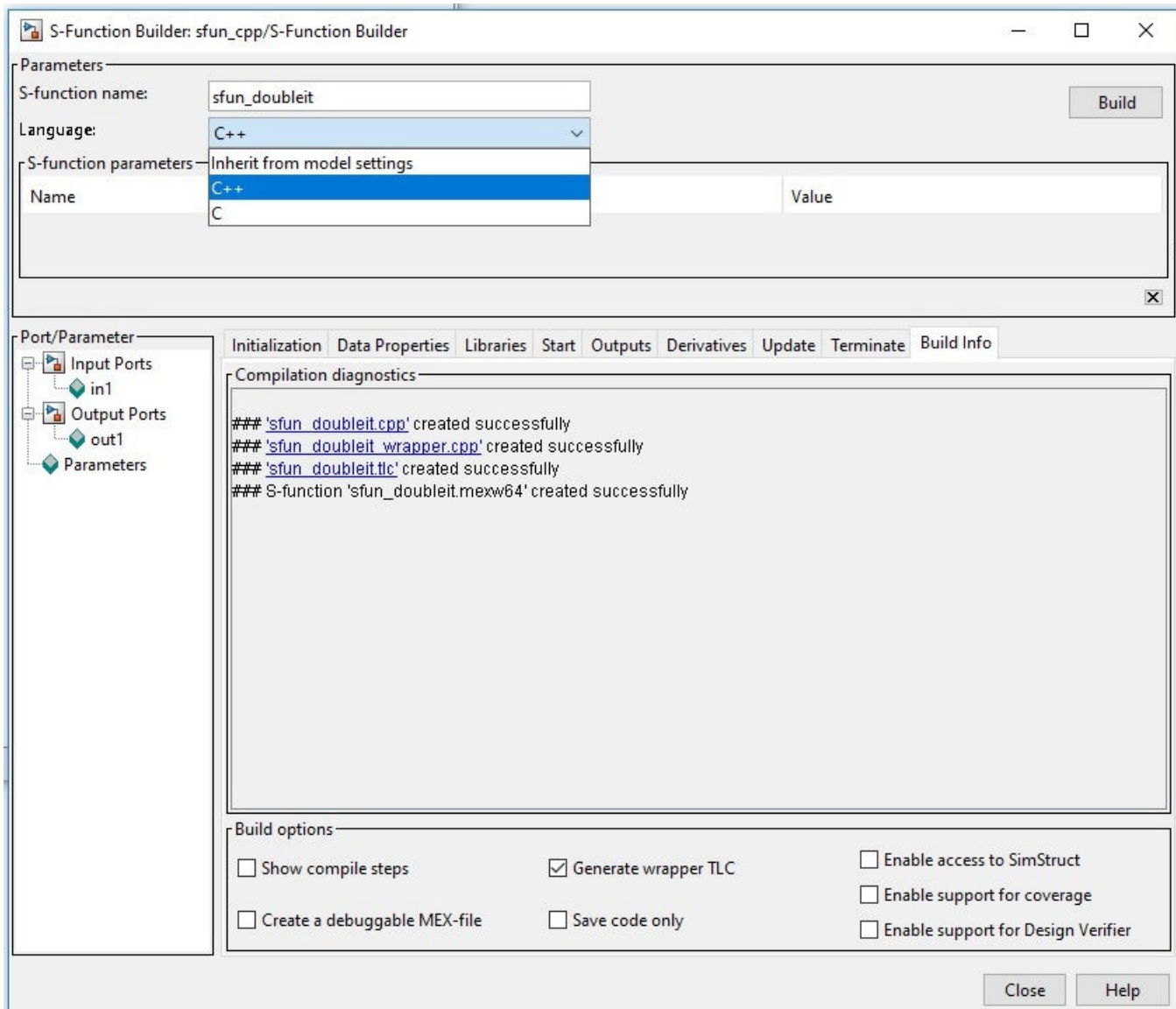
In the scope window, select the signal selection button . The Simulink canvas greys, indicating connect mode is active for connecting signals to the scope. For each signal you want to connect to the scope, click on the signal or block and select the checkbox in the **Connect** pop-up dialog.

Click-and-drag to select multiples signals and blocks to connect to the scope.

When you've finished adding signals, turn off connect mode by selecting the signal selection button again or by clicking the **X** button in the upper right corner.

S-Function Builder: Option to configure the language to generate the S-function

Starting in R2018b, S-function builder includes the option to configure the language that is used to generate the S-functions. Language options include C or C++ programming languages, or inheriting from the model settings. The new language option lets you choose the target language easily from the S-function builder dialog. In previous releases, to control the generated S-function language, you accessed the language setting using **Configuration Parameters > Code Generation > Language**.



Faster Standalone Code for Linear Algebra: Generate code that takes advantage of your own target-specific BLAS library

To improve the simulation speed of MATLAB System block algorithms that perform certain low-level vector and matrix computations (such as matrix multiplication), the simulation software can call BLAS functions. In R2018b, if you use Simulink Coder to generate C/C++ code for these algorithms, you can specify that the code generator produce BLAS function calls. If you specify that you want to generate BLAS function calls and the input arrays for the matrix operations meet certain criteria, the code generator produces calls to relevant BLAS functions. The code generator uses the CBLAS C interface.

BLAS is a software library for numerical computation of basic vector and matrix operations that has several highly optimized machine-specific implementations. MATLAB uses this library for basic matrix computations. Simulink uses the BLAS library that is included with MATLAB. Simulink Coder uses the

BLAS library you specify. If you do not specify a BLAS library, the code generator produces code for the matrix operation instead of generating a BLAS call.

To specify that you want to generate BLAS function calls and link to a specific BLAS library, see [Speed Up Matrix Operations in Code Generated from a MATLAB Function Block \(Simulink Coder\)](#) in the Simulink Coder documentation.

Connection to Hardware

Log data in MAT-files on an Android device

You can save data from Simulink models in MAT-files on an Android device. MAT-file logging enables you to monitor the behavior of the signal and perform analysis of historical data. To save signals in MAT-files, your Simulink model must contain an Outport or a To Workspace block. In MAT-files, the data can be saved in three formats: Structure, Structure with time, and Array.

To use this feature with the Simulink Support Package for Raspberry Pi Hardware, you must have a Simulink Coder (Simulink Coder) license.

Retrieve data from an Android device using the Internet of Things

You can retrieve data from ThingSpeak and load the data on your Android device by using the ThingSpeak Read block.

ThingSpeak is a web service for Internet of Things applications that provides real-time data collection, data processing, visualizations, apps, and plug ins.

Measure data using Arduino sensor blocks

You can use these blocks to measure data from the sensors connected to an Arduino hardware.

- Ultrasonic Sensor: Measures distance between the ultrasonic sensor and the nearest object in front of the sensor.
- BNO055 IMU Sensor: Measures acceleration, angular rate, and magnetic field and calculates fusion values such as Euler angles, quaternion, linear acceleration, and gravity vector along the axes of the Adafruit® BNO055 sensor.
- Tachometer: Measures the rotational speed of the shaft from the tachometer.

Publish and subscribe to messages using Raspberry Pi MQTT blocks

You can use the MQTT Publish and MQTT Subscribe blocks to publish and subscribe to messages from the Raspberry Pi hardware. Message Queuing Telemetry Transport (MQTT) (Simulink Support Package for Raspberry Pi Hardware) is a publish-subscribe architecture to connect bandwidth and power-constrained devices over wireless networks. For more information, see Publish and Subscribe to Messages on ThingSpeak Using MQTT Blocks (Simulink Support Package for Raspberry Pi Hardware).

Support added for Raspberry Pi 3 Model B+ board

Simulink Support Package for Raspberry Pi Hardware now supports building Simulink models and generating code for the Raspberry Pi 3 Model B+ board. You can use all the blocks from the support package with the Raspberry Pi 3 Model B+ board.

MATLAB Function Blocks

Faster Standalone Code for Linear Algebra: Generate code that takes advantage of your own target-specific BLAS library

To improve the simulation speed of MATLAB Function block algorithms that perform certain low-level vector and matrix computations (such as matrix multiplication), the simulation software can call BLAS functions. In R2018b, if you use Simulink Coder to generate C/C++ code for these algorithms, you can specify that the code generator produce BLAS function calls. If you specify that you want to generate BLAS function calls and the input arrays for the matrix operations meet certain criteria, the code generator produces calls to relevant BLAS functions. The code generator uses the CBLAS C interface.

BLAS is a software library for numeric computation of basic vector and matrix operations that has several highly optimized machine-specific implementations. MATLAB uses this library for basic matrix computations. Simulink uses the BLAS library that is included with MATLAB. Simulink Coder uses the BLAS library that you specify. If you do not specify a BLAS library, the code generator produces code for the matrix operation instead of generating a BLAS call.

To specify that you want to generate BLAS function calls and link to a specific BLAS library, see *Speed Up Matrix Operations in Code Generated from a MATLAB Function Block (Simulink Coder)* in the Simulink Coder documentation.

Sparse Matrices: Generate code for the backslash operation on sparse matrices

MATLAB Function block now supports code generation for the backslash operation on sparse matrices. The backslash operation (also called `mldivide` and matrix left division) solves linear systems of equations. To implement backslash, the code generator uses the SuiteSparse library and code from the CXSparse package. See <https://faculty.cse.tamu.edu/davis/suitesparse.html>.

Class Support: Use objects in more functions for code generation

These functions now support value classes for code generation:

- `coder.load`
- `load`

Code generation for more MATLAB functions

- `lscov`
- `func2str`
- `num2str`

Code generation for `rng` 'shuffle' option

Code generation now supports the `rng` 'shuffle' option.

The generated code for `rng('shuffle')` might produce different seeds than MATLAB produces.

Code generation for more Communications Toolbox functions

- `awgn`
- `bsc`
- `wgn`

Fuzzy Logic Toolbox Code Generation: Generate code to load and evaluate Fuzzy Inference Systems

In R2018b, Fuzzy Logic Toolbox™ supports C/C++ code generation for built-in membership functions and these functions:

- `evalfis`
- `evalfisOptions`
- `getFISCodeGenerationData`

For the list of Fuzzy Logic Toolbox functions supported for code generation in R2018b, see [Fuzzy Logic Toolbox in Functions and Objects Supported for C/C++ Code Generation — Category List](#).

External C code integration

Use your custom C code in MATLAB Function blocks. For more information, see [Import custom code and Integrate C Code Using the MATLAB Function Block](#).

Functionality being removed or changed

MATLAB Function blocks that integrate custom code may need to turn off option Import Custom Code in the Configuration Parameters

Behavior change

When using `coder.cinclude`, specify your header and source code file locations in the Simulation Target pane of the Configuration parameters and uncheck the **Import Custom Code** option. Alternatively, you can use the `coder.cinclude` and `coder.updateBuildInfo` functions to specify source and header files within your MATLAB code. Prior to R2018b, this option was called **Parse Custom Code Symbols**. For more information see [Import custom code](#).

S-Functions

Check your S-function for undocumented APIs

In R2018b, to avoid any incompatibility issues in the future releases due to the use of undocumented SimStruct APIs, you can check your S-function source code for the use of undocumented APIs using:

- S-function checks.
- `mex -DUSE_PUBLISHED_ONLY` command. This command gives a compilation error if any undocumented APIs are used in the S-function code.

See [Enable Undocumented API Checks](#) for more information.

Row-major code generation support for S-functions

To write user-defined S-functions that expect row-major array layout for code generation, use the new function `ssSetArrayLayoutForCodeGen` function added to the SimStruct API. The syntax is:

```
ssSetArrayLayoutForCodeGen(S, SSArrayLayout)
```

You can set `SSArrayLayout` to:

- `SS_UNSET` - This setting is the default setting that disables the block for row-major code generation.
- `SS_COLUMN_MAJOR` - Specify the block for column-major code generation only.
- `SS_ROW_MAJOR` - Specify the block for row-major code generation only.
- `SS_ALL` - Specify the block as allowed for code generation with any array layout.

Because by default the S-function is set to `SS_UNSET`, use the model configuration parameter **External functions compatibility for row-major code generation** (Simulink Coder) to test if the S-function is compatible with the generated code. This parameter applies only if you have a multidimensional array in your S-function. For more details, see [Generate Row-Major Code for S-Functions](#) (Simulink Coder).

When you use the S-Function Builder block, you can set the array layout of your custom function by using the **Array layout** setting on the **Data Properties** pane. You can set **Array layout** to:

- Column-major
- Row-major
- Any

For more details, see [S-Function Builder Dialog Box](#).

For more information about row-major code generation, see [“Row-Major Array Layout: Simplify integration with external C/C++ code for LookUp Table and other blocks”](#) (Simulink Coder).

Modeling Guidelines

High-Integrity Modeling Guidelines: Verify compliance with DO, IEC, EN, and ISO safety standards

This table identifies removed and modified modeling guidelines. For a complete list of High-Integrity System Modeling guidelines, including their applicable Model Advisor checks, see Model Advisor Checks for High-Integrity Modeling Guidelines.

High-Integrity Modeling Guideline	Rationale
himl_0005: Usage of global variables in MATLAB functions	<p>Removed — Guideline no longer applies. Simulink is able to detect read and write access order in MATLAB function blocks.</p> <p>Corresponding Model Advisor check "Usage of global variables" was removed.</p>
hisf_0001: State Machine Type	Former guideline hisf_0001: Mealy and Moore semantics has been updated. Classic, Mealy, and Moore State Machine types are supported, as long as the same type is used for all charts in the model. The title of and support documentation for the guideline were updated to reflect this change.
hisf_0002: User-specified state/transition execution order	Removed condition "Set default transition to evaluate last.". The condition is covered in hisl_0311: Configuration Parameters > Diagnostics > Stateflow.
hisl_0006: Usage of While Iterator blocks	<p>Removed condition "Consider selecting Show iteration number port to observe the iteration value during simulation". The condition was a recommended method for debugging and is not appropriate as a guideline rule.</p> <p>This change does not affect the corresponding Model Advisor check.</p>
hisl_0009: Usage of For Iterator Subsystem blocks	<p>Removed — Guideline consolidated into hisl_0007: Usage of While Iterator subsystems because both guidelines prohibit the same set of blocks within iterator subsystems.</p> <p>The title for guideline hisl_0007 is updated to include For Iterator subsystems.</p> <p>This change does not affect the corresponding Model Advisor check.</p>
hisl_0015: Usage of Merge blocks	Added condition D to the guideline. This change reflects the behavior of the Merge block when connecting the output of a conditional subsystem.

High-Integrity Modeling Guideline	Rationale
hisl_0020: Blocks not recommended for MISRA C:2012 compliance	<p>Added condition G to the guideline because use of the specified String blocks is not recommended for compliance with MISRA C:2012 or secure coding standards.</p> <p>Block usage is checked in Model Advisor check Check for blocks not recommended for MISRA C:2012.</p>
hisl_0043: Configuration Parameters > Diagnostics > Solver	Removed Unspecified inheritability of sample times from the "If Diagnostic Parameter..." table. This parameter is covered in guideline hisl_0044: Configuration Parameters > Diagnostics > Sample Time.
hisl_0046: Configuration Parameters > Simulation Target > Block reduction	In R2018b, configuration parameter Block reduction moved to the Simulation Target pane in the Configuration Parameters dialog box. The title of and support documentation for the guideline were updated to reflect this change.
hisl_0060: Configuration parameters that improve MISRA C:2012 compliance	<p>Added configuration parameters Undirected event broadcasts, Compile-time recursion limit for MATLAB functions, Enable run-time recursion for MATLAB functions, and External mode to the guideline.</p> <p>These parameter settings are checked in Model Advisor check Check configuration parameters for MISRA C:2012.</p>
hisl_0201: Define reserved keywords to improve MISRA C:2012 compliance	<p>Removed — Guideline consolidated into hisl_0032: Model object names because both guidelines check the use of reserved names and keywords in models.</p> <p>There are no Model Advisor checks for hisl_0201. This change does not affect the corresponding Model Advisor check for hisl_0032.</p>
hisl_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters	Added configuration parameter Detect loss of tunability to reflect the behavior of the corresponding Model Advisor check.
hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses	Added configuration parameter Bus signal treated as vector to reflect the behavior of the corresponding Model Advisor check.
hisl_0311: Configuration Parameters > Diagnostics > Stateflow	<p>Added configuration parameters Undirected event broadcasts and Transition action specified before condition action to match the configuration parameter constraints for Stateflow.</p> <p>These parameter settings are checked in Model Advisor check Check safety-related diagnostic settings for Stateflow.</p>

High-Integrity Modeling Guideline	Rationale
<p>hisl_0022: Data type selection for index signals</p> <p>hisl_0031: Model file names</p> <p>hisl_0039: Configuration Parameters > Code Generation > Interface</p> <p>hisl_0045: Configuration Parameters > Math and Data Types > Implement logic signals as Boolean data (vs. double)</p> <p>hisl_0046: Configuration Parameters > Simulation Target > Block reduction</p> <p>hisl_0047: Configuration Parameters > Code Generation > Code Style</p> <p>hisl_0048: Configuration Parameters > Math and Data Types > Application lifespan (days)</p> <p>hisl_0049: Configuration Parameters > Code Generation > Symbols</p> <p>hisl_0052: Configuration Parameters > Code Generation > Optimization > Data initialization</p> <p>hisl_0053: Configuration Parameters > Code Generation > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values</p> <p>hisl_0054: Configuration Parameters > Code Generation > Optimization > Remove code that protects against division arithmetic exceptions</p> <p>hisl_0056: Configuration Parameters > Code Generation > Optimization > Optimize using the specified minimum and maximum values</p> <p>hisl_0062: Global variables in graphical functions</p> <p>hisl_0063: Length of user-defined object names to improve MISRA C:2012 compliance</p>	<p>Updated to reference the new Model Advisor check that corresponds to each guideline. For additional information about these new checks, see Additional MAAB 3.0 and High-Integrity Checks (Simulink Check) in the R2018b Simulink Check™ Release Notes.</p>

High-Integrity Modeling Guideline	Rationale
hisf_0014: Usage of transition paths (passing through states)	
himl_0001: Usage of standardized MATLAB function headers	
himl_0006: MATLAB code if / elseif / else patterns	
himl_0007: MATLAB code switch / case / otherwise patterns	
himl_0008: MATLAB code relational operator data types	
himl_0009: MATLAB code with equal / not equal relational operators	
himl_0010: MATLAB code with logical operators and functions	

R2018a

Version: 9.1

New Features

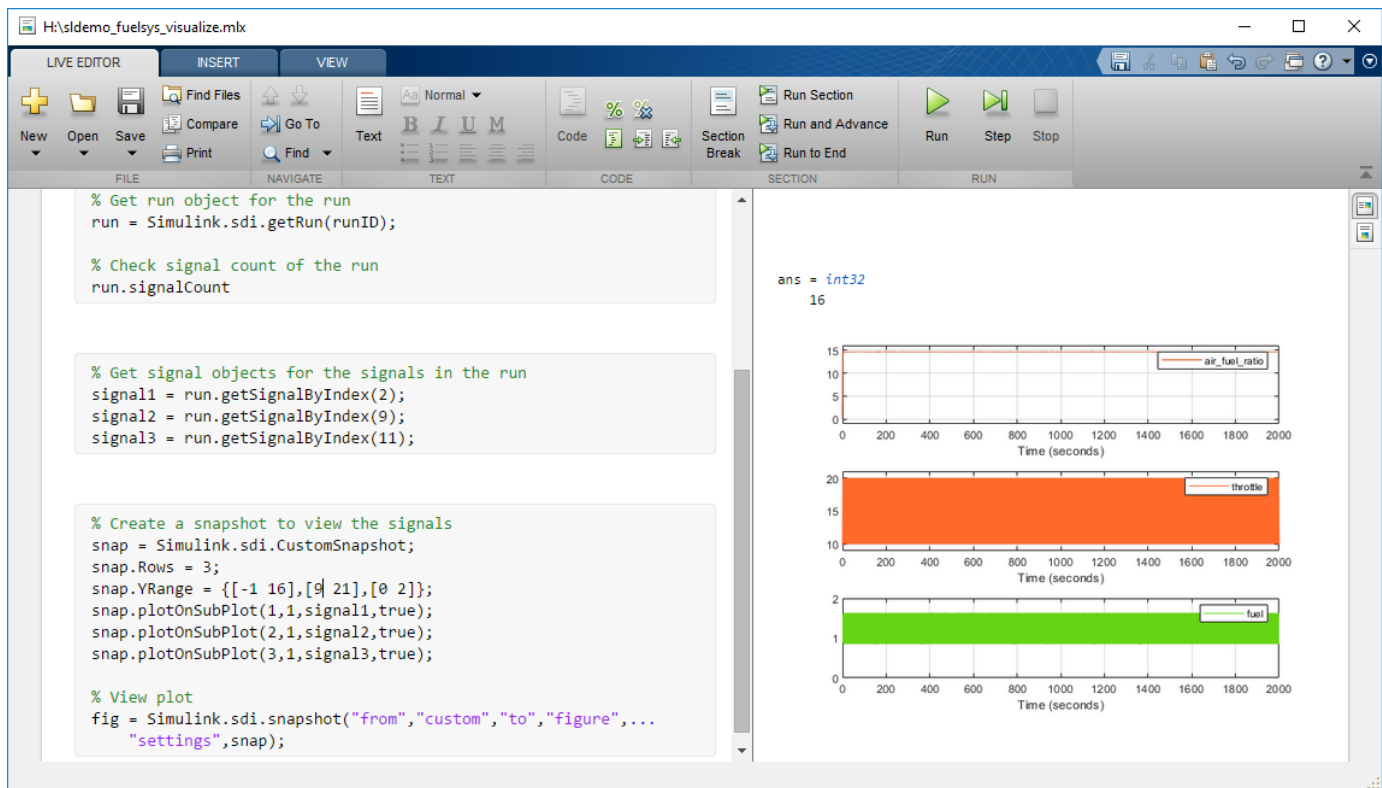
Bug Fixes

Version History

Simulation Analysis and Performance

Simulation Data Inspector in the Live Editor: Add, view, and edit Simulation Data Inspector plots directly in the Live Editor

Use the `Simulink.sdi.snapshot` function to create figures and images in the Live Editor. For fully scripted workflows, use a `Simulink.sdi.CustomSnapshot` object to configure a Simulation Data Inspector plot without opening the Simulation Data Inspector or affecting an open session. You can also use the `Simulink.sdi.snapshot` function to save a plot from the Simulation Data Inspector to a file or copy the image to your clipboard.



Simulation Pacing: Run simulations at wall clock speed or other specified pace for improved visualization

Simulation Pacing enables you to slow-down your simulations. You can specify a pace and Simulation Pacing ensures that simulations run slower than this pace. Slowed down simulation animations help you to observe the behavior of your model in finer detail. You can specify the rate as a ratio of simulation time to wall clock time. This feature also incorporates Dashboard blocks such as gauge, toggle switch and slider so you can fine-tune your simulation visualization as the simulation is running. With Simulation Pacing enabled, you can change the pace while the simulation is running without dirtying the model. For more information, see [Simulation Pacing](#).

Controllable Sample Time: Adjust the time interval for block execution during the simulation, for example, to vary the sampling frequency

You can now use a controllable sample time in C S-Function blocks. To understand how to use it in your own model, open `sfcn_demo_pwm`. You can register a C S-function block for the sample time using `ssSetControllableSampleTime` and specify its executions using `ssSetNumTicksToNextHitForControllableSampleTime`. For more information, see [Controllable Sample Time](#).

You can also use this sample time in MATLAB System Objects. See “Controllable sample time support for MATLAB System block” on page 9-25

Subsystem Domain Specification: Improve your solver performance and code by specifying the domains of subsystems within your model

You can constrain a subsystem in the time domain to Continuous or Discrete in order to improve solver performance and execution. See [Specify Time Domains](#) for more information.

Diagnostic Handling: Directly fix errors and warnings from diagnostic messages, such as algebraic loops, as well as manage suppressed diagnostics

You can quickly address solver-related issues such as algebraic loops and consecutive zero-crossings in your model directly from the diagnostic message. The diagnostic viewer and the command line now show a **Fix** button for the error or warning message.

The screenshot shows the MATLAB diagnostic viewer interface. At the top, there is a toolbar with icons for home, save, print, close, filter, and help, along with a search bar. Below the toolbar, a tab labeled "mStartTimeCorrected" is visible. The main content area shows a warning message under the "Simulation" category:

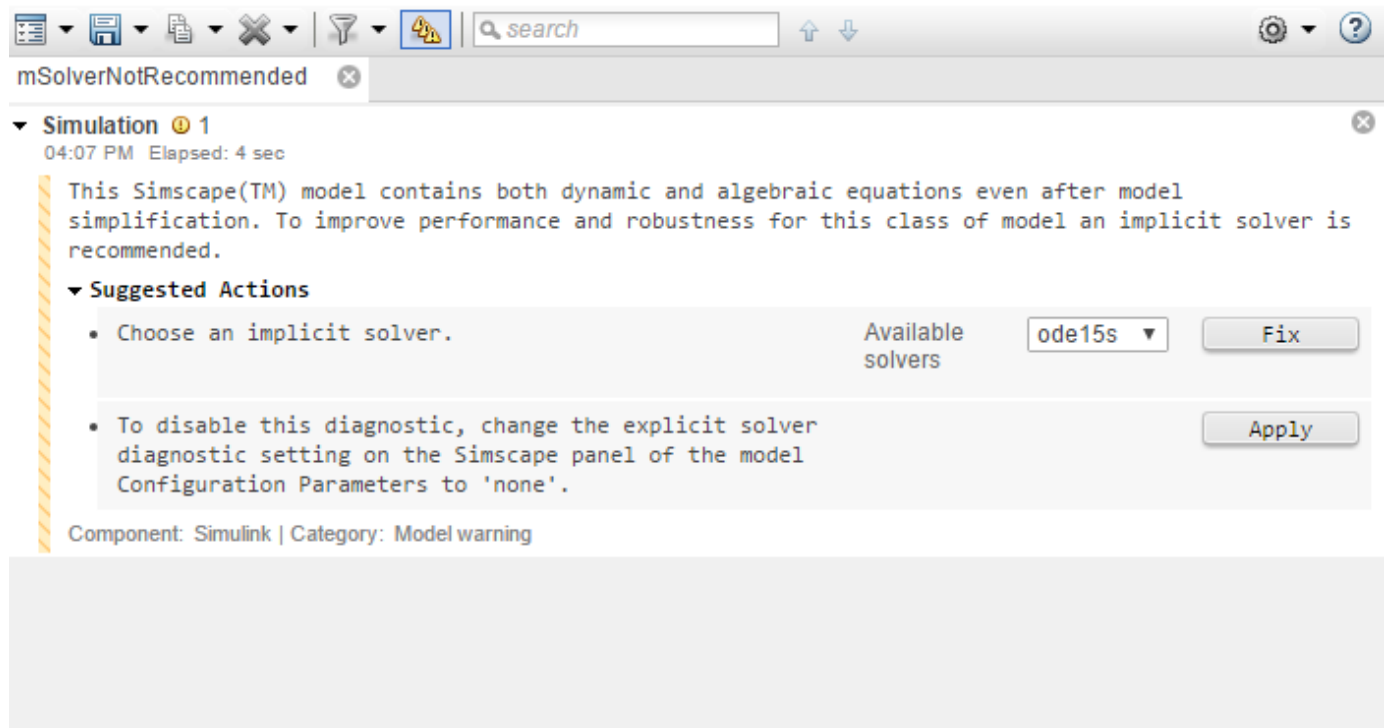
Simulation 1
04:19 PM Elapsed: 4 sec

The specified simulation start time (0.13456236234) is not an integer multiple of the fixed step size (0.1). Changing the start time to (0.2).

Suggested Actions

- You can disable this diagnostic by specifying a start time that is an integer multiple of the fixed step size. Enter start time:
- Set 'Automatic solver parameter selection' diagnostic to 'none' in the Diagnostics tab of the Configuration Parameters dialog.

Component: Simulink | Category: Block diagram warning



Simulink Execution Engine

Improved workflow across multiple simulation modes, such as accelerator and rapid accelerator, using the Simulink redesigned execution engine. The updated engine provides improved simulation robustness, better diagnostics, and provides a platform for future improvements in simulation performance and capabilities.

State Logging Compression: Reduce the size of your logged states data by choosing to use a real-time states sampling algorithm

To reduce the size of logged state information, the Solver Profiler now uses a real-time sampling algorithm based on state dynamics.

Adaptive Absolute Tolerance: Automatically fine-tune absolute tolerance values with an initial value to improve both performance and accuracy during simulation

In 2018a, the Solver configuration pane contains a new parameter called **Auto scale absolute tolerance**. You can enable or disable this parameter when you set your own value for **Absolute tolerance** to specify whether the tolerance should adapt to the states over the course of the simulation.

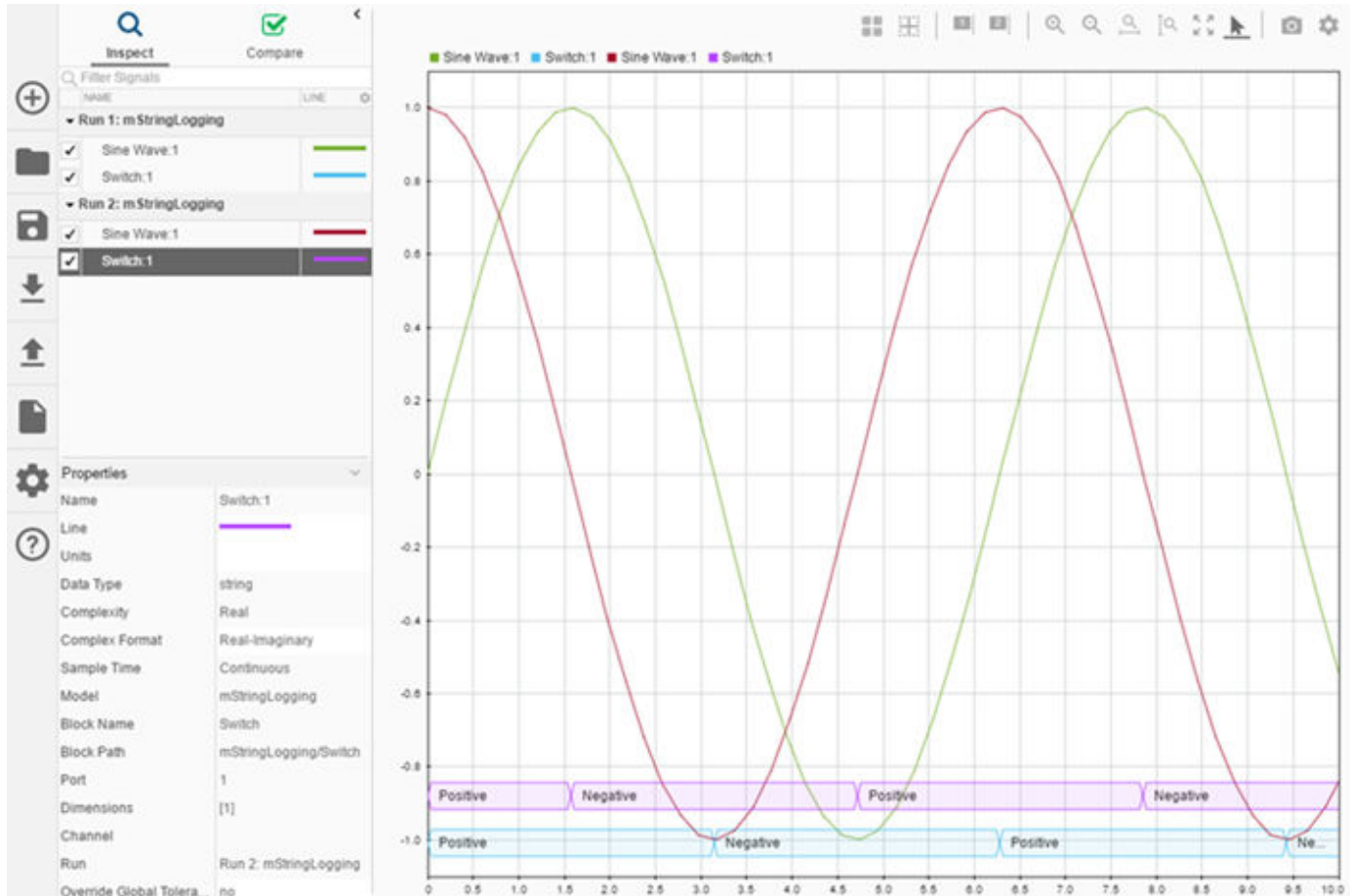
Log parameters and variables connected to Dashboard blocks

Parameters and variables connected to Dashboard blocks automatically log to the Simulation Data Inspector, where you can view them along with logged signal data. When you export a run containing

logged parameter data from the Simulation Data Inspector, each logged parameter signal appears in the exported `Simulink.SimulationData.Dataset` as a `Simulink.SimulationData.Parameter` object.

View string data in the Simulation Data Inspector

In R2018a the Simulation Data Inspector supports visualization of string data. You can log string signals to the Simulation Data Inspector to view along with other signal data from your model.



Access logged signal data during simulation

Run-time data access allows you to retrieve simulation data without adding blocks to your model and with little performance overhead. Provide a MATLAB function to process data during simulation. The function can access signal values, simulation time, and user-defined parameters in the model workspace or base workspace. You can configure callbacks individually for each logged signal. Right click a signal's logging badge and select **Properties** to access the **Data Access** parameters in the **Instrumentation Properties** dialog.

For more information, see [Process Signal Data During Simulation](#).

Import CSV and MDF files into the Simulation Data Inspector

In R2018a, you can import CSV and MDF files into the Simulation Data Inspector using the **Import** dialog. The mdf, mf4, mf3, data, and dat MDF file extensions are supported when you have a license for the Vehicle Network Toolbox™.

When you import large files, a status overlay in the lower right of the Simulation Data Inspector shows progress and allows you to cancel the import operation.

For more information about importing data into the Simulation Data Inspector, see [View Data from the Workspace or a File](#).

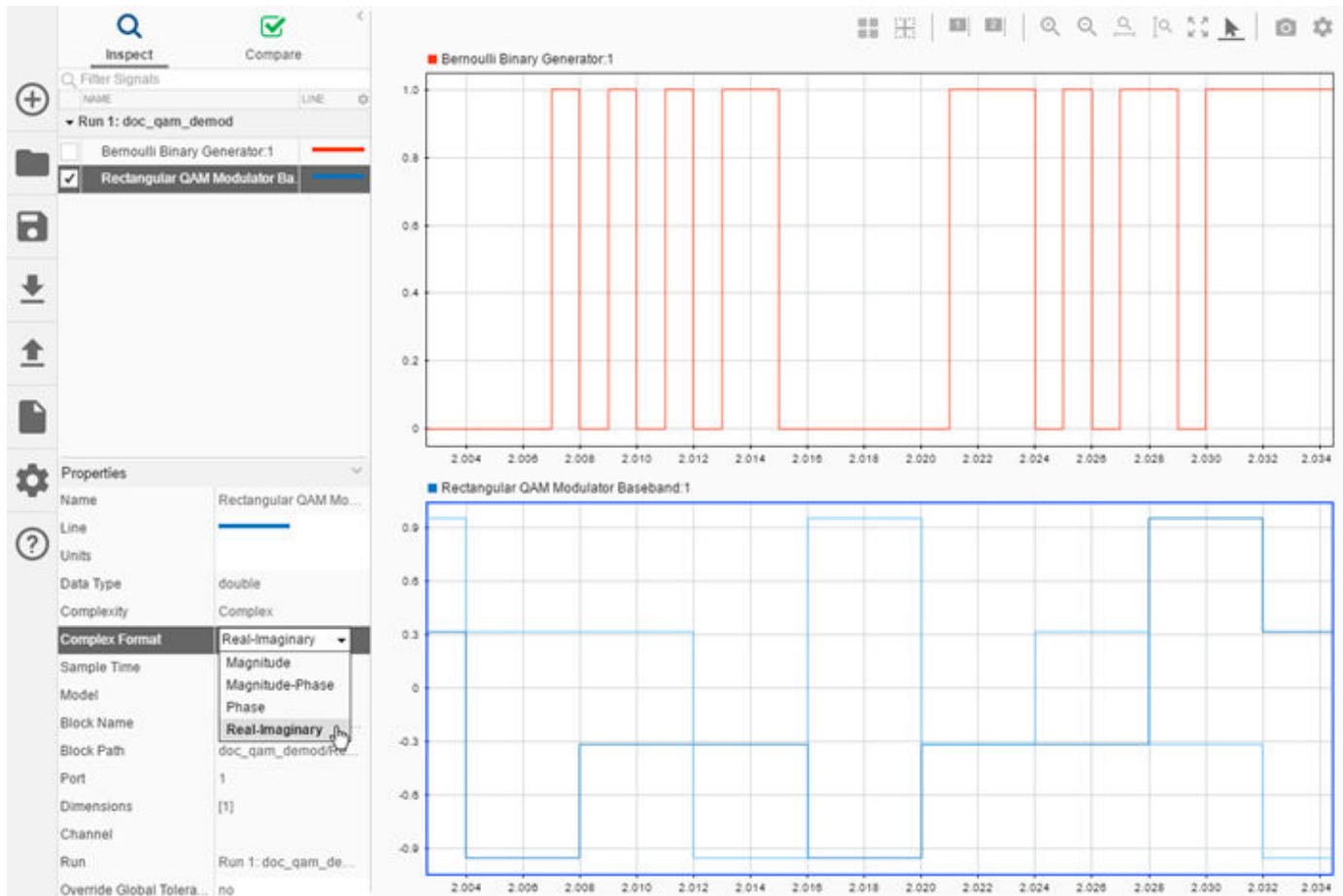
Control automatic import of parallel simulation data to the Simulation Data Inspector

In R2018a, you can specify which data to automatically import to the Simulation Data Inspector from parallel simulations. Using the `Simulink.sdi.enablePCTSupport` function, you can configure the parallel simulation support to import no data, only data from local workers, all data, or manually selected simulation runs. By default, simulation data from local workers automatically streams to the Simulation Data Inspector.

Improved Simulation Data Inspector support for signal processing

Using the **Instrumentation Properties** dialog box, you can specify a signal as frame-based and visualize it using the Simulation Data Inspector.

A Complex Format parameter in the Signal Properties allows you to specify how to display complex signals in the Simulation Data Inspector. You can specify Complex Format as Magnitude, Magnitude-Phase, Phase, or Real-Imaginary. When you specify Magnitude-Phase or Real-Imaginary format, the Simulation Data Inspector plots the two signal components together, using a shade variation for the phase or imaginary component.



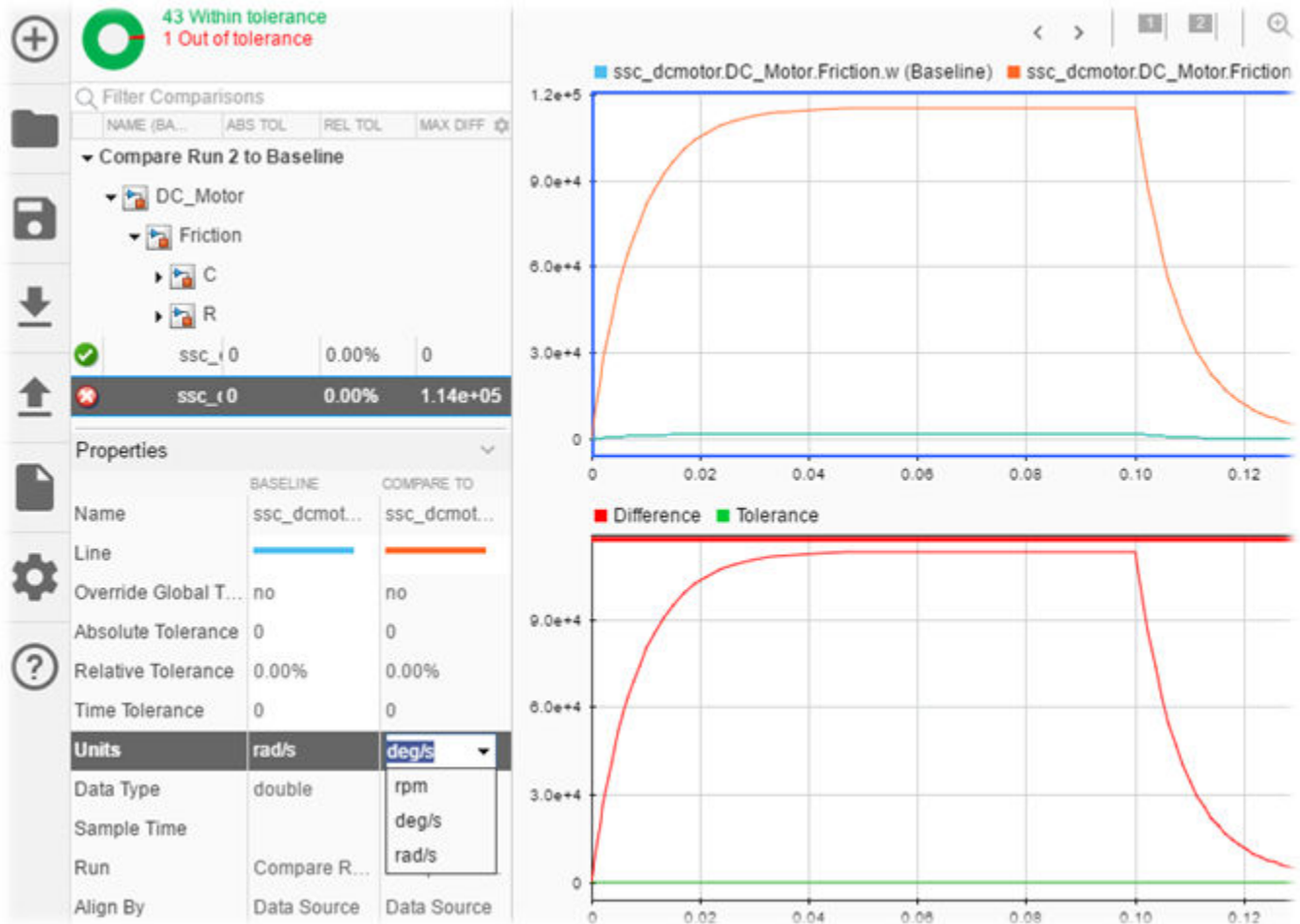
For more information on viewing frame-based and complex data in the Simulation Data Inspector, see [View Data with the Simulation Data Inspector](#).

Simulation Data Inspector comparison view signal grouping

Organize comparison results by grouping signals by **Model Hierarchy** and **Data Hierarchy** in the **Compare** pane. You can also group Simscape data in the **Compare** pane by **Physical System Hierarchy**. For more information on grouping data in the Simulation Data Inspector, see [Organize Your Simulation Data Inspector Workspace](#).

Convert signal units in the Simulation Data Inspector

Specify and convert signal units in the Simulation Data Inspector using the Units parameter in the Signal Properties. When your signal has units associated with it, the Simulation Data Inspector provides a list of suggested units for conversion.



For more information, see [Modify Signal Units](#).

Log Simscape data directly to the Simulation Data Inspector

In R2018a, you can log data directly from Simscape to the Simulation Data Inspector. You can view and compare data logged directly to the Simulation Data Inspector during simulation.

Simulink Editor

Predictive Quick Insert: Connect a recommended block to an existing block in your model

As you build your model, you can double-click the end of new signal lines to search for and select a block from a menu to connect to the signal. In R2018a, the menu suggests blocks to insert based on context. For example, if you draw a signal from a Gain block and double-click the new signal line, the menu includes math blocks such as Sum. You can select a listed block to insert or start typing to search for and insert any block.

Simulink comes with default suggestions. To learn about enhancing the list of suggested blocks based on your model designs, see [Improve Quick Block Insert Results](#).

Sample Time Legend: Interactively identify rates in your model through highlighting with the new Sample Time Legend

To assist you with interpreting your Simulink block diagram, the Sample Time Legend provides the color, annotation, description, and value for each sample time in the model. Using the updated Sample Time Legend, selecting a rate highlights the corresponding block in the model. For more information, see [View Sample Time Information](#).

Port Connection Cues: Highlight compatible ports as you draw a signal line

You can now use visual cues in your model to find compatible elements. You can also connect elements using two clicks instead of dragging.

In R2018a, you can highlight a port by clicking it. Compatible ports that you can connect to appear highlighted. As you move the cursor closer to a port, a preview of the connection appears. To make the connection, click the preview or the port you want to connect.

You can also now easily create a branch. After you click a port, move your cursor near a compatible line segment. A preview of the connection appears. Click to complete the connection. You can alternatively click the line first and then move toward the port to create the branch.

You can make multiple consecutive connections, for example, to branch off a new connection. Hold **Shift** and click the element you want to connect to. To make another connection, move the cursor toward the next element, and then click.

For more information, see [Build and Edit a Model in the Simulink Editor](#).

Model Notes: Add text or web links to describe any model layer

When you add notes to a model, you can enter text or show web page content to describe any system in the model hierarchy. Once you add notes to a model, any system that does not have its own notes inherits notes unless you explicitly specify not to show any notes for that system.

You can edit the notes using the text editor, which includes an equations editor in addition to standard formatting capabilities. To get started, in your model, select **View > Notes**. After you create

a notes file, you can add text or web content to each layer. Click the gear for help. For more information, see [Add or Read Notes for Model Layers](#).

Attribute Mismatch Detection: Find mismatches between components related to data type, complexity, and dimensions while editing your model

At model load time and edit, Simulink models now checks signal attributes in the interface display if the **Display > Errors & Warnings** check box is selected. For more information, see [Find Attribute Mismatches Between Components](#).

Content Preview: Display contents on new hierarchical elements by default

For new hierarchical items, such as subsystems, Stateflow charts, and model reference blocks that are open, a preview of the subsystem contents appear on the block by default. Existing subsystems are not affected. In previous releases, content preview was off by default. You can also now see content preview on some masked blocks.

You can turn the display on and off for each model element, and you can change the default using a Simulink Editor preference. For more information, see [Preview Content of Hierarchical Items](#).

Automatic Layout: Rearrange model elements from the command line

You can now use the `Simulink.BlockDiagram.arrangeSystem` command at the MATLAB command prompt to help to lay out your model. When you use this command, the input blocks align on the left, the output blocks align on the right, and the model elements align—left to right—in columns between the inputs and the outputs. For more information, see [Arrange Model Layouts Automatically](#).

Property Inspector displays sample command-line interface for block parameters

Property Inspector now displays the `set_param` and `get_param` calls for block parameters. You can use this sample code to help you programmatically set and get block parameters in your scripts. To access these samples, in the Property Inspector, click **Script Reference** to expand the pane. This pane displays the `set_param` and `get_param` calls that correspond to the block parameters and their current settings for each block parameter.

Find Model Elements: Interact with model while search executes

You can now interact with search results in the Simulink Editor as a search executes. For example, you can select a result and double-click it to go to the model element before the search finishes. You can sort results only after the search is complete.

For information on searching using Find, see [Search for Model Elements Using Find](#).

Signal Tracing: Show all possible paths

You can now see all possible paths a trace can take using the **Ctrl** and **Left/Right** arrow keys.

For more information on signal tracing, see [Highlight Signal Sources and Destinations](#)

Search Models Programmatically: New commands in Simulink API

Several functions and a class have been added to the Simulink API to search models programmatically.

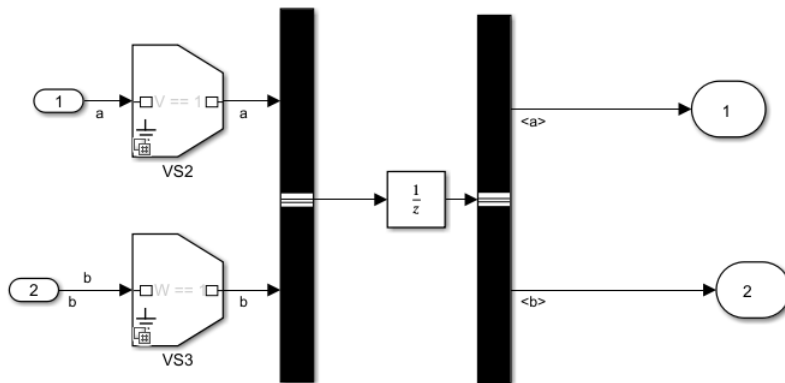
Syntax	Purpose
<code>Simulink.findBlocks</code>	Find blocks in a model or subsystem. Optionally constrain the search using a <code>FindOptions</code> object or by specifying block parameters.
<code>Simulink.findBlocksOfType</code>	Find blocks of a specified type in a model or subsystem. Optionally constrain the search using a <code>FindOptions</code> object or by specifying block parameters.
<code>Simulink.allBlockDiagrams</code>	Return all loaded block diagrams or return only loaded models or libraries.
<code>Simulink.FindOptions</code>	Class to create a <code>FindOptions</code> object that constrains the search. For example, specify the levels of hierarchy to search or whether to look under masks.

The new functions and class enable you to perform searches you can also perform with the existing function `find_system`. However, the new commands are more straightforward to use and the results easier to troubleshoot. The `find_system` function continues to work and is useful for returning model objects other than blocks.

Component-Based Modeling

Variant Signals in Buses: Use variant conditions to dynamically add and remove bus elements

Simulink now supports creation of buses from signals having variant conditions and selection of signals from bus signals having variant conditions. You can simulate or generate code from bus signals with variant conditions.



Mask Constraints: Ensure that mask parameters are within desired specifications without writing additional code

You can use the **Constraint Manager** available in the Mask Editor to manage constraints. Constraints ensure that mask parameters are within a desired specification, removing the need for writing additional code. You can associate a constraint with the **Edit** type mask parameters. Constraints can either be applied on one or many mask parameters or among mask parameters.

For example, you can restrict the data type of a mask parameter and ensure that this parameter value is greater than that of another mask parameter.

Diagnostic Suppression Review: Manage the activity of all suppressed diagnostics in a single interface

The **Suppress** button in the Diagnostic Viewer allows you to suppress certain diagnostics for a model.

From R2018a, you can use the **Suppressions** section of the Diagnostic Viewer to manage these suppressed diagnostics. You can also add justifications to the suppressed diagnostics.

set_param function on mask parameter reports edit-time and run-time errors

Before R2018a, the `set_param` command on a mask parameter ignored the edit-time errors. However, these errors were reported during run-time, that is, during update diagram or model simulation.

From R2018a, the `set_param` command on a mask parameter reports both edit-time and run-time errors.

Consider mask parameters, `min` and `max` with a condition `min < max` and initial values of `min` and `max` to be 0 and 10, respectively.

To change the value of `min` and `max` parameters to 20 and 40, respectively, you type in MATLAB command window:

```
set_param(gcb, 'min', '20', 'max', '40')
```

However, if you use this command sequentially to change the parameter value, the `set_param` command reports error during edit-time and run-time:

```
set_param(gcb, 'min', 20)
set_param(gcb, max, '40')
```

Simulink Mask: Enhance mask usability using additional parameters and options

From R2018a, you can use these parameters and options in the Mask Editor to enhance the mask usability:

- **Run-to-run** — To tune a mask parameter value between simulations in fast restart mode, use the **run-to-run** Tunable option on a mask parameter.
- **Align Prompts** — To align the parameter prompts on the mask dialog box, use the **Align Prompts** option available in the Property editor section of the mask container (for example, Group box, Tab) that contains the mask parameters.
- **Unit Parameter** — To define and control the measuring units for the inports and outport blocks of a masked Subsystem, use the **Unit** mask parameter.
- **Type option**: To change the container type of mask parameters, use the **Type** option available in the Property editor section of the Mask Editor dialog box.

For more information, see Mask Editor Overview.

Variant Systems: Structures as variant control is supported

From R2018a, Variant Source, Variant Sink, Variant Subsystem, Simulink Function, and the Initialize, Reset, and Terminate blocks support MATLAB structures as variant control.

- **Analyze all choices during update diagram and generate preprocessor conditionals** check box in the Block Parameter dialog box of the block has to be cleared to support structures as variant control.
- Simulink variant objects and Simulink parameters within structures are not allowed.

Library Links: Better architecture to control multilevel variants in library links

Before R2018a, changing the parameter value programmatically for a Variant block that is a linked block resulted in warning messages and a red star icon was added to the link block badge.

From R2018a, changing the parameter value programmatically for such a block does not result in warnings or change in link badge.

Variant Subsystem Block: Use conditionally executed systems within the Variant Subsystem block

Simulink now supports use of conditionally executed subsystems such as Enable, Trigger, Reset, and Function-Call within the Variant Subsystem block as Variant choices.

Enhanced Variant Manager functionalities for better usability

From R2018a, you can use these options in the Variant Manager to enhance the usability:

- Variant Manager Control Variables - Use the import option in the Variant Manager to import control variables across entire model reference hierarchy and see the usage of variant control in Variant Manager.
- Variant Reducer Summary - While reducing a model, use the **Generate detailed summary** option to generate a detailed summary containing information on **Variant Reducer Options, Original and Reduced Model Differences, Dependent Artifacts, Callbacks, and Warnings**. To generate detailed summary, you must have **Simulink Report Generator** license.
- Variant Reducer Full-Range Control Variables - While reducing a model, use the **Full - range** option to reduce a model based on the full-range variant control variable values. With this option, all possible valid values in the original model are considered for model reduction. For more information, see Reduce Models Containing Variant Blocks.

Project and File Management

Impact Analysis: Easily identify dependencies and impacts of block or requirement changes in your Simulink project

In a Simulink project, you can now find specific upstream or downstream dependencies at the block or requirement level. For example, you can find impacted files for a single block in a library, or for a single requirement. For details, see [Perform Impact Analysis](#).

Simulink Project Upgrade: Streamline upgrade to a new release by checking compatibilities of MATLAB code files in your project

In R2018a, when you upgrade your project, you can automatically run code compatibility checks on MATLAB code as well as all the Simulink upgrade checks. For details, see [Upgrade All Project Models, Libraries, and MATLAB Code Files](#).

SVN Source Control Workflows: Browse and compare change sets, and manage repository locks

When you are using SVN source control, the SVN file revisions history now shows a tree view of change set files with a context menu. You can select a particular file revision and view changes or save revisions. You can browse and compare files within committed SVN change sets. For details, see [Compare Revisions](#).

You can now monitor and break SVN locks. The SVN Repository Locks dialog box provides support for SVN locking workflows within teams:

- Determine who has a lock
- Break locks
- Group locks by user or file

For details, see [Get SVN File Locks](#).

Data Dictionary Workflows: Search inside data dictionaries and check for unsaved changes

Project-wide search can now search inside data dictionaries. On the **Simulink Project** tab, click **Search** and enter some characters to search for. See [Search Inside Project Files and Filter File Views](#).

You can now check your project for data dictionaries with unsaved changes, as well as models and MATLAB files. On the **Simulink Project** tab, in the **Tools** section, click **Unsaved Changes**. See [Find Models and Other Project Files With Unsaved Changes](#).

Simulink cache files (.slxc) and slprj folder generated more frequently

Previously, Simulink generated cache files (.slxc) and an slprj folder under specific conditions. In R2018a, Simulink generates these files and folders more frequently—when you update a block

diagram. By default, the `s_lxc` file appears in your current folder, but you can control the default location.

The cache file now contains information about workspace variables that the model uses. For information about the benefits, see “Permanent Storage of Variable Usage Information” on page 9-17. For general information about cache files, including how to control the default storage location, see [Reuse Simulation Builds for Faster Simulations](#).

Version History

If you use the `sim` function to simulate a model in a `parfor` loop, in R2018a, each worker in the parallel pool can attempt to access the same `s_lxc` file or a file in the `s_lprj` folder. The simulations can yield crashes and unexpected results.

To avoid this issue, at the beginning of the `parfor` loop, use `cd` to set the current folder of each worker to a unique location. For more information about this technique, see [Resolving Data Concurrency Issues](#).

Data Management

Model Design Data: Use multiple sources for your design data by allowing Data Dictionary to access the base workspace

In R2018a, you can configure a Simulink data dictionary to provide models with access to data in the base workspace. In the Model Explorer, in the **Model Hierarchy** pane, select the dictionary node. In the Dialog pane (the right pane), select **Allow access to base workspace**.

Configuring a dictionary in this way can help you to:

- Incrementally migrate a model from base workspace usage to dictionary usage. You can store new data in the dictionary while continuing to use existing base workspace data.
- Incrementally migrate a model reference hierarchy or collection of interdependent models to dictionary usage. You can keep a piece of shared data in the base workspace until you migrate all of the relevant models. See *Partition Data for Model Reference Hierarchy Using Data Dictionaries*.
- Take advantage of new features that depend on data dictionaries, such as the Embedded Coder Dictionary (see “Embedded Coder Dictionary: Create custom code generation definitions for data and functions” (Embedded Coder)). You can use a data dictionary without moving design data into it.

Undefined Variable Detection: Get instant notifications about missing variables while editing your model

If the variables that a model needs are not available when you update the block diagram, the model generates errors in the Diagnostic Viewer. R2018a introduces features to help you more quickly and easily recover.

Edit-Time Checks for Block Parameters and Signal Lines

When you open a model, error cues appear near affected blocks. You do not need to update the block diagram to discover the affected blocks. For options to resolve an error, click the cue.

To remove these cues from the block diagram, you can remove all such cues by clearing **Display > Errors and Warnings**.

Permanent Storage of Variable Usage Information

In previous releases, closing MATLAB discarded some variable usage information, limiting the recovery options that the Diagnostic Viewer offered in your next session or in a coworker's session.

In R2018a, Simulink cache files (.slxc) contain information about workspace variables and objects that models use. This stored information affords these benefits:

- Because the files persist when you close MATLAB, in your next session, the Diagnostic Viewer and edit-time checks can offer more recovery options than in previous releases.
- When you share a model with a coworker, sharing the cache file can enable the Diagnostic Viewer and edit-time checks to offer your coworker more options than in previous releases.

Simulink generates cache files and slprj folders more often. For more information, see “Simulink cache files (slxc) and slprj folder generated more frequently” on page 9-15.

Create and edit signals using MATLAB expressions

Use the Signal Editor user interface to:

- Use signal notations to create more complicated signals using the Author and Insert dialog window and MATLAB expressions.
- Edit multidimensional signals in the Signal Editor user interface.

For more information, see [Create and Edit Signal Data](#).

Import Signal Builder block signals and properties to Signal Editor block

To import signal data and properties from a Signal Builder block to a Signal Editor block, use the `signalBuilderToSignalEditor` function.

Parameter Dependencies: Explicitly model relationships between dependent and independent variables

You can use MATLAB variables and parameter objects (such as `Simulink.Parameter`) to set and share block parameter values in a model. In R2018a, you can set the value of a parameter object to a mathematical expression involving other variables or parameter objects. Use this technique to capture the relationships between dependent and independent variables in equations that you model.

For example, to represent the moment of inertia of an automobile wheel, you can create a parameter object whose value is an expression involving the wheel mass and radius. Store the mass and radius values in other objects. With the layer of abstraction that the expression provides, if you change the mass and radius values, you do not need to manually recalculate the moment value. For more information, see [Set Variable Value by Using a Mathematical Expression](#).

If you have Simulink Coder and Embedded Coder, you can configure the generated code to initialize a global variable by using an expression that involves system constants and other macros. See [“Parameter Initialization: Statically initialize tunable parameters from system constants and other macros” \(Embedded Coder\)](#).

Model Data Editor Enhancements: Configure data type and other design attributes of block parameters

R2018a introduces these enhancements to the Model Data Editor (**View > Model Data**):

- A button that opens documentation.
- A new menu item name, **Model Data Editor**, and an associated keyboard shortcut, **Ctrl+Shift+E**.
- The ability to configure design attributes, such as data type and minimum and maximum values, of block parameters. Use the **Design** view on the **Parameters** tab.

The target block must allow configuration of the attributes through the block dialog box and the Property Inspector. For example, you can configure the attributes of Gain or Discrete Filter block parameters, but not continuous-time Integrator or State-Space block parameters.

- In the Property Inspector, more accessible placement of properties shared by multiple sections (such as **Design** and **Code**)—at the top, outside any section.
- More relevant placement of the **Resolve** column—in the Design view as well as the Code view.

Simplified Specification of Initial Values: Use consistently named programmatic parameters

In R2018a:

- For the Ramp block, the programmatic parameter `X0` has a new name, `InitialOutput`.
`X0` corresponds to **Initial output** in the block dialog box.
- For the Fixed-Point State-Space block, the programmatic parameter `X0` has a new name, `InitialCondition`.
`X0` corresponds to **Initial condition for state** in the block dialog box.

Your existing scripts that use the old parameter names continue to work.

Configuration Parameters: View parameters that were on the Optimization panes on the Simulation Target and new Math and Data Types panes

Previously, in the Configuration Parameters dialog box, there were three optimization panes. In R2018a, there are no **Optimization** panes. The parameters that were on the **Optimization** panes are now on either the new **Math and Data Types** pane or the **Simulation Target** pane. This table shows the moved parameters and their locations in R2018a.

Parameter	Pane
Default for underspecified data type	Math and Data Types
Use division for fixed-point net slope computation	Math and Data Types
Use floating-point multiplication to handle net slope corrections	Math and Data Types
Application lifespan	Math and Data Types
Implement logic signals as Boolean data (vs. double)	Math and Data Types
Evaluated application lifespan	Math and Data Types
Block reduction	Simulation Target
Compiler optimization level	Simulation Target
Conditional input branch execution	Simulation Target
Signal storage reuse	Simulation Target
Verbose accelerator builds	Simulation Target

Load `matlab.io.datastore.SimulationDatastore` objects directly using root level Inport blocks

In R2018a, you can directly load `matlab.io.datastore.SimulationDatastore` objects using root level Inport blocks. The object can be passed as an external input alone, as an element of a

`Simulink.SimulationData.Dataset` object, or in the comma-separated list mixed with other types of data.

Flexible SimState Restore

You can rename a model between saving and restoring its `SimState`.

You can add, remove, and modify the following components in a model without affecting your ability to restore a previously saved `SimState`:

- Signal logging and state logging
- Outports
- Scope blocks
- Display blocks
- `SimViewingDevice` S-Functions. See, `SS_OPTION_SIM_VIEWING_DEVICE`
- To Workspace blocks
- To File blocks

Block Enhancements

Hit Crossing Messages: Create Messages for debugging and send them to Stateflow and SimEvents to indicate events in Simulink for hybrid system modeling

You can configure the Hit-Crossing block's Output Port to send messages to SimEvents or Stateflow. With the **Show output port parameter** enabled, set the **Output type** to message.

Strings: Design embedded systems with native support of strings

You can now use strings in Simulink models. Simulink strings are compatible with MATLAB strings.

Simulink strings are a built-in signal data type. They appear in the Simulink Editor as "strN" (for example, string with maximum length of N characters) or "string" for strings without maximum length. String lengths can range from 1 to 32766.

String literals are specified with double quotes ("Hello") or single quotes ('Hello'). To be consistent with MATLAB strings, use double quotes.

Simulink strings support 256 characters of the ISO/IEC 8859-1 character set. These characters are the first 256 code points of Unicode. Simulink does not support the first character char(0) ("NULL") and returns an error if the string contains this character.

This capability adds these string manipulation blocks:

Action	Blocks
Convert a uint8 vector to a string signal.	ASCII to String
Compose an output string signal based on the Format parameter and input signals.	Compose String
Scan an input string and convert it to signals per the format specified by Format parameter.	Scan String
Compare two input strings.	String Compare
Concatenate input strings to form one output string.	String Concatenate
Output the string specified by the String parameter.	String Constant
Return the index of the first occurrence of the pattern string sub in the text string str.	String Find
Output the number of characters in the input string.	String Length
Convert a string signal to a uint8 vector.	String to ASCII
Convert string signal to double signal.	String to Double
Convert string signal to single signal.	String to Single
Convert an input string to an enumerated signal.	String to Enum

Action	Blocks
Extract a substring from a string signal.	Substring
Convert the input signal to a string signal.	To String

For more information, see Simulink Strings.

Lookup Tables: Add enumerated inputs and breakpoints and use Lagrange interpolation for more powerful lookup tables

The lookup table blocks have been updated to support:

- Enumerated data inputs and breakpoints
- Lagrange interpolation

Enumerated data type in lookup table blocks

These blocks and objects now support enumerated data types for inputs and breakpoints:

- n-D Lookup Table
- Prelookup
- Simulink.LookupTable
- Simulink.Breakpoint

When using enumerated data types, save the enumerated data in the same folder as the model or on the MATLAB path.

For Prelookup blocks, you can use enumerated data types for these parameters:

- Breakpoints specified with explicit values or breakpoint objects accept enumerated data types. Breakpoints specified with even spacing do not accept enumerated data types.
- **Index search mode > Linear search** — You can use ordered or unordered data.
- **Index search mode > Binary search**— You must use ordered data.

For n-D Lookup Table blocks, you can use enumerated data types for:

- Breakpoints specified with explicit values or lookup table objects accept enumerated data types. Breakpoints specified with even spacing do not accept enumerated data types.
- **Index search mode > Linear search** — You can use ordered or unordered data.
- **Index search mode > Binary search**— You must use ordered data.

The Lookup Table Editor also reflects enumerated data types.

Lagrange interpolation method for lookup tables

The n-D Lookup Table, 1-D Lookup Table, 2-D Lookup Table, and Interpolation Using Prelookup blocks now support the Lagrange interpolation method (**Interpolation method > Linear Lagrange**).

Version History

The 'Linear' option for the command-line parameter `InterpMethod` has changed to 'Linear point-slope'.

Option Name	What Happens When You Use the Property Name?	Use These Functions or Function Elements Instead	Compatibility Considerations
'Linear'	Fails	'Linear point-slope'	<p>If you have scripts that get the interpolation method using the <code>get_param</code> function, update your scripts and replace the <code>InterpMethod</code> property 'Linear' value with 'Linear point-slope'.</p> <p>The <code>set_param</code> function continues to work and silently sets the parameter to 'Linear point-slope'.</p>

Direct Lookup n-D block Column option

The Direct Lookup Table (n-D) block **Inputs select this object from table** parameter `Column` option has been renamed to `Vector`. If you have existing scripts that set this option to `Column`, they continue to work.

Version History

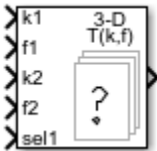
The 'Column' option for the command-line parameter 'InputsSelectThisObjectFromTable' has changed to 'Vector'.

Option Name	What Happens When You Use the Property Name?	Use These Functions or Function Elements Instead	Compatibility Considerations
'Column'	Fails	'Vector'	If you have scripts that get the object that inputs select from the table using the <code>get_param</code> function, update your scripts and replace the <code>'InputsSelectThisObjectFromTable'</code> property <code>'Column'</code> value with <code>'Vector'</code> . The <code>set_param</code> function continues to work and silently sets the parameter to <code>'Vector'</code> .

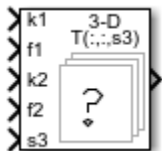
Interpolation Using Prelookup block visual affordance for subtables

To let you more easily see your subtable, the Interpolation Using Prelookup block icon now reflects the subtable selection ports.

In previous releases, a 2-D subtable selection from a 3-D table looked like this:



In R2018a, a 2-D subtable from a 3-D table looks like this:



Signal Editor block supports fast restart

The Signal Editor block now supports fast restart. For more information, see [Signal Editor](#).

Co-simulation data exchange

The Simulink platform supports co-simulation between components. A side effect of co-simulation are numeric inaccuracies introduced by co-simulation signal delays. To compensate for these inaccuracies, Simulink provides numerical compensation for the data exchange between co-

simulation components such as C MEX S-functions and FMU Co-Simulation blocks. For more information, see Co-Simulation Numerical Compensation.

Multicore co-simulation

The Simulink platform supports co-simulation between components. To help performance, you can run co-simulation components, such as C MEX S-functions and FMU Co-Simulation blocks, on multiple cores. For more information, see Run Co-Simulation Components on Multiple Cores.

FMU block changes

The FMU Co-Simulation block has the following changes:

- Debug tab renamed FMU Settings
- Communication step size parameter has moved to FMU Settings tab
- Relative tolerance parameter has been added to FMU Settings tab

Discrete FIR Filter block supports custom state attributes for better customization and efficiency of generated code

The Discrete FIR Filter block now supports custom state attributes to customize and generate code more efficiently. To access or set these attributes, in the Simulink editor, select **View > Model Data**.

Run-time diagnostics

Starting in R2018a, simulating these blocks in accelerator or rapid accelerator mode may generate run-time warnings or errors for invalid inputs or parameter settings. In previous releases, these inputs or settings caused no errors.

- Delay
- Magnitude-Angle to Complex
- Trigonometric Function
- Shift Arithmetic
- Substring
- String Concatenate
- String to ASCII

Unit System Configuration update

In previous releases, when adding a Unit System Configuration block to a nested subsystem, it inherited the settings of any Unit System Configuration block in a parent subsystem. Starting in R2018a, the Unit System Configuration block no longer inherits these settings.

Controllable sample time support for MATLAB System block

You can now use a controllable sample time in the MATLAB System block. To register a MATLAB System object for this sample time, use `createSampleTime` and specify its executions using `setNumTicksUntilNextHit`.

System object 'Discrete Periodic' sample time renamed to 'Discrete'

When you specify discrete sample time for a MATLAB System block, in the System object with `createSampleTime`, use 'Discrete' instead of 'Discrete Periodic'.

Version History

If your System object specifies the previous sample time name, 'Discrete Periodic', you must change it to 'Discrete'.

Nested protected models

You can now protect models that contain protected referenced models. See [Protect Models for Third-Party Use \(Simulink Coder\)](#).

Log data from Scope block as timetable

When logging signals using the Scope block, you can log data as a timetable.

If you set the configuration parameter **Dataset signal format** to `timetable` and the scope parameter **Save format** to `Dataset`, the data is saved as a timetable instead of a timeseries object.

Connection to Hardware

Automatically run Simulink model on Raspberry Pi after each restart

You can now use the run-on-boot feature to start a Simulink model automatically after each restart of the Raspberry Pi hardware. For more information on how to enable the run-on-boot feature, see [Automatically Run Simulink Model on Raspberry Pi After Restart \(Simulink Support Package for Raspberry Pi Hardware\)](#).

Better audio quality using Simulink Support Package for Raspberry Pi Hardware

- Acquire and generate multichannel audio data using Advanced Linux Sound Architecture (ALSA) blocks: Use the ALSA Audio Capture and ALSA Audio Playback blocks to acquire and generate multichannel audio data on audio devices connected to the Raspberry Pi hardware.
- Play high-quality audio from Raspberry Pi using I2S-based digital-to-analog convertor (DAC): You can use the I2S-based DAC with the ALSA Audio Playback block to play high-quality audio from Raspberry Pi. For more information, see [Play High-Quality Audio from Raspberry Pi Using I2S-Based DAC \(Simulink Support Package for Raspberry Pi Hardware\)](#).

Support added for Raspberry Pi Zero W board

Simulink Support Package for Raspberry Pi Hardware now supports building Simulink models and generating code for the Raspberry Pi Zero W board.

You can use all the blocks from the Simulink Support Package for Raspberry Pi Hardware with the Raspberry Pi Zero W board. You can also communicate with Raspberry Pi Zero W using Wi-Fi or a micro-USB cable connection.

Support added for ThingSpeak Read block on Raspberry Pi

You can now use the ThingSpeak Read block to read data from the ThingSpeak channel on the ThingSpeak website. For more information, see [Publish or Retrieve Data to Internet of Things Using ThingSpeak \(Simulink Support Package for Raspberry Pi Hardware\)](#).

Measure data using LEGO EV3 Compatible NXT Sensor blocks

You can now measure data from the LEGO NXT sensors connected to the LEGO MINDSTORMS EV3 brick using these blocks:

- NXT Touch Sensor: Reads the state of the NXT touch sensor.
- NXT Ultrasonic Sensor: Measures the distance between the NXT ultrasonic sensor and the nearest object in front of the sensor.
- NXT Gyro Sensor: Measures the rate of rotation (yaw) along the single axis of the HiTechnic NXT gyro sensor.
- NXT Sound Sensor: Reads the sound level measured by the NXT sound sensor.

Measure data using LEGO EV3 blocks

You can now read data from the LEGO MINDSTORMS EV3 brick using these blocks:

- Battery: Measures the voltage of the battery located in the EV3 brick.
- Timer: Measures the elapsed time from when the EV3 brick starts running the Simulink model.

Flexibility in Android support package installation

Before R2018a, installation of Simulink Support Package for Android Devices included installation of Android platform packages and Software Development Kit (SDK) tools. As of R2018a, the installations of the Android platform packages and SDK tools can be done independently after a successful installation of the Android support package. This change in the installation process shortens the time required to install the support package. For more information, see [Hardware Setup \(Simulink Support Package for Android Devices\)](#).

Accept data from Android keyboard

You can use the Data Input block to create a data input field on an Android device screen. This input field accepts data from the keyboard of the device.

Measure data using Android sensor blocks

You can use the Magnetometer block and Orientation block to measure data from the built-in sensors on an Android device.

- Magnetometer: Measures the magnetic field along the x , y , and z axes of an Android device.
- Orientation: Measures device rotation along the x , y , and z axes of an Android device.

Communicate with Arduino during Normal mode simulation using Simulink IO

Simulink Support Package for Arduino Hardware supports Simulink IO during Normal mode simulation. When you simulate a Simulink model in Normal mode with Simulink IO, the model communicates with the IO peripherals on the hardware, enabling you to verify the model design before deploying it on the hardware.

MATLAB Function Blocks

Redesigned MATLAB Function Report: View information in new user interface with improved navigation

In R2018a, the MATLAB Function report has a new user interface with improved navigation.

The screenshot displays the MATLAB Function Report interface. The main window shows the MATLAB source code for a function named `stats`. The code includes comments, a function definition, and several lines of MATLAB code for calculating mean and standard deviation. Below the code, there is a table titled "ALL MESSAGES (0)" and "VARIABLES" which lists the variables used in the function.

Name	Type	Size	Class
mean	Output	1 × 1	double
stdev	Output	1 × 1	double
vals	Input	4 × 1	double
len	Local	1 × 1	double

See MATLAB Function Reports.

Sparse Matrices: Enable more efficient computation in a MATLAB Function block

In R2018a, you can generate code for sparse matrices and sparse matrix operations in a MATLAB Function block. Sparse matrices provide efficient storage for arrays with many zero elements. In simulation and in generated code, sparse matrices can improve performance and reduce memory usage compared with full matrices. Computation time on sparse matrices scales only with the number of operations on nonzero elements.

In R2018a, code generation support focuses on sparse matrix construction, arithmetic, concatenation, indexing, and some element-wise functions. See Code Generation for Sparse Matrices.

Cell Array Support: Import cell arrays into a MATLAB Function block by using `coder.load` and `load`

In previous releases, you could not import cell arrays from saved data into a MATLAB Function block. In R2018a, cell arrays are supported for loading. For example, in the Command Window, save a cell array in the `data.mat` file:

```
x = {1, 'a', 3};
save('data.mat', 'x');
```

Write a function `foo` that imports the cell array data into a MATLAB Function block. Use the `load` function to import data during simulation. For code generation, use `coder.load` instead of `load`.

```
function [out1, out2] = foo
    % for simulation
    runTimeData = load('data.mat');
    out1 = runTimeData.x{3};
    % for code generation
    compileTimeData = coder.load('data.mat');
    out2 = compileTimeData.x{3};
end
```

Delete Method: Call clean-up code automatically when handle classes are deleted

In 2018a, you can use `delete` methods (destructors) for handle classes in MATLAB code in a MATLAB Function block. To perform clean-up operations, such as closing a previously opened file before an object is destroyed, use a `delete` method. For the guidelines and restrictions on using `delete` methods in a MATLAB Function block, see [Code Generation for Handle Class Destructors](#).

Version History

In 2017b, you could explicitly call the `delete` method of the handle superclass, but not of any derived class. In 2018a, you cannot explicitly call `delete` methods in a MATLAB Function block. The generated code automatically calls them. If you attempt to explicitly call a `delete` method, you get an error.

Statistics and Machine Learning Toolbox Code Generation: Generate code for distance calculation on vectors and matrices, and for prediction by using k-nearest neighbor with Kd-tree search and nontree ensemble models

The following functions support code generation:

- `grp2idx` — Create an index vector from a grouping variable.
- `pdist` — Find the pairwise distance between pairs of observations.
- `squareform` — Format a distance matrix.

When you train a *k*-nearest neighbor classification model by using `fitcknn` for code generation, you can now use the Kd-tree search algorithm. For more details, see the [C/C++ Code Generation](#) section of the `ClassificationKNN` class.

When you find nearest neighbors by using the functions `knnsearch` and `rangesearch` and the object functions `knnsearch` and `rangesearch` for code generation, you can now use the Kd-tree search algorithm.

When you train an ensemble by using `fitcensemble` for code generation, you can now specify 'discriminant' or 'knn' as weak learners by using the 'Learners' name-value pair argument. For more details, see the C/C++ Code Generation section of the `CompactClassificationEnsemble` class.

Code generation for MATLAB `sprintf` function

In R2018a, you can generate code from your MATLAB code that uses `sprintf` to construct formatted strings and character arrays.

Code generation for MATLAB sort function options

In R2018a, code generation supports additional options for MATLAB sort functions:

- Code generation of `issorted`, `issortedrows`, `sort`, and `sortrows` supports the `ComparisonMethod` and `MissingPlacement` options.
- Code generation of `issorted` and `issortedrows` supports the 'monotonic', 'strictascend', 'strictdescend', and 'strictmonotonic' values for sorting direction.
- Code generation of `issortedrows` supports specification of the sorting direction as a cell array of character vectors.

Code generation support for `issortedrows` is new in R2018a.

Code generation for more MATLAB functions

- `maxk`
- `mink`
- `nzmax`
- `rescale`
- `spalloc`
- `sparse`
- `spdiags`
- `speye`
- `spfun`
- `spones`
- `vecnorm`

Code generation for more Audio Toolbox functions

- `cepstralFeatureExtractor`
- `splMeter`
- `voiceActivityDetector`

Code generation for more Communications System Toolbox functions

- `apskdemod`
- `apskmod`
- `dvbsapskdemod`
- `dvbsapskmod`
- `mil188qamdemod`
- `mil188qammod`
- `tpcdec`
- `tpcenc`

Code generation for more DSP System Toolbox functions

- `dsp.ComplexBandpassDecimator`

S-Functions

S-functions support interleaved complex data

Starting in R2018a, S-functions support interleaved storage representation of complex MATLAB data. This change does not affect the functionality of your existing S-functions. However, to take advantage of the interleaved complex representation of S-function parameters and other MATLAB data, consider upgrading your S-functions. This upgrade may improve the efficiency of S-functions interfacing with the C/C++ code that expects interleaved complex data. For more information on `mxArrays` using interleaved complex, see `mxArrays Using Interleaved Complex Representation`. See `MATLAB Support for Interleaved Complex API in C MEX Functions` to learn more about this change.

To check how to upgrade your code, see `Do I Need to Upgrade My MEX Files to Use Interleaved Complex API? (MATLAB)`.

`simstruc.h` is restructured to include only documented SimStruct APIs

The `simstruc.h` header file is restructured to include only documented SimStruct APIs.

C-MEX S-functions built without using the `mex` command

Starting R2018a, your C-MEX S-functions built without using the `mex` command may not compile or work as expected if processing complex MATLAB data in the following ways:

- Use `mxGetPi` or `mxGetImagData` in your source code.
- Call `mxGetPr` on a complex `mxArray`.
- Treat `mxGetData` or `mxGetElementSize` output as real data, as opposed to interleaved.

C-MEX S-function built without using the `mex` command in R2017b or prior releases will continue to work. However, before re-compiling your S-function for R2018a or later, ensure that your build instructions are updated according to the information in `Compiling MEX files without the mex Command`. To check your model for S-function upgrade issues, use **Model Advisor > Upgrade Advisor > Check model for S-function upgrade issues**. For more information on this check, see `Check model for S-function upgrade issues`. It is recommended that you re-run the Upgrade Advisor check after updating your build instructions and rebuilding your S-functions.

R2017b

Version: 9.0

New Features

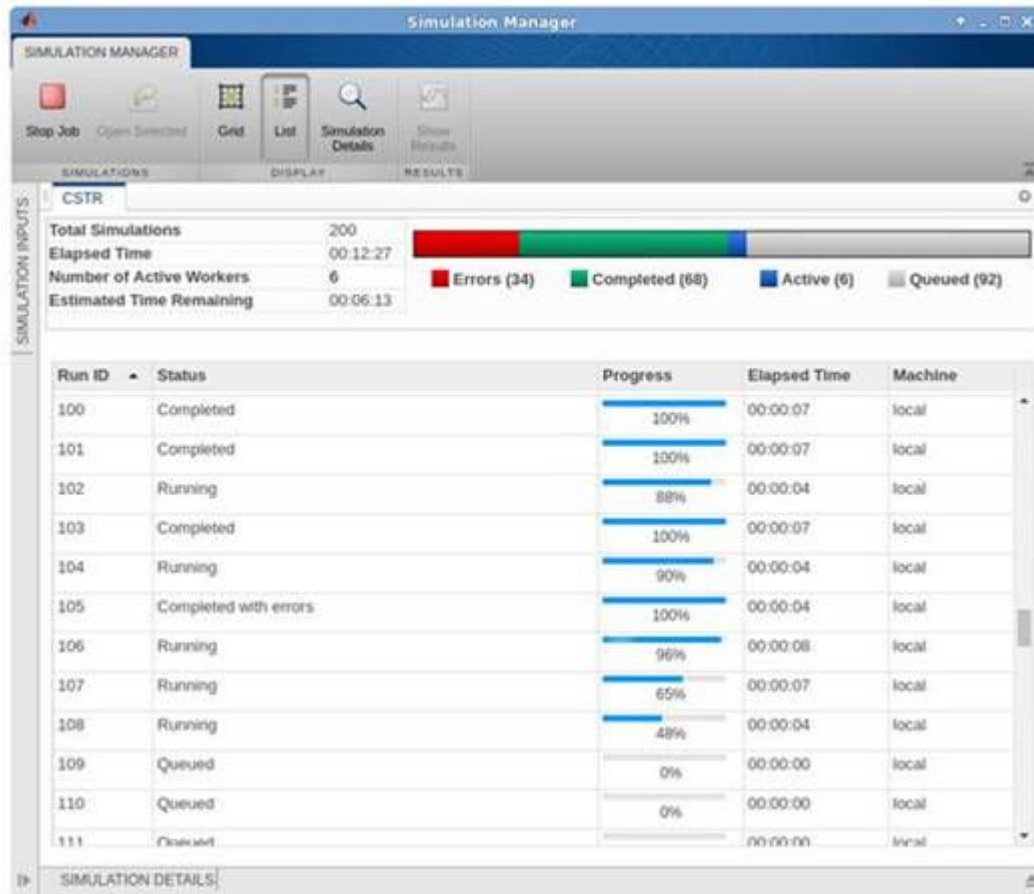
Bug Fixes

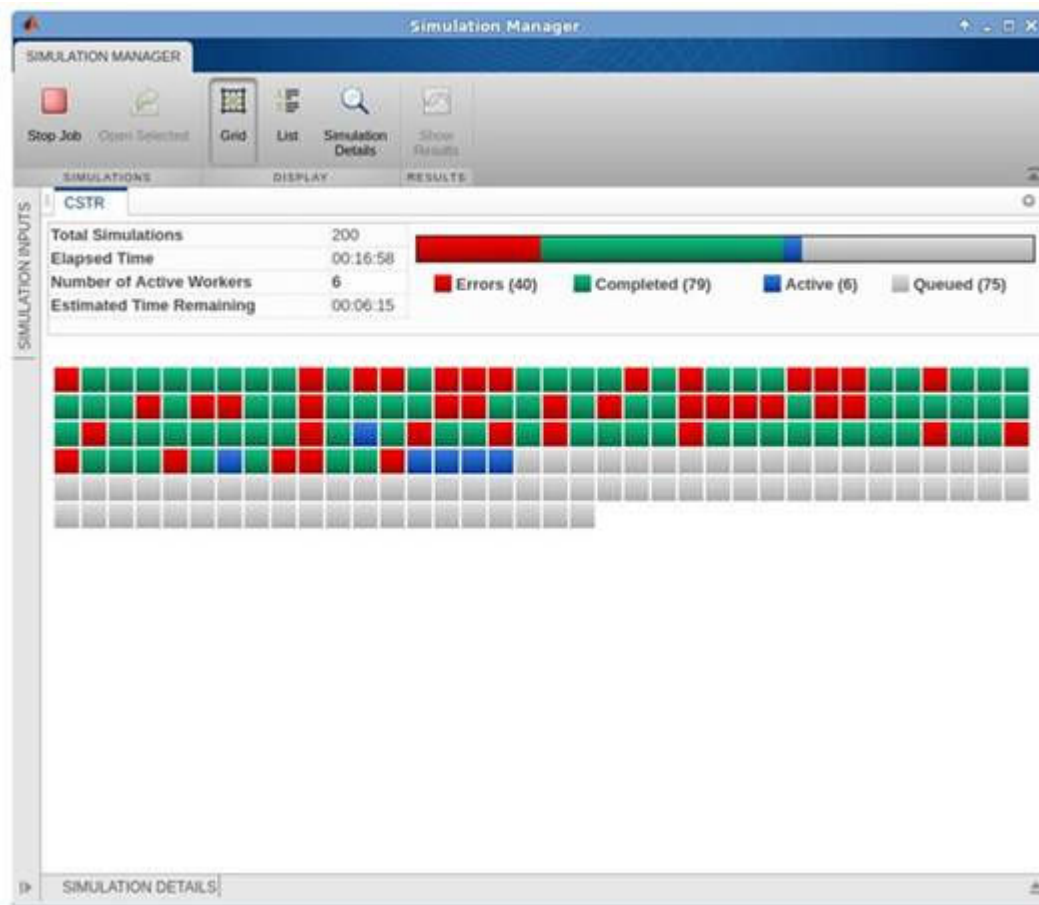
Version History

Simulation Analysis and Performance

Simulation Manager: Monitor, inspect, and visualize simulation progress and results

The Simulation Manager allows you to visually monitor multiple simulations. You can now see the progress of all simulations. The Simulation Manager points out the simulations that error out by highlighting them in red. This UI also gives you information about the total number of simulations, the elapsed and remaining time, and the number of active workers in case of parallel simulations. The UI provides a high-level grid view and a detailed list view of simulations. You can select a simulation run and load it on your client machine. You can also view the specifications of each simulation and apply those specifications to the model. Through the Simulation Manager app, you can view the simulation results in the Simulation Data Inspector.





Enhanced Simulation Data Inspector UI and API

Compact toolbars in the streamlined Simulation Data Inspector user interface provide quicker access to tools and settings. You can easily navigate comparison results to find out of tolerance regions using **Previous** and **Next** buttons added to the **Compare** view.

Additions to the Simulation Data Inspector API support workflows with large data sets and improve usability. For more information, see [Inspect and Analyze Simulation Results](#).

Logging Improvements for the Simulation Data Inspector

In R2017b, the Simulation Data Inspector supports live streaming of root-level outputs in Simulink when you set the **Format** to **Dataset**. The Simulation Data Inspector also offers improved support for Stateflow and the Parallel Computing Toolbox.

New Simulation Control and Visualization Blocks in the Dashboard Library

You can create intuitive and interactive displays using Dashboard blocks. New blocks include:

- Check Box

- Combo Box
- Radio Button
- Edit
- Display
- Callback Button

For more information, see [Control Simulations with Interactive Displays](#).

Solver Profiler supports fixed step solvers

Solver Profiler now supports fixed-step solvers. You can now obtain relevant solver information when analyzing simulation performance for models utilizing fixed-step solvers.

Configure Solver Profiler from the command line

Run Solver Profiler from the command line

The `solverprofiler.profileModel` function allows you to configure and open Solver Profiler from the command line. This example demonstrates how to use the function to profile the vdp model.

```
model = 'vdp';
file = solverprofiler.profileModel(model, ...
    'SaveStates' , 'On' , ...
    'SaveSimscapeStates' , 'On' , ...
    'SaveJacobian' , 'On' , ...
    'StartTime' , 5, ...
    'StopTime' , 20, ...
    'BufferSize' , 1000, ...
    'TimeOut' , 5, ...
    'OpenSP' , 'on');
```

Profile models in parallel

It is now possible to run parallel and batch profiling operations. This example shows how to profile the vdp and f14 models in parallel using the function.

```
models = {'vdp' , 'f14'};
files = cell(1,2);
parfor i=1:length(models)
    files{i} = solverprofiler.profileModel(models{i}, ...
        'SaveStates' , 'On' , ...
        'SaveSimscapeStates' , 'On' , ...
        'SaveJacobian' , 'On' , ...
        'StartTime' , 5, ...
        'StopTime' , 20, ...
        'BufferSize' , 1000);
end
solverprofiler.exploreResult(files{1});
```

Reduce memory usage when linearizing large models

You can reduce memory usage when linearizing large models by enabling the parameter `JacobianLinearizationMemoryReduction` before you run `linearize`, `trim`, or other linearization operations. Use `set_param` to enable the parameter.

```
set_param mdl, 'JacobianLinearizationMemoryReduction', 1)
```

Simulink Cache: Generate Simulink cache files for top models in accelerator mode

For R2017b, Simulink cache (`.slxc`) files are generated for top-level models run in accelerator mode, if necessary. This accelerator mode support extends the R2017a behavior, where updating a diagram or running a simulation on a new model that builds a model reference SIM target or rapid accelerator target creates a Simulink cache file.

slsfnagctrl function not supported to report diagnostics programmatically

From R2017b, the `slsfncgctrl` function is not supported to report the user-defined diagnostics programmatically. You can use the `sldiagviewer` instead. For more information, see [Report Diagnostic Messages Programmatically](#).

Suppress immaterial diagnostic warnings and errors from specific blocks

You can now suppress certain diagnostics that are treated as errors for specific objects in your model. In past releases, only warning diagnostics were supported for suppression.

Click the **Suppress** button next to the error or warning in the Diagnostic Viewer to suppress the diagnostic from the specified source. You can restore the diagnostic from the source by clicking the **Restore** button.

You can also configure suppressions from the command line. For more information, see `Simulink.suppressDiagnostic` and `Simulink.restoreDiagnostic`.

Comprehensive run-time diagnostics for wrapping and saturating overflows from Stateflow and MATLAB Function blocks

The Diagnostic Viewer now reports overflows due to wrap and saturation that occur within a MATLAB Function block or in a Stateflow chart that uses MATLAB as the action language. In cases of overflows that occur within a MATLAB Function block, the diagnostic includes the line number at which the overflow occurred.

You can suppress and restore these diagnostics at the block level by clicking the **Suppress** and **Restore** buttons respectively in the Diagnostic Viewer.

Simulink Editor

Hidden Block Names: Improve model appearance by hiding default block names

In R2017b, block names given by the Simulink Editor are hidden by default in new and existing models. For example, by default, the block names Gain1, Gain2, Gain3, and so on given by the Simulink Editor do not appear in the model. To turn off the option that hides these names, select **Display** and clear the **Hide Automatic Names** check box.

For more information, see [Manage Block Names](#). For an example that shows the effects, see [Build and Edit a Model in the Simulink Editor](#).

Signal Tracing: Highlight and navigate a signal from its source to a destination

You can now operate the Signal Highlighting tool using the keyboard. The left and right arrow keys move the signal trace towards the source and destination, respectively. R2017b also provides these improvements:

- Propagation of the signal trace to a subsystem does not automatically step into the subsystem except in cases of a discontinuity in the diagram, such as a From block inside the subsystem. Pressing the arrow key in the direction of the trace steps into the highlighted subsystem. However, you will not be able to step over a subsystem while traversing it. The signal trace must step into the subsystem and through its path before stepping back out.
- If a trace can take multiple possible paths in the advancing direction, the options are highlighted in blue. Select the option using the Up and Down arrow keys.
- Display the port values for the highlighted signals by pressing the P key on the keyboard.
- If the trace steps out of a subsystem block, the subsystem crossfades to tell you where the trace stepped out.

For more information on how to use this feature, see [Highlight Signal Sources and Destinations](#)

Autoroute Lines: Select blocks and multiple lines for improved line routing

You can now select multiple lines for improved line routing in a model. In addition, you can select a block to improve the routing of the lines that connect to that block. In earlier releases, you selected only a single line for automatic routing around blocks.

Routing of lines is also improved when connecting model elements that have multiple ports to each other, such as subsystems. Select one subsystem and **CTRL+click** the subsystem you want to connect to. The connecting lines route around other model elements, making the best use of the space.

You can also improve routing when adding lines programmatically. For details, see [add_line](#).

Create Subsystem: Intuitive port placement and naming

Creating a subsystem from selected model elements using **Create Subsystem** now places ports in the subsystem at intuitive orientations. In addition, if the selection includes a named signal as the

input or output, creating the subsystem gives the corresponding port the name of the signal. For an example, see [Organize Your Model](#).

This capability also applies to ports added to subsystems by dragging a signal to a subsystem. If the signal you drag to a subsystem has a name, the new port gets the name of the signal.

Version History

The change in how subsystem ports are named can affect scripts that expect new inport and outport block names of In1, In2, and so on. Instead of using an explicit name, use `find_system` to get the block name. For example:

```
open_system('vdp');
h = get_param('vdp/x2','handle');
Simulink.BlockDiagram.createSubsystem(h);
newout = find_system('vdp/Subsystem1','SearchDepth',1,'BlockType','Outport','Port','1')

newout =

    1x1 cell array

    'vdp/Subsystem1/x1'
```

Double-Click Action: Add blocks or annotations on the canvas

To add blocks to a model using an in-canvas shortcut, you can now double-click to start the interaction. In previous releases, you used a single click. Double-click the canvas where you want to create the block and start typing the block name. For an example, see [Add More Blocks in Build and Edit a Model in the Simulink Editor](#). You can continue to use a single click and then start typing the block name to add a block.

In previous releases, you double-clicked the canvas to create an annotation. Now when you double-click, you can select **Create Annotation** from the menu. For more information, see [Describe Models Using Annotations](#).

Searching Models: Cancel find and use search history across models

You can now cancel model searches started with **Edit > Find**. In previous releases, you had to wait until the search was complete, which can take a long time. Now you can interrupt a search, for example, when you want to refine your search after seeing some results. In the detailed search view, click the **Stop Search** button.

In addition, you can now perform other actions in the Simulink Editor while a search occurs. For example, you can add blocks to the model, enter commands at the MATLAB command prompt, and start a search in another model. In earlier releases, you had to wait for the search to complete.

Searching also now uses your search history in all open models in the current Simulink session. As you enter the search string, previous searches that match appear in a list. In earlier releases, each model had its own search history.

Exiting Simulink clears the search history. For more information on searching, see [Find Model Elements in Simulink Models](#).

You can also cause a programmatic search to stop after it finds a match. Use the `FirstResultOnly` option with `find_system`.

Simulink API: Column vector and matrix values are not supported in some functions

In previous releases, entering a column vector or matrix as a value to `new_system`, `open_system`, and `save_system` did not return an error. These value types were not documented. For example:

- `new_system(['abc'])` created a model named `abc`.
- `new_system(['abc'; 'def'])` created a model named `adbecf`.

Now, specifying these values returns an error.

Component-Based Modeling

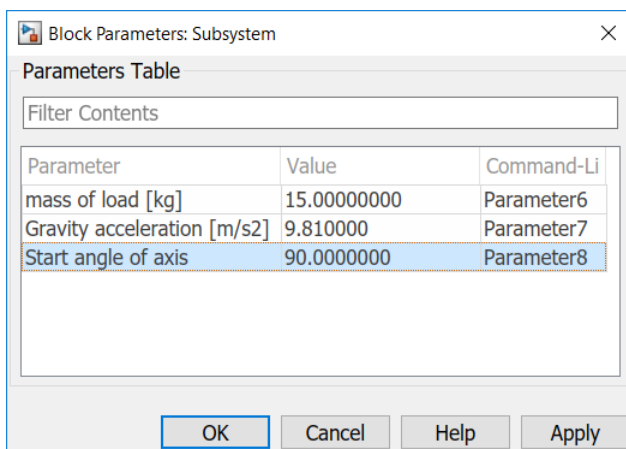
Schedulable Components: Explicitly schedule models for simulation and adaptation to your software environment

With implicit scheduling, you let Simulink decide when a functional component (subsystem or model) is executed. Sample time is inherited from the blocks connected to the component.

With explicit scheduling, you specify the periodic execution of functional components (subsystems and models) by specifying the sample time on Inport blocks. For more information, see Explicitly Schedule Execution of Subsystems and Models.

Tables in Masks: Present and sort your mask parameters in a searchable table

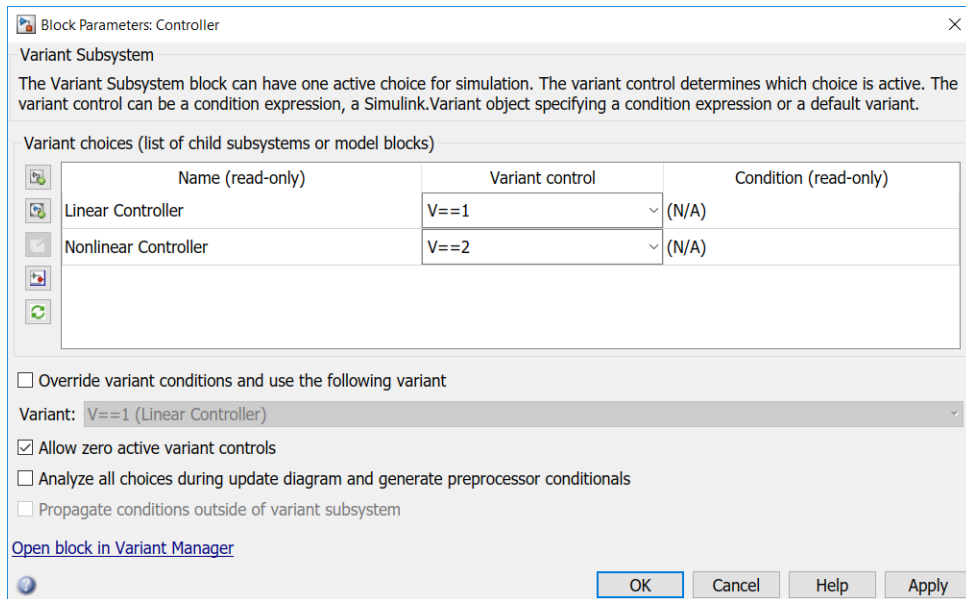
The **Table** control in the Mask Editor enables you to group a large number of parameters in a tabular form. You can also search and sort the content listed within the table.



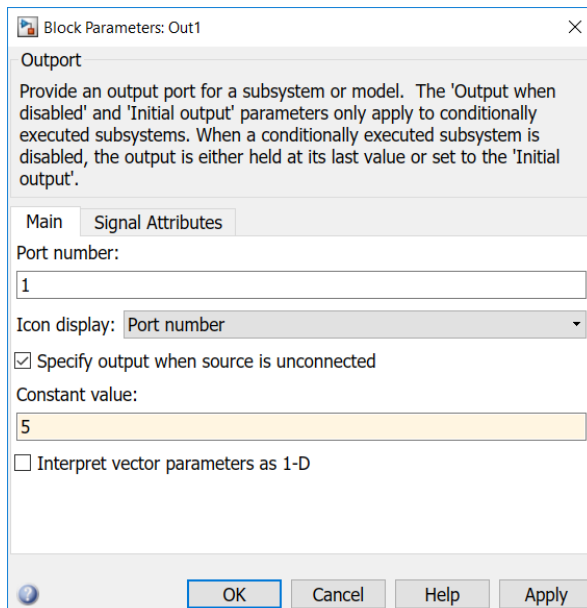
Simulink Variants: Create more customizable variant models by using improved Variant Subsystem and variant condition propagation capabilities

In R2017b, you can use these variant features to develop more flexible design:

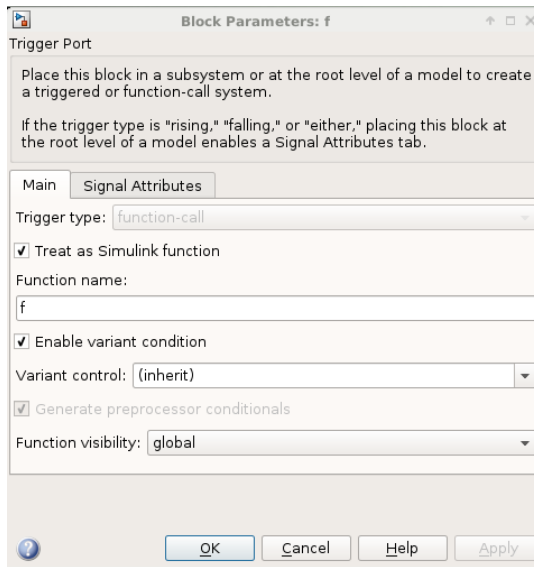
- **Allow zero active variant controls:** When this option is selected on a Variant Subsystem block, you can simulate a model with a Variant Subsystem block without an active variant choice.



- **Specify output when source is unconnected:** When this option is selected on a non-driven Outport block of a Variant Subsystem block, you can specify a constant value as an output of that block.



- **Inherit variant condition from callers:** When this option is selected on a Trigger port block, the Simulink Function block containing the Trigger port block inherits the variant conditions from the corresponding Function Caller blocks available in the model.



Variant Reducer Enhancements

- Before R2017b, the removal of inactive variants led to an excessive number of signal bends and white spaces in reduced models. Reduced models now have fewer signal bends and white spaces.
- Variant Reducer now allows you to preserve compiled signal attributes between the original and reduced models by adding signal specification blocks in the reduced model. For more information, see `Simulink.VariantManager.reduceModel`.

Variant Systems: Convert Model blocks with model variants to Variant Subsystem blocks

R2017b introduces two approaches to convert a Model block that contains variant models to a Variant Subsystem block that contains Model blocks that reference the variant models. Use of a Variant Subsystem block provides these advantages:

- Allows you to mix Model blocks and subsystems as variant systems
- Supports flexible I/O, so that all variants do not need to have the same number of input and output ports

For new models, use a Model block for model variants only if you need to use variants that are conditionally executed models (models with control ports). Using a Model block for variant models is supported for backward compatibility. However, support for using a Model block to contain model variants will be removed in a future release. For an example of a model that uses a Variant Subsystem block as a container for variant models, see [Model Reference Variants](#).

To convert a Model block that contains variant models to a Variant Subsystem block that contains Model blocks that reference the variant models, use one of these approaches:

- Right-click the Model block and select **Subsystems & Model Reference > Convert to > Variant Subsystem**.
- Use the `Simulink.VariantManager.convertToVariant` function. Specify the Model block path or block handle.

Tip Starting in R2017b, to make it easier to set up model variants, you can use the new Variant Model block, which is a subsystem variant template block preconfigured to contain two Model blocks that to use as variant systems.

Version History

In R2017b, there is no longer a Model Variants block in the Simulink library. You can still use the Model block to contain model variants.

If you convert a Model block to a Variant Subsystem block, note that the behavior of the Model block parameter **Generate preprocessor conditionals** is different than the Variant Subsystem block parameter **Analyze all choices during update diagram and generate preprocessor conditionals**. For Model blocks that contain model variants, enabling the parameter causes simulation and update diagram to compile the active variant only. For Variant Subsystem blocks, enabling the parameter compiles all the variants, which can make simulation and updates slower.

Converting model variants to subsystem variants can require that you update scripts that use the `Variants` command-line parameter.

Signal attribute display at load and edit time

In previous releases, pressing **Ctrl+D** displayed signal attributes. At model load time and edit, Simulink models now display these signal attributes on the signal line in the interface display if the model property `ShowPortDataTypes` is enabled. You do not compile the model first.

- Data type
- Complexity

In addition, if the `ShowLineDimensions` property is enabled, signal dimensions display.

These blocks support the capability:

- Inport
- Outport
- Subsystem
- Model
- Signal Specification
- Bus Creator

Displaying signal attributes when editing the model and at load time is helpful when working with multiple model components in large models. For more information, see [Display Signal Attributes at Model Load Time](#).

The display of units at load time has been supported since R2016a.

Bus Signals: Learn about buses for simpler diagrams, with productivity tips

R2017b includes a new Buses in Simulink example that gets you started with these bus signal capabilities:

- Bus Creator, Bus Selector, and Bus Assignment blocks
- Bus element ports at subsystem interfaces
- Smart editing to perform common bus workflows quickly

The example includes animations to illustrate model editing techniques.

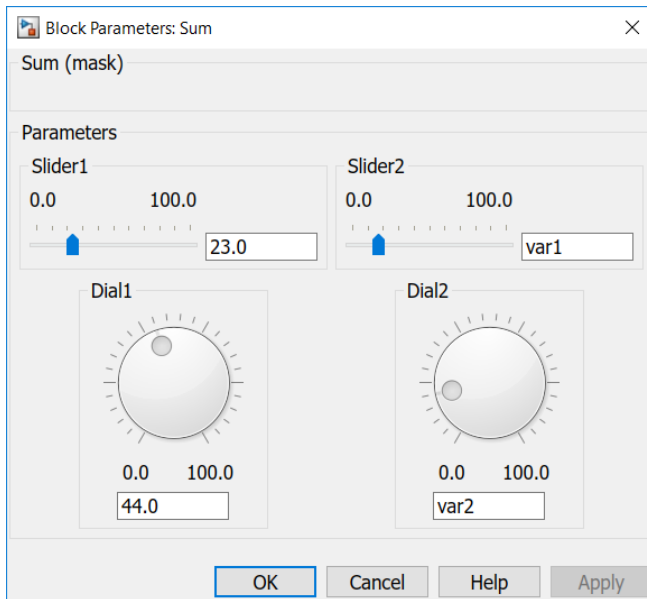
slreplace_mux function not supported

Functionality	Result	Use Instead	Compatibility Considerations
The function <code>slreplace_mux</code> used to replace Mux blocks used to create bus signals with Bus Creator blocks.	Error: Undefined function or variable 'slreplace_mux'.	In R2017b and future releases, Mux blocks cannot create bus signals. For models created in earlier releases, either replace the Mux blocks with Bus Creator blocks manually, or run the Upgrade Advisor with the Check Mux blocks that create bus signals check enabled.	In your scripts, remove instances of the <code>slreplace_mux</code> function.

Tune Slider and Dial using variables or tune variables from Slider and Dial

From R2016a, you were able to use variables to specify values for the **Slider** and **Dial** parameters.

In R2017b, you can continue to specify values using variables for the **Slider** and **Dial** parameters. Additionally, you can also tune the base workspace variable value through the **Slider** and **Dial** parameter.



Update library links programmatically

From R2017b, the links to a library block from a model can be modified or updated programmatically by using the `set_param` command. For more information, see [Linked Block Information](#).

Project and File Management

Simulink Project Upgrade: Update all models and library blocks used in your project to the latest release

Use Project Upgrade to easily update all models and linked library blocks to the latest release.

On the Simulink Project tab, click **Run Checks > Upgrade**.

The project automatically runs all upgrade checks on multiple libraries, including any checks that require an **Update Diagram**.

Project upgrade helps you with the management overhead of library links. Linked library blocks inherit attributes from the surrounding models, such as data types and sample rate. The block behavior can differ depending on the context where the block is used. Without project upgrade, you manually have to handle the state of each library link and upgrade each block from the model. As a result, you can inadvertently overwrite previous upgrades as you iterate through the models that use the block. If models require a different upgrade of the same library block, you are prompted to view and resolve the upgrade conflict.

For details, see Upgrade All Project Models and Libraries.

Automatic Project Creation: Easily turn a folder into a project and manage your files, data, and environment in one place

Easily convert a folder of files into a Simulink project by using the **Folder to Project** template in the Simulink start page. The template automatically adds your files to the project and prompts you to set up the path and startup files. This simple, quick process sets up your project to manage your files and introduces you to project features. When you open the project, it automatically puts the folders you need on the path, and runs any setup operations to configure your environment. Startup files automatically run (.m and .p files), load (.mat files), and open (Simulink models) when you open the project.

For details, see Create a New Project From a Folder.

Model Compare and Merge: Identify differences between model elements, Stateflow charts, and MATLAB Function blocks

Use model comparison to review and merge changes in models, display changes in original models, and filter and save comparison results.

For details, see Model Comparison.

Automate Comparison Reports: Compare models and generate reports programmatically

You can now programmatically generate comparison reports without opening the Comparison Tool. Use the new output from the `visdiff` function to manipulate the comparison at the command line. You can apply filters and publish comparison reports to files. See `visdiff` (MATLAB).

Programmatic Project Setup: Manage startup and shutdown files

Use functions to set up or remove startup and shutdown files. Startup files automatically run (.m and .p files), load (.mat files), or open (models) when you open the project. See `addStartupFile`, `addShutdownFile`, `removeStartupFile`, and `removeShutdownFile`.

Setting up startup and shutdown files interactively is also simplified in Simulink Project. You no longer have to create shortcuts to set startup and shutdown files. See [Automate Startup Tasks](#).

If you convert a folder to a new project, you are prompted to set up startup and shutdown files. See “[Automatic Project Creation: Easily turn a folder into a project and manage your files, data, and environment in one place](#)” on page 10-15.

Missing Product Identification: Find and install required products for project templates

In the Simulink start page, project templates created in R2017b or later warn you if required products are missing. Click the links to open Add-On Explorer and install required products. Create a [New Project Using Templates](#).

Project Shortcuts Toolstrip: Simplified workflow for setting up shortcuts for frequent tasks

You can now use the Project Shortcuts tab to create and manage shortcuts and shortcut groups. The Shortcut Management view has been removed. See [Create Shortcuts to Frequent Tasks](#).

Compare Git Branches: Show differences from parent files and save copies

In a project under Git source control, you can now show differences to parent files and save branches in the Branches dialog box. If you want to examine added or deleted files, or to test how the code ran in previous versions, you can save a copy of the selected or parent files. See [Branch and Merge Files with Git](#).

Data Management

Model Data Editor: Easily view, filter, group, and edit more data used by a model including signals, states, and referenced variables

In R2017b, the Model Data Editor (**View > Model Data** in the Simulink Editor) shows additional information:

- Properties for internal signal lines, which do not participate in root-level input and output. Use the existing **Signals** tab.
- Properties for block states. Use the new **States** tab.
- Properties for workspace variables and objects (such as `Simulink.Parameter` and `Simulink.Signal`) that the model uses. Showing this information requires updating the block diagram.
- Compiled information for signals (on the **Inports/Outports** and **Signals** tabs). For example, if you set the data type of an Inport block to `Inherit: auto` (the default setting), the compiled data type is the one that the block ultimately uses during simulation and in the generated code. Showing compiled information requires updating the block diagram.
- Properties for Data Store Read and Data Store Write blocks. Use the existing **Data Stores** tab.

By default, as you select blocks and signals in the model, the Model Data Editor highlights the corresponding rows in the data table. To show only these rows by filtering the table, toggle the new **Filter by Selection** button.

For all tabs, the **Block** column has a new name, **Source**. On the **Parameters** tab, the **Parameter** column has a new name, **Name**. Some tabs now have a slightly different column arrangement.

Signal Editor: Create and edit input signals that can be organized for multiple simulations

Use the Signal Editor user interface to create and edit input signals that you can organize for multiple simulations. You can access the Signal Editor in the following ways:

- `signalEditor` function
- From the Root Inport Mapper
- From the new Signal Editor block

The changes you can make using the Signal Editor include::

- Changing the bus hierarchy and signal order.
- Creating input signals from root input ports.
- Plotting signals and view data.
- Adding and edit data for signals.
- Opening MAT-files exported from the Signal Builder block.

For more information, see [Create and Edit Signal Data](#).

Defining Missing Variables: Identify and easily fix missing, deleted, or renamed variables

In R2017b, if a variable or object (such as a numeric MATLAB variable or a `Simulink.Parameter` object) is not available to a model when you update the block diagram, the Diagnostic Viewer offers you options for fixing the resulting error. For example, you can create a new variable, select a MAT-file to load or a script file to run, or recover a deleted variable. Storing variables in a data dictionary can afford you additional options for remediation.

Lookup table class supports even spacing

The `Simulink.LookupTable` class now supports using even spacing to generate evenly spaced breakpoints. Use even spacing to generate evenly spaced breakpoints.

The `BreakpointsSpecification` property now provides the `Even spacing` option. You can find the `Even spacing` option in the **Breakpoints > Specification** drop-down list of the lookup table property dialog box.

Lookup table object structures

You can now change the order of the size, breakpoints, and table entries in a lookup table object-generated structure. To change the order of the table entries, on the Configuration Parameters dialog box, set these parameters:

- **LUT object struct order for even spacing specification** — Select the order `Size,Breakpoints,Table` or `Size,Table,Breakpoints`. This parameter applies when the **Specification** parameter of the `Simulink.LookupTable` object is set to `Even spacing`.

Command line: `LUTObjectEvenSpacingStructOrder`

- **LUT object struct order for explicit value specification** — Select the order `Size,Breakpoints,Table` or `Size,Table,Breakpoints`. This parameter applies when the **Specification** parameter of the `Simulink.LookupTable` object is set to `Explicit values`.

Command-line: `LUTObjectExplicitValueStructOrder`

Root Inport Mapper update

The Root Inport Mapper now supports the MATLAB `timetable` function.

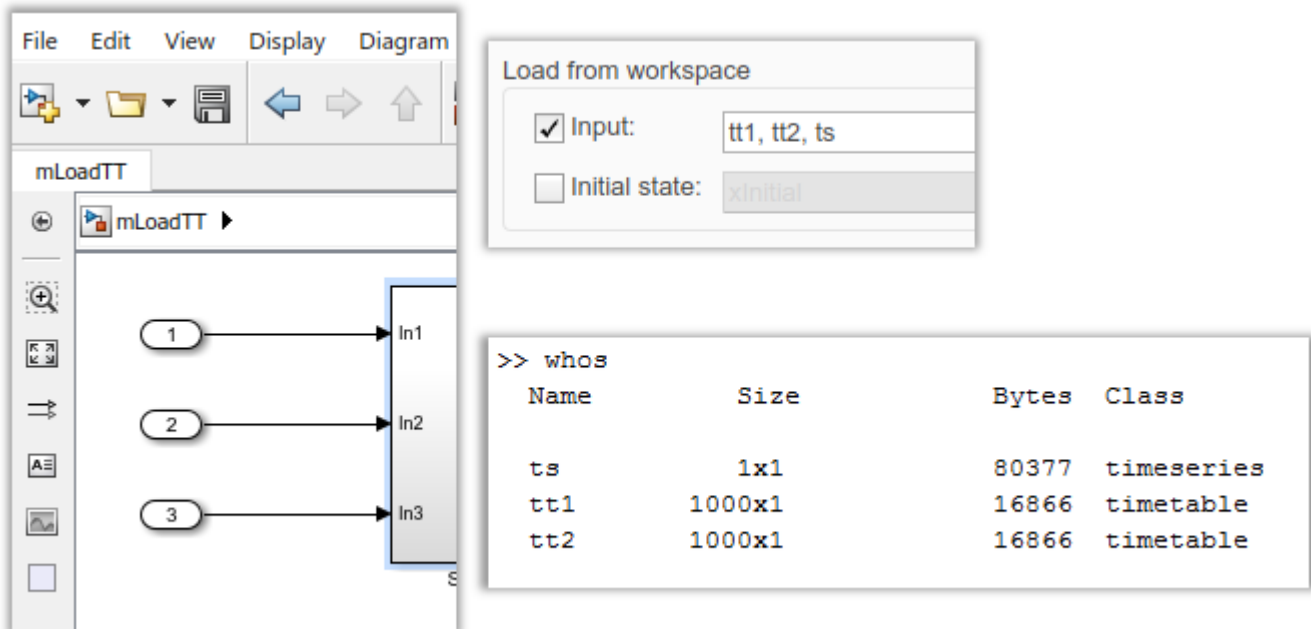
The Root Inport Mapper **Generate MATLAB Script** button now generates scripts that use the `Simulink.SimulationInput` object and `parsim` function. Scripts generated in previous releases remain functional.

Signal Loading: Load timetable data into Simulink models

You can load `timetable` signal data to a root-level Inport block. You can use a `timetable` object for signal loading the same way that you use a `timeseries` object.

- In the **Input** configuration parameter, you can use a `timetable` object either by itself or in a comma-separated list.

- As a leaf node of a `Simulink.SimulationData.Dataset` object, you can specify a timetable object.

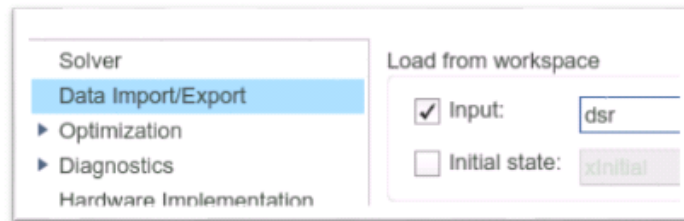
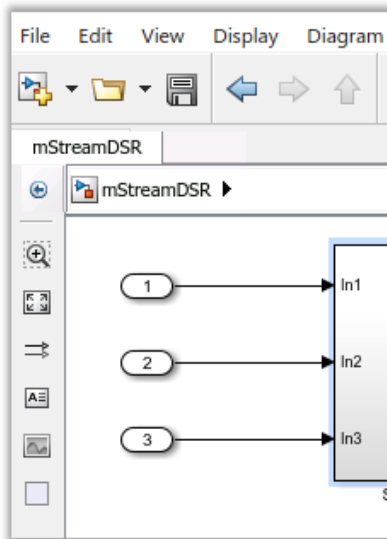


The screenshot shows the Simulink environment. On the left, a model diagram titled 'mLoadTT' features three input blocks labeled '1', '2', and '3', each connected to an input port labeled 'In1', 'In2', and 'In3' respectively. On the right, the 'Load from workspace' configuration panel is open, showing the 'Input' field set to 'tt1, tt2, ts' and the 'Initial state' field set to 'xinitial'. Below this, a command window displays the output of the `whos` command:

```
>> whos
Name          Size          Bytes  Class
ts            1x1           80377  timeseries
tt1          1000x1        16866  timetable
tt2          1000x1        16866  timetable
```

Signal Loading: Incrementally stream DatasetRef data from MAT-file to root-level Inport blocks

You can stream signal data into a model directly from a MAT-file referenced by a `Simulink.SimulationData.DatasetRef` object. Streaming involves incrementally loads data into a model, instead of loading the whole data to memory all at once. Use a `DatasetRef` object to reference a dataset in a MAT-file (for example, a MAT-file created while logging to persistent storage).



Current Folder	
Name	Size
inputs.mat	169 KB
inputs.mat (MAT-file)	
Name	Value
ds	Dataset

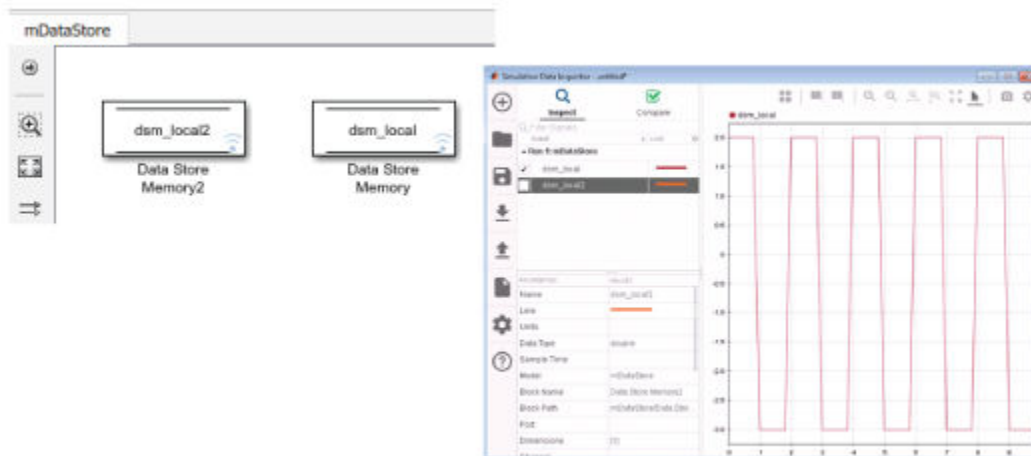
```
>> dsr = Simulink.SimulationData.DatasetRef('inputs.mat','ds')
dsr =
Simulink.SimulationData.DatasetRef
Characteristics:
    Location: inputs.mat (C:\Users\atrubits\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\...
    Identifier: ds

Resolved Dataset: 'my streamed inputs' with 3 elements
```

	Name	BlockPath
1	data for inport 1	''
2	data for inport 2	''
3	data for inport 3	''

Data Store Memory Logging: Stream to MAT-File or Simulation Data Inspector


In normal simulation mode, you can stream data store logging data into a MAT-file (logging to persistent storage) and to the Simulation Data Inspector. You can use this feature with local and global data stores. For details, see [Log Data Stores and Log to Persistent Storage](#).



Variable Editor Access: Launch the Variable Editor from more convenient locations


You use the Variable Editor to inspect and modify structures and arrays that set block parameter values in a model. Before R2017b, you launched the Variable Editor from the Model Explorer **Contents** pane.

In R2017b, you can launch the Variable Editor from these additional locations:

- The Model Explorer **Dialog** pane (the right pane).
- The Model Data Editor (when you choose to display referenced workspace variables).
- The property dialog box of a parameter object (for example, `Simulink.Parameter`) when you open the dialog box from Simulink (for example, by using the action button  next to the **Gain** parameter in a Gain block dialog box).

To launch the Variable Editor, click the action button .

Data Navigation Enhancements: More easily create workspace variables from block parameters

As you set block parameters, signal names, and state names in a model, you can use the nearby action button  to create and navigate to workspace variables. R2017b introduces several enhancements to the action button:

- When you set the value of a block parameter to an expression involving a function, such as the expression $\sin(\text{myVar})+0.27$, you can navigate to the function documentation or, for a custom function, to the source code.
- When you set the name of a signal or block state, you can use the action button to simultaneously create a `Simulink.Signal` object and configure the signal or state name to explicitly resolve to the object.

Previously, you created the object and configured the name resolution in two separate steps.


- When you use the action button to create a variable or object, in the **Create New Data** dialog box, the **Value** drop-down list contains only options that are relevant. For example, for the **Gain** parameter of a Gain block, the drop-down list contains only parameter object options (such as `Simulink.Parameter`).

To further constrain or augment the list of relevant options, select the new `Customize class lists` option.

- When you use the action button to create a variable or object, in the **Create New Data** dialog box, the **Location** drop-down list contains all data dictionaries (`.sldd` files) that are available to the model.

Previously, the list did not include referenced dictionaries.

Button for Lookup Table Objects: Create `Simulink.LookupTable` and `Simulink.Breakpoint` objects in the Model Explorer more quickly

In R2017b, to quickly create `Simulink.LookupTable` and `Simulink.Breakpoint` objects, you can use the new button  on the Model Explorer toolbar. For more information about these objects, see `Simulink.LookupTable` and `Simulink.Breakpoint`.

Shared Local Data Store: Share data between instances of a reusable model

When you break a large system into components with model referencing, you can reuse a component (referenced model) by referring to it with multiple Model blocks.

In R2017b, you can make all instances of a reusable referenced model interact with the same piece of data during a simulation. For example, use this technique to share a communal fault indication between the instances. In the referenced model, represent the data with a Data Store Memory block and select the new block parameter **Share across model instances**. For an example, see `Share Data Between Instances of a Reusable Algorithm`.

Simplified Configuration of Block States: Configure block states by using uniformly named programmatic parameters

Some blocks maintain state data (such as the Unit Delay block). Before R2017b, to programmatically configure the name of the state data, for some blocks, you used the programmatic block parameter `StateIdentifier`. For other blocks, you used `StateName` instead of `StateIdentifier`.

Similarly, to set the initial value of the state, you sometimes used the parameter `X0` and sometimes `InitialCondition`.

In R2017b, these blocks use the same parameter names, `StateName` and `InitialCondition`, so you can write simpler scripts.

Your existing scripts that use the old parameter names continue to work.

Tunable Parameters: Tune parameters in model workspace

Before R2017b, during a simulation, you could not use the Model Explorer **Contents** pane to change the values of variables or parameter objects (such as `Simulink.Parameter`) that you stored in a model workspace. In R2017b, during a simulation, you can change these values by using the **Contents** pane.

During a simulation, you must update the block diagram after you change the value of a workspace variable. For more information about updating the block diagram, see [Update Diagram and Run Simulation](#).

Configuration Parameters Dialog Box: View your model configuration parameters in unified dialog box with search capability

Previously, the Configuration Parameters dialog box contained two tabs: a tab for commonly used parameters and a tab that provided a searchable list of all available parameters. In R2017b, the Configuration Parameters dialog box combines these features in a unified dialog box with a search capability.

- View commonly used parameters on a category pane. Access advanced category parameters on the same pane.
- To quickly find a specific parameter in the dialog box, use the search tool.
- Right-click a parameter to get the parameter name to use in scripts, view parameter dependencies, and navigate to parameter documentation.

For more information, see [Configuration Parameters Dialog Box Overview](#).

Version History

- In R2017b, advanced parameters that were previously available only on the **All Parameters** tab can be found under the **Advanced Parameters** toggle of the relevant category pane. To access this toggle, hover over the ellipsis at the bottom of the pane. Alternatively, to find an advanced parameter, use the search tool at the top of the dialog box.
- If you use an `sl_customization.m` script to hide or disable parameters in the Configuration Parameters dialog box, the script requires updates to widget ID's and callback registrations. For example:

- In R2017a:

```
function sl_customization(cm)

% Disable for standalone Configuration Parameters dialog box.
cm.addDlgPreOpenFcn('Simulink.ConfigSet',@disableRTWBrowseButton)
% Disable for Configuration Parameters dialog box
cm.addDlgPreOpenFcn('Simulink.RTWCC',@disableRTWBrowseButton)

end

function disableRTWBrowseButton(dialogH)

% Takes a cell array of widget Factory ID.
dialogH.disableWidgets({'Tag_ConfigSet_RTW_Browse'})

end
```

- In R2017b:

```
function sl_customization(cm)

% Disable for all Configuration Parameters dialog boxes
configset.dialog.Customizer.addCustomization(@disableRTWBrowseButton,cm);

end

function disableRTWBrowseButton(dialogH)

    % Takes a cell array of widget Factory ID.
    dialogH.disableWidgets({'STF_Browser'})

end
```

For more information on getting widget ID's and customizing the dialog box, see [Disable and Hide Dialog Box Controls](#).

Legacy Code Tool: Convert N-D matrix column-major format to row-major format

When you use the `legacy_code` function, you can now specify whether to convert an N-D column-major matrix to a row-major matrix. Use the **`convertNDArrayToRowMajor`** S-Function option. If true, the generated S-function specifies the automatic conversion of a matrix from column-major format to row-major format. This option replaces the **`convert2DMatrixToRowMajor`** option that handled only a 2-D column-major matrix. If you currently specify the **`convert2DMatrixToRowMajor`** option, the function automatically specifies the new **`convertNDArrayToRowMajor`** option.

Block Enhancements

Scoped Simulink Functions: Create Simulink Functions that can now cross model boundaries for reusable software components

The scope of a Simulink Function block is defined in its parent subsystem within the context of a model. If you place a function in a model at the root level, the function is scoped to the model by default. If you place a function in any Subsystem block, access to the function from outside the model is prohibited by default. In both cases, the Trigger block parameter **Function visibility** is set to **scoped**. For more information, see

- [Scoped and Global Simulink Function Blocks](#)
- [Scoping Simulink Functions in Subsystems](#)
- [Scope Simulink Functions in Models](#)

Tunability of n-D Lookup Table and Prelookup blocks

The n-D Lookup Table and Prelookup blocks now support tunability of the **First point** and **Spacing** parameters when **Breakpoints specification** is set to **Even spacing** for simulation and generated code.

Import Functional Mockup Unit (FMU) model for simulation

Use the FMU block to import FMUs into Simulink for simulation.

Signal Builder block updates

The Signal Builder block now exports groups formatted as `Simulink.SimulationData.Dataset`. Use the Signal Builder block **File > Export Data > To MAT-File** option to export data in this format. The **File > Export Data > To Workspace** option continues to behave as in previous releases. For more information, see [Signal Groups](#).

The support for `Simulink.SimulationData.Dataset` enables you to:

- Exchange groups between Signal Builder blocks.
- Migrate data for harness-free modeling with the Root Inport Mapper.
- Use Signal Builder exported data as inputs for the `parsim` function.
- Edit signal groups in MATLAB and reimport them to the block.

These `signalbuilder` functions were modified to support the new option:

Function	Syntax	Description
<code>get</code>	<code>ds=signalbuilder(block, 'get', group)</code>	Output one data set for one requested group.
<code>get</code>	<code>[ds1 ds2] =signalbuilder(block, 'get', group)</code>	Output <i>N</i> data sets for <i>N</i> requested groups.

Function	Syntax	Description
set	<code>signalbuilder(block, 'set', groupid, ds)</code>	Set one data set for the requested groups. Specifying an empty data set deletes the groups specified in <i>groupid</i> .
set	<code>signalbuilder(block, 'set', groupid, [ds1 ds2])</code>	Set <i>N</i> data sets for <i>N</i> requested groups.
append, appendgroup	<code>signalbuilder(block, 'append', ds)</code> <code>signalbuilder(block, 'appendgroup', ds)</code>	Append one data set.
append, appendgroup	<code>signalbuilder(block, 'append', [ds ds2])</code> <code>signalbuilder(block, 'appendgroup', [ds ds2])</code>	Append <i>N</i> data sets.

Limitations for the set, append, and appendgroup functions:

- Elements must be MATLAB timeseries data.
- Timeseries data and/or time must not be empty.
- Timeseries data must be of type double.
- Timeseries data must be 1-D (scalar value at each time).

Assignment and Selector blocks support enumerated data type

The Assignment and Selector blocks now support enumerated data type indexing.

Display, create, edit, and switch scenarios with Signal Editor block

To display, create, edit, and switch scenarios, use the Signal Editor block. This block lets you manipulate signals and scenarios, and lets you start the Signal Editor interface to create and edit signals.

Faster code generation and rapid accelerator mode startup for MATLAB System block

MATLAB System is now optimized for faster code generation. In addition, for rapid accelerator and code generation modes, MATLAB System block no longer generates code for simulation. Block is treated as if the **Simulate using** parameter is set to Interpreted execution.

Specify discrete single-rate sample time for MATLAB System block

You can specify discrete sample time for a MATLAB System block by implementing the `getSampleTimeImpl` method in your System object. This method is part of the `matlab.system.mixin.SampleTime` class. To specify or query single-rate sample time, use these methods:

- `createSampleTime` - Create a sample time specification object.
- `getCurrentTime` - Query a MATLAB System block for the current simulation time.
- `getSampleTimeImpl` - Specify the sample time for a System object.
- `getSampleTime` - Query the sample time for a System object.

S-Function Builder: New Start and Terminate methods and option to add PWorks

S-Function Builder has new capabilities, and changes include:

- Additional methods under **Build Info** have been integrated into **Start** and **Terminate** pane on the S-Function Builder dialog box. See Start Pane and Terminate Pane for more information.
- You can write your own code in the `mdlStart` method for one-time initialization, and runtime resource allocation, and the `mdlTerminate` method for freeing of memory or runtime resources.
- You can access pointers to file or memory using **PWorks** that can be referenced from any block methods within S-Function Builder. The number of **PWorks** can be configured using the **Number of PWorks** parameter on the **Initialization** pane. See Number of PWorks for more information.

From Spreadsheet block updates

The From Spreadsheet block now lets you specify if the first column of the spreadsheet contains time or data. Use the new parameter, **Treat first column as**, to select:

- **Time** — Treat first column as time.
- **Data** — Treat first column as data.

Step back support in the Floating Scope block

From the Floating Scope window, you can now step back the simulation. In the Floating Scope window, use the simulation controls in the toolbar to start, stop, step forward, and step backward.

Improvements to interactive legend in scope blocks

For scope blocks and System objects, use the scope legend to toggle signal visibility. In the scope legend, click a signal name to hide the signal in the scope. To show the signal, click the signal name again. To show only one signal, right-click the signal name, which hides all other signals.

New model advisor check and command-line API to analyze S-functions for possible problems and improvements

You can now analyze the quality of S-functions by using command-line APIs. The advisor check looks for potential problems and improvement opportunities in the S-function source C file and MEX file and parameter robustness.

- `Simulink.sfunction.Analyzer` — Create an S-function analyzer object.
- `Simulink.sfunction.analyzer.BuildInfo` — Create an object to represent build information such as additional header and source files.

- `Simulink.sfunction.analyzer.Options` — Specify categories of checks for S-function analyzer.
- `Simulink.sfunction.analyzer.findSfunctions` — Find all eligible S-functions in a model.

For more information, see [Check S-Functions Using S-Function Analyzer APIs](#) for more information.

MinMax block now supports boolean

The MinMax block now accepts and outputs boolean data types.

Math Function block `log` and `log10` changes

The Math Function block `log` and `log10` functions now ignore an imaginary part value of `-0` for mathematical expressions. This change enables these functions to behave more consistently with their MATLAB equivalents.

Version History

In previous releases, the Math Function block `log` and `log10` returned a negative imaginary part for some inputs with a `-0` imaginary part. Starting in R2017b, the Math Function block `log` and `log10` functions now return positive imaginary parts for inputs with `-0` imaginary parts.

For example, a Math Function block `log` function with an input of:

```
complex(-1,[0 -0]')
```

now returns:

```
0 + 3.142i  
0 + 3.142i
```

Before R2017b, the Math Function block `log` returned:

```
0 + 3.142i  
0 - 3.142i
```

Sum block changes

The Sum block now prevents overflows when:

- The Sum block **Output data type parameter** is set to `Inherit: Inherit via internal rule`.
- The Sum block inputs are built-in integers of the same type.
- The model is configured with **Configuration Parameters > > Hardware Implementation > Device vendor** set to `ASIC/FPGA`.

Version History

In previous releases, the Sum block could overflow with these conditions. Starting in R2017b, the Sum block outputs a data type with an appropriate data type to prevent overflows.

For example, a Sum block with these conditions and two `uint8` input data types now outputs a data type of `uint9`. Before R2017b, a Sum block with these conditions and two `uint8` input data types outputted a data type of `uint8`, which could cause overflows.

To maintain the behavior before R2017b, explicitly set the Sum **Output data type parameter** to the desired data type. For example, in this example, set it to `uint8`.

MATLAB Function block change

MATLAB Function blocks now have badges (downward facing arrows) in the lower-left corner when you create masks for these blocks. You can use these badges to look under block masks.

Algebraic constraint block updates

You can now specify additional parameters for the Algebraic Constraint block:

- **Constraint Type:** You can now choose to solve for $f(z) = z$ or $f(z) = 0$
- **Algorithm Type:** You can now explicitly choose between `Line Search` and `Trust Region` for the algebraic loop solver. The default is `auto` which selects the algebraic solver based on the model configuration.
- **Tolerance:** This is visible only when you explicitly select the algorithm to be used. Specify a smaller value for higher accuracy and a larger value for faster execution. The default tolerance is `auto`.

You can access and modify these parameters for each block from the Block Parameters dialog in Simulink. For more information, see Algebraic Constraint

Connection to Hardware

Simulink Support Package for Parrot Minidrones: Deploy flight control algorithms on PARROT minidrones

You can use the Simulink Support Package for Parrot Minidrones to build and deploy flight control algorithms on PARROT® minidrones. The support package lets you deploy algorithms wirelessly over Bluetooth. The algorithms can access onboard sensors—such as the ultrasonic, 6-DOF, and air pressure sensors—as well as the downward facing camera.

Simulink add-on tools provide additional capabilities. Aerospace Blockset includes the Quadcopter Project example, which makes use of PARROT minidrones. The project lets you model 6-DOF equations of motion and simulate aircraft behavior under various flight and environmental conditions. Simulink Coder lets you record flight data on the minidrone and access to the C code generated from Simulink models.

Support for Arduino MKR1000 Hardware: Run Simulink models on Arduino MKR1000 boards

You can use the Simulink Support Package for Arduino Hardware to generate code for the Arduino MKR1000 board. The support package also supports the wireless capabilities provided by the board. Using the WiFi blocks from the support package library, you can send and receive TCP/IP and UDP messages on MKR1000 board.

Blocks added to Android support package

This table lists the new blocks available in the Simulink Support Package for Android Devices.

Block	Usage
BLE Send	Send data to a connected Bluetooth device using the Bluetooth Low Energy (BLE) protocol.
BLE Receive	Receive data from a connected Bluetooth device using the Bluetooth Low Energy (BLE) protocol.

Support for DSP System Toolbox Array Plot block on Apple iOS and Android apps

The following support packages now support the DSP System Toolbox™ Array Plot block:

- Simulink Support Package for Android Devices
- Simulink Support Package for Apple iOS Devices

When you add an Array Plot block to a model created using one of these support packages, an equivalent real-time array plot display is added to the mobile app user interface.

MATLAB Function Blocks

FFT Library Calls for FFT Functions: Speed up simulation of FFT functions in a MATLAB Function block

In previous releases, for simulation of a MATLAB Function block that contained Fast Fourier Transform (FFT) functions (`fft`, `fft2`, `fftn`, `ifft`, `ifft2`, and `ifftn`), the simulation software used its own FFT algorithms. In R2017b, to improve the simulation speed of FFT functions, the simulation software uses the library that MATLAB uses for FFT algorithms.

If you use Simulink Coder to generate C/C++ code, you can generate calls to a specific, installed FFTW library by providing an FFT library callback class. See [Speed Up Fast Fourier Transforms in Code Generated from a MATLAB Function Block \(Simulink Coder\)](#).

For more information about the FFTW library, see www.fftw.org.

In R2017b, for simulation of a model, you can use the MATLAB `fftw` function in a MATLAB Function block. For code generation, to specify the planning method, implement a `getPlanMethod` method in an FFT library callback class. See [Speed Up Fast Fourier Transforms in Code Generated from a MATLAB Function Block \(Simulink Coder\)](#).

Strings: Represent text as a string scalar in a MATLAB Function block

In previous releases, in MATLAB code in a MATLAB Function block, you represented text as a character vector. For example:

```
c = 'Hello World';
```

In R2017b, you can represent text as a string scalar (a 1-by-1 MATLAB string array). For example:

```
s = "Hello World";
```

The MATLAB Function block does not support string arrays that have more than one element.

See [Code Generation for Strings](#).

Cell Arrays and Classes in Structures: Use structures that contain cell arrays and classes in a MATLAB Function block

In previous releases, in a MATLAB Function block, you could not assign a cell array or object to a structure field. In R2017b, in a MATLAB Function block, structures can contain cell arrays and classes. For example:

```
function result = assignToStruct(in1)
x = MyClass;
x.prop = in1;
y.val = x;           % object in struct
y.val2 = {1,2,3};   % cell in struct
result = y.val.prop;
end
```

Class Folders for Classes in a MATLAB Function block: Use MATLAB classes defined by using multiple files

In a MATLAB Function block, you can use a class that is defined in a class folder. When you define a class in a class folder, you can put the class definition in one file and the methods in other, separate files. The class folder name consists of the @ character followed by the class name. For example, the class folder @MyClass contains the class definition file MyClass.m. The folder can also contain separate files for the methods. For more information about class folders, see Folders Containing Class Definitions (MATLAB).

Property Validation: Use classes that restrict property values in a MATLAB Function block

A MATLAB Function block can use classes that restrict property values according to size, class, and other criteria. To establish criteria that a property value must conform to, use MATLAB validation functions or write your own validation functions. For information about property validation, see Validate Property Values.

A property validation error ends a simulation with an error message. To test property validation, it is a best practice to run a simulation over the full range of input values. C/C++ code generated by Simulink Coder does not detect or report property validation errors.

Value Classes in a MATLAB Function Block: Pass objects of value classes to and from extrinsic functions

In R2017b, in a MATLAB Function block, you can pass an object of a value class as an input to or output from an extrinsic function.

Statistics and Machine Learning Toolbox Code Generation: Generate C code for prediction by using discriminant analysis, k-nearest neighbor, SVM regression, regression tree ensemble, and Gaussian process regression models

You can generate code for these Statistics and Machine Learning Toolbox™ functions:

- `predict (CompactClassificationDiscriminant)` — Classify observations or estimate classification scores and costs by applying a discriminant analysis classification to new data.
- `predict (ClassificationKNN)` — Classify observations or estimate classification scores and costs by applying *k*-nearest neighbor classification, based on an exhaustive search, to new data.
- `predict (CompactRegressionSVM)` — Predict responses by applying a support vector machine (SVM) regression to new data.
- `predict (CompactRegressionEnsemble)` — Predict responses by applying ensembles of regression trees to new data.
- `predict (RegressionLinear)` — Predict responses by applying a linear regression to new data.
- `predict (CompactRegressionGP)` — Predict responses or estimate confidence intervals on predictions by applying a Gaussian process regression to new data.
- `knnsearch (ExhaustiveSearcher)` and `knnsearch`— Identify the *k*-nearest neighbors using the exhaustive search algorithm.

- `rangesearch` (`ExhaustiveSearcher`) and `rangesearch` — Identify all neighbors within a specified distance using the exhaustive search algorithm.
- `pdist2` — Compute the pairwise distance between two sets of observations.

When you train an SVM model by using `fitcsvm` for code generation, you can now specify a score transformation function by using the `'ScoreTransform'` name-value pair argument or by assigning the `ScoreTransform` object property. Therefore, `saveCompactModel` can accept compact SVM models equipped to estimate class posterior probabilities, that is, models returned by `fitposterior` or `fitSVMPosterior`. Also, you can now implement one-class learning.

When you train a linear classification model by using `fitclinear` for code generation, you can now specify either `'svm'` or `'logistic'` for the `'Learner'` name-value pair argument.

Code generation for more MATLAB functions

Characters and Strings

- `cellstr`
- `contains`
- `count`
- `endsWith`
- `erase`
- `extractAfter`
- `extractBefore`
- `insertAfter`
- `insertBefore`
- `isstring`
- `replace`
- `replaceBetween`
- `reverse`
- `startsWith`
- `string`
- `strip`
- `strlen`

Data Type Conversion

- `int2str`

Data Types

- `enumeration`

Fourier Analysis and Filtering

- `fftw`

Moving Statistics

- movmad
- movmax
- movmean
- movmedian
- movmin
- movprod
- movstd
- movsum
- movvar

Preprocessing Data

- isoutlier
- filloutliers

Programming Utilities

- builtin

Property Validation Functions

- mustBeFinite
- mustBeGreaterThan
- mustBeGreaterThanOrEqual
- mustBeInteger
- mustBeLessThan
- mustBeLessThanOrEqual
- mustBeMember
- mustBeNegative
- mustBeNonempty
- mustBeNonNan
- mustBeNonnegative
- mustBeNonpositive
- mustBeNonsparse
- mustBeNonzero
- mustBeNumeric
- mustBeNumericOrLogical
- mustBePositive
- mustBeReal

Code generation for more Audio Toolbox System objects

- `graphicEQ`

Code generation for more DSP System Toolbox System objects

- `dsp.BlockLMSFilter`
- `dsp.FrequencyDomainFIRFilter`
- `dsp.ZoomFFT`

Code generation for more Phased Array System Toolbox System objects and functions

- `phased.HeterogeneousConformalArray`
- `phased.HeterogeneousULA`
- `phased.HeterogeneousURA`
- `phased.UnderwaterRadiatedNoise`
- `range2tl`
- `sonareqsl`
- `sonareqsnr`
- `sonareqtl`
- `tl2range`

Code generation for more Robotics System Toolbox functions

- `lidarScan`
- `matchScans`

S-Functions

New continuous time C S-Function examples

These new C S-function examples demonstrate continuous time applications:

- `sfcndemo_sfun_zc_cstate_sat` shows an example implementing a continuous integrator with saturation limits and zero-crossing detection.
- `sfcndemo_angle_events` shows a method for robust and efficient detection of a rotating body crossing specified angles.
- `sf_angle_events` demonstrates angle detection and incorporates Stateflow to schedule function calls.
- `sfcndemo_sfun_localsolver` demonstrates a continuous integrator where the continuous states are solved by a separate local solver instead of the model's solver

Variable Discrete Sample Time

You can now use a variable discrete sample time in C S-Function blocks. For more information, see [Variable Discrete Sample Time](#). To understand how to use it in your own model, open `sfcndemo_pwm`. You can register a C S-function block for the sample time using `ssSetVariableDiscreteSampleTime` and specify its executions using `ssSetNumTicksToNextHitForVariableDiscrete`.

R2017a

Version: 8.9

New Features

Bug Fixes

Version History

Simulation Analysis and Performance

Parallel Simulations: Directly run multiple parallel simulations from the `parsim` command

Using this feature, you can iteratively change various parameter values in your model, starting from a baseline setting, and perform a series of simulations with these values. You can provide these changes to your model through a `SimulationInput` object and run multiple simulations with them. Some common use cases for this approach include model testing, design of experiments, Monte Carlo runs, and sensitivity and robustness analysis.

In addition, with a Parallel Computing Toolbox™ license, you can use the `parsim` command to run multiple simulations in parallel. With the `parsim` command, the amount of custom code you need to run multiple simulations in parallel is significantly reduced compared to using the `sim` command within a `parfor` loop.

When you use a `SimulationInput` object to run multiple simulations, the simulations use the values in the `SimulationInput` object rather than the values defined in your model. This enables you to run multiple simulations without needing to modify your model between each simulation. The `SimulationInput` object allows you to change the following settings in your model:

- Initial State
- External Inputs
- Model Parameters
- Block Parameters
- Variables

For more information, see [Run Multiple Parallel Simulations](#).

Simulink Cache: Get simulation results faster by using shared model artifacts

In R2017a, performing an update diagram or running a simulation on a new model that builds a model reference SIM target or rapid accelerator target creates a Simulink cache file (an `.slxc` file). Use Simulink cache files to share referenced model build artifacts without repeating the cost of a first-time build. Some examples of situations when the cache files can speed up simulation include:

- First-time builds for later use of a referenced model by yourself or others
- Parallel simulations

For details, see [Reuse Simulation Builds for Faster Simulations](#).

Inport File Streaming: Stream large input signals from MAT-files without loading the data into memory

To log, load, and analyze large amounts of simulation data that cannot be stored in memory, you can use Simulink data streaming and MATLAB big data features. R2017a includes features that help you to:

- Use big data in Simulink logging and loading.
- Use in-memory `timetable` format data within `Simulink.SimulationData.Dataset` objects, in combination with other kinds of data, such as `timeseries`.
- Use MATLAB big data analysis features.

For details, see *Working with Big Data for Simulations*.

Log and Load Big Data

You can use a `matlab.io.datastore.SimulationDatastore` object in a `Simulink.SimulationData.Dataset` object to stream data into a root Inport block. Create a `SimulationDatastore` object for an individual signal that is stored in an in-file `Dataset` object referenced by a `Simulink.SimulationData.DatasetRef` object.

To obtain a `SimulationDatastore` object from a `DatasetRef` object, you can use curly braces indexing syntax with a `DatasetRef` object. For example, `simDatastore = dsr{3}` returns the third `DatasetRef` element as a `SimulationDatastore` object.

To load logged signal data into root Inport blocks, you can use the new `createInputDataset` function to create a `Dataset` object that contains elements that correspond to root-level Inport blocks in the model.

Use timetable Data in Dataset Objects

You can use the new **DatasetSignalFormat** configuration parameter to specify whether you want data logged in `Dataset` format to be saved in `timeseries` (default) or in-memory `timetable` format within the `Dataset` object.

When you read data from a `SimulationDatastore` object, the data is returned as a `timetable` object.

Analyze in MATLAB Big Data Created by Simulink

You can use a `SimulationDatastore` object to output a MATLAB tall `timetable` object to analyze big data from a simulation.

Unified Streaming and Logging: Mark a signal once to stream it to the Simulation Data Inspector and log it to the MATLAB workspace

In R2017a, when you log signals, the data is logged to the workspace and to the Simulation Data Inspector. Simulink interprets streaming badges in models from prior releases as logging badges. You can use the **Data Import/Export > Signal logging** configuration parameter to control whether signal data for logged signals is exported to the MATLAB workspace. When you use the **Override signals** logging mode to disable logging for a signal, that signal no longer streams to Dashboard blocks or the Simulation Data Inspector. For information about how to configure your model and select signals for logging see “Configure a Signal for Logging” and “View Data in the Simulation Data Inspector”.

You can use the normal, accelerator, rapid accelerator, SIL, and PIL simulation modes when you log signals to the Simulation Data Inspector.

You can use signal logging with these products:

- Stateflow — Elements in a chart that can be logged are not streamed, but are logged to the workspace.
- Simulink Real-Time

Simulation Data Inspector: Run simulation comparisons with a new UI, time tolerance support, and faster performance

The Simulation Data Inspector has a new, streamlined interface. Examples of the changes include:

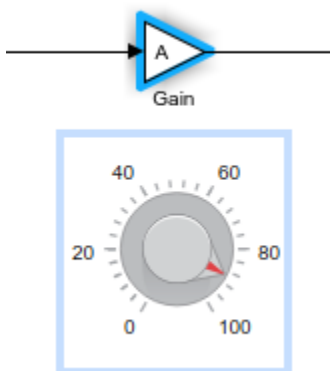
- Inspection and formatting options are integrated into the **Inspect** and **Compare** views, instead of being on a toolbar.
- You can specify display options for axes lines and plots, such as color.
- You can format, group, and sort the signal data.
- The **Compare** view displays status information about how many signals are within the tolerance range.

When you compare simulation runs and signals, you can now set time tolerances. The plot for each signal displays a band representing the tolerance range.

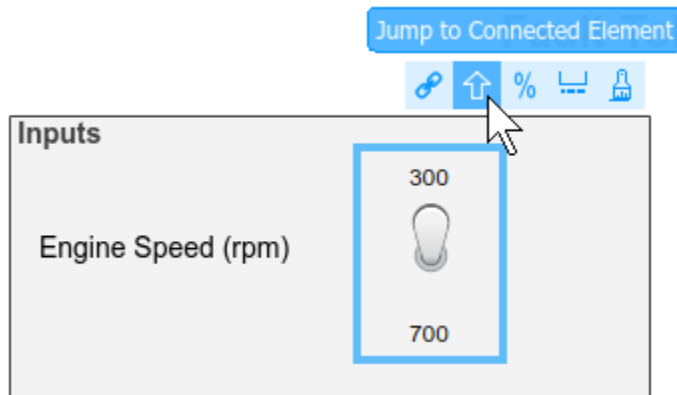
In R2017a, to save Simulation Data Inspector session data and setup is faster than in previous releases, you can specify the `.mldatx` file extension instead of the `.mat` extension. A status overlay displays information about the save operation. For `.mldatx` files, you can specify preferences to compress the file and to limit the amount of data stored. When you open a `.mldatx` file with MATLAB closed, MATLAB and the Simulation Data Inspector open.

Dashboard Block Connection Indicators: Easily determine which block in your model is associated with a given Dashboard block

Dashboard block connection indicators provide visual indications of the connections between Dashboard blocks and signals and components in your model. You can click a Dashboard block to highlight the connected block or signal and vice versa.



Dashboard block binding indicators also allow you to jump to the connected element through subsystem hierarchies to locate the connection in the model.



Signal Tracing: Incrementally trace and highlight paths for debugging

Use the **Highlight Signal to Destination** option from the context menu for the signal to highlight the destination for a signal. For example, selecting this option for a bus highlights only the signals actually selected from the bus. For more information, see [Display Signal Sources and Destinations](#).

Root-Level Inport Blocks: Create dataset for root-level Inport blocks

To generate a `Simulink.SimulationData.Dataset` object from the root-level Inport blocks in a model, you can use the `createInputDataset` function. Signals in the generated dataset have the properties of the Inport blocks and the corresponding ground values at model start and stop times. You can create `timeseries` objects for the time and values for signals for which you want to load data for simulation. The other signals use ground values.

Simulation Logging Data and Metadata: Access simulation data and information more directly

You can use these new capabilities to access simulation outputs in `SimulationOutput` objects and data in `Dataset` format more easily.

- Access logged simulation data (for example, `tout`, `xout`, and `yout`) as properties of the `out` element of a `Simulink.SimulationOutput` object. For example, you can access output data that uses the default variable named `yout`:

```
simOut = sim('vdp', 'SaveOutput', 'on')
myOutput = simOut.simOut.yout
```

- Use the Variable Editor to access logged data.
 - Each element in a `Simulink.SimulationData.Dataset` is a separate row.
 - Display `Dataset` objects to view information about elements, such as the block path and type of data the element contains.
 - Use the `openvar` function to view a logged data variable in the Variable Editor.
 - Use curly braces to streamline indexing syntax to access elements in a `Dataset` object, instead of using `get`, `getElement`, `setElement`, or `addElement` methods. For example, to return the second element of a `Dataset`:

```
xout{2}
```

You can use these new capabilities to access simulation logging error messages and metadata on a `SimulationOutput` object.

- View error message and information about the stack and causes for simulation data by using the `SimulationOutput` object `ErrorMessage` property.
- Display the error message in the Variable Editor, by double-clicking the **ErrorMessage** row.
- Access an exception by using:

```
Simulink.SimulationMetadata.ExecutionInfo.ErrorDiagnostics.Diagnostic()
```
- View simulation metadata by using the `SimulationMetadata` property of the `SimulationOutput` object.
- Use tab completion to access `SimulationMetadata` object properties such as `ModelInfo` and to access field names.
- Display simulation metadata in the Variable Editor using one of these approaches:
 - Select the **Show Simulation Metadata** check box (which displays the data in a tree structure).
 - Double-click the **SimulationMetadata** row.
 - View the `SimulationOutput` object.

For parallel simulations, for which you specify an array of input objects, if you are logging to file, `Simulink`

- Creates `Simulink.SimulationData.DatasetRef` objects to access output data in the MAT-file and includes those objects in the `SimulationOutput` object data
- Enables the `CaptureErrors` argument for simulation.

Rapid Accelerator mode: Rapid Accelerator now supports S-functions without source code

Rapid Accelerator mode can now use C S-functions without storing the S-function source code. You do not need to collocate the S-function source code with the S-function binaries. This option enables easier sharing of models.

Signal Editor: Create and edit input signals that can be organized for multiple simulations

Use the Signal Editor to create and edit input signals that you can organize for multiple simulations. For more information, see [Create and Edit Signal Data](#).

Improved simulation performance when stepping back is enabled

The performance of stepping back using the Simulation Stepper has been improved. For more information on the Simulation Stepper, see [Simulation Stepping](#).

Simulink Diagnostic Management: Control which simulation and fixed-point diagnostic warnings you receive from specific blocks, including model reference

Select blocks with certain diagnostic suppressions by default

Beginning in R2017a, the Counter Free-Running, HDL Counter, Counter Limited, and Extract Bits blocks no longer report wrap on overflow warnings. The blocks continue to report errors due to wrap on overflows. You can restore the warning diagnostic by breaking the library link and using the `Simulink.restoreDiagnostic` function.

Diagnostic suppressor functions support `MSLDiagnostic` as input argument

You can now suppress and restore certain diagnostic warnings thrown by your model using a `Simulink.MSLDiagnostic` object as an input to the `Simulink.suppressDiagnostic` and `Simulink.restoreDiagnostic` functions.

To use simulation metadata and `MSLDiagnostic` objects, use `set_param` to set `ReturnWorkspaceOutputs` to `on`. Store the simulation output in a variable.

```
set_param(model_name, 'ReturnWorkspaceOutputs', 'on');
out = sim(model_name);
```

Access the `MSLDiagnostic` object through the simulation output.

```
diag = out.getSimulationMetadata.ExecutionInfo.WarningDiagnostics(1).Diagnostic
diag =
```

MSLDiagnostic with properties:

```
  identifier: 'SimulinkFixedPoint:util:fxpParameterPrecisionLoss'
  message: 'Parameter precision loss occurred for 'Value' of
  'Suppressor_CLI_Demo/one'. The parameter's value cannot be
  represented exactly using the run-time data type. A small
  quantization error has occurred. To disable this warning or error,
  in the Configuration Parameters > Diagnostics > Data Validity pane,
  set the 'Detect precision loss' option in the Parameters group to 'none'.'
```

Use the `Simulink.suppressDiagnostic` function to suppress the diagnostic warning specified by the `MSLDiagnostic` object, `diag`.

```
Simulink.suppressDiagnostic(diag)
```

You can restore the diagnostic using the `Simulink.restoreDiagnostic` function

```
Simulink.restoreDiagnostic(diag)
```

Improved workflow for suppressing diagnostics from referenced models

You can now suppress certain diagnostic warnings for specified instances of warnings in a referenced model. By accessing the `MSLDiagnostic` object of the specific instance of the warning, you can suppress the warning only when the block inside the referenced model is simulated from the specified top model.

Absolute tolerance for continuous variable step solver tied to the relative tolerance

The behavior of the auto absolute tolerance for continuous variable step solvers has changed. Previously, setting the absolute tolerance to `auto` specified the initial value as $1e-6$, which was adjusted by the solver during simulation. Now, the absolute tolerance is tied to the relative tolerance. The value is initially set to the minimum of $1e-6$ and $1e-3$ times the relative tolerance and adjusted by the solver during simulation. This change enhances the overall solver robustness and improves simulation accuracy. For models with auto absolute tolerance developed before R2017a, simulation performance may be improved by increasing maximum step size and loosening relative tolerance.

Simulink Editor

Automatic Port Creation: Add inports and outports to blocks when routing signals

For some blocks in Simulink models, dragging a line to connect another block to it adds a port. For example, dragging a line from a block to a subsystem adds a port to the subsystem and the inport or output block inside the subsystem. For an example, see [Build and Edit a Model in the Simulink Editor](#).

These blocks add ports when you connect a signal to them:

- Subsystem blocks except masked blocks and the Configurable Subsystem block
- Stateflow charts, truth tables, and state transition tables
- Bus Creator, Bus Selector, Mux, Demux, and Merge blocks
- Vector Concatenate and Matrix Concatenate blocks
- Add, Sum, Subtract, Sum of Elements, Product, and Product of Elements blocks
- Scope blocks
- Logical Operator blocks
- MATLAB Function block

Model Block Masking: Customize the parameter dialog boxes for referenced models

You can now apply a mask on a Simulink model and create a customized parameter dialog box for the referenced model. For more information, see [Create and Reference a Masked Model](#).

Quick Find: Use a modifier to search for model properties in search box

In R2017a, you can search for property values in a model using a modifier in the search box. Enter the property and value you want to search for, in the form `Property:Value`. For more information on using modifiers in the **Find** box, see [Search for Model Elements Using Find](#).

Format Painter: Copy formatting between model elements

Using the ellipsis menu on a block, area, or signal line in a model, you can copy the formatting from one model element to another. The **Copy Formatting** brush appears on the menu when the model element you select has formatting applied. Examples of formatting include a font change or foreground or background color. For more information, see [Adjust Visual Presentation to Improve Model Readability](#).

Refresh Library Browser: Update quick insert list with custom libraries using menu command

In R2017a, you can update the contents of the quick insert list to use your current configuration of custom libraries by using the **Refresh Library Browser** command. Use the quick insert interface to

add blocks to your model without leaving the canvas. To use quick insert, click where you want to add a block and start typing the name of the block.

To ensure that the quick insert interface includes the blocks from your custom libraries currently in effect, use the Library Browser **Refresh Library Browser** command. In the Library Browser, right-click anywhere in the library list and select **Refresh Library Browser**.

Functionality Being Removed or Changed

Two functions in the `Simulink.BlockDiagram` class have been renamed.

Functionality

These functions were renamed:

- `Simulink.BlockDiagram.copyContentsToSubSystem`
- `Simulink.BlockDiagram.createSubSystem`

Result

The renamed functions still run.

Use Instead

The functions were renamed to:

- `Simulink.BlockDiagram.copyContentsToSubsystem`
- `Simulink.BlockDiagram.createSubsystem`

Compatibility Consideration

For each function, you can use the earlier or newer syntax.

Version History

Scripts that use the earlier syntax will continue to work. Use the new syntax going forward to match the spelling of other functions in this class.

Optimize rendering during mask icon drawing

When the mask drawing commands in the Mask Editor do not have dependency on the mask workspace, you can specify the value of the **Run initialization** option as **Off**.

Setting the value to **Off** helps to optimize Simulink performance during mask icon rendering because the mask initialization commands are not executed. For more information, see Rules for Initialization commands.

Component-Based Modeling

Reduced Bus Wiring: Quickly group signals as buses and automatically create bus element ports for fewer signal lines between and within subsystems

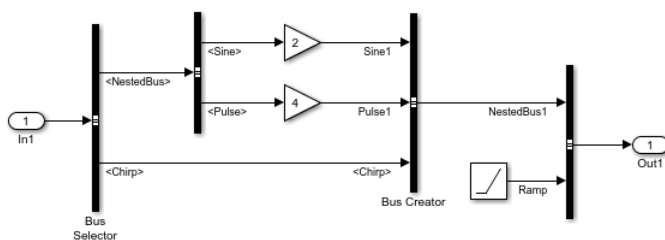
The Ports & Subsystems library contains new In Bus Element and Out Bus Element blocks. These bus element port blocks provide a simplified and flexible way to use bus signals as inputs and outputs to subsystems.

The In Bus Element block is equivalent to an Inport block combined with a Bus Selector block. The Out Bus Element block is equivalent to an Outport block combined with a Bus Creator block. These new blocks are of block type Inport and Outport, respectively. There are no specifications allowed on bus element port blocks, which support inherited workflows. You cannot use the Block Parameters dialog box of a bus element port block to specify bus element attributes, such as data type or dimensions.

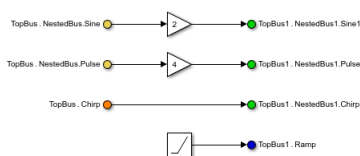
To work with buses at subsystem interfaces, consider using In Bus Element and Out Bus Element blocks. This bus element port block combination:

- Reduces signal line complexity and clutter in a block diagram.
- Makes it easy to change the interface incrementally.
- Allows access to a bus element closer to the point of usage.
 - For input, avoid a duplicate Inport blocks and a Bus Selector, Goto, and From block configuration.
 - For output, avoid a Goto, From, and Bus Creator block configuration.

This model uses Inport, Bus Selector, Bus Creator, and Outport blocks.



Here is an equivalent model using bus element port blocks.



You can refactor a subsystem interface that has Inport, Bus Selector, Bus Creator, and Outport blocks to use In Bus Element and Out Bus Element blocks. Conversions are supported only when the signal lines or blocks do not have any extra specifications. You can use single-click operations to convert:

- Inport and Bus Selector blocks in a subsystem to In Bus Element blocks.
- Outport and Bus Creator blocks in a subsystem to Out Bus Element blocks.

To transform input or output interfaces of subsystems to use bus element port block, you can use marquee selection options.

For more information, see [Simplify Subsystem Bus Interfaces](#).

Bus and Vector Mixtures Not Supported

A mixtures of bus and vector (mux) signals occurs when some blocks treat a signal as a vector, while other blocks treat that same signal as a bus. Mixing bus and vector signals in a model causes your model to be less robust. Configuring your model to prevent bus and vector mixtures:

- Improves loop handling
- Produces clear error messages
- Contributes to consistent edit and compile-time behavior

To check a model for bus signals that are used as vectors, use the new Model Advisor **Check bus signals treated as vectors** check.

Version History

In R2017a, the Demux block parameter **Bus selection mode** (BusSelectionMode) is no longer supported.

In R2017a, these checks and diagnostics are no longer available:

- In the Model Advisor, the **Check bus usage** check
- In the Upgrade Advisor, the **Check Mux blocks that create bus signals** check
- The **Configuration Parameters > Diagnostics > Connectivity > Mux blocks used to create bus signals** parameter

For R2017a, to handle a legacy model that contains vector and bus mixtures:

- 1 In R2016b, run the Upgrade Advisor with the **Check Mux blocks that create bus signals** check enabled.
- 2 In R2017a, open the upgraded model.

You can continue to use existing `sreplace_mux` function to replace with Bus Creator blocks any Mux blocks are used to create buses. However, in a future release, the `sreplace_mux` function could be removed.

Inline Variants: Single-Input/Single-Output Inline Variant blocks support zero active variant control

A variant model with Single-Input/Single-Output Inline Variant block can now be simulated without an active variant choice. You can use the **Allow zero active variant controls** option to specify the choice.

For more information on Inline Variants, see [Define, Configure, and Activate Variants](#).

Searchable, sortable tables for parameterizing reusable models with model arguments

Before R2017a, you used comma-separated lists to identify model arguments and specify argument values for reusable referenced models.

In R2017a:

- To identify variables in a model workspace as model arguments, in the Model Explorer **Contents** pane, use the **Argument** check box instead of a comma-separated list.
- To specify argument values in a Model block, use a searchable, sortable table instead of a comma-separated list.

Now, when you set the programmatic parameter of the Model block, `ParameterArgumentValues`, you can use a structure instead of a comma-separated list. See [Parameterize Instances of a Reusable Referenced Model](#).

Version History

If you use the **Argument** check box in the Model Explorer, you must modify scripts that manipulate the `ParameterArgumentValues` parameter of referencing Model blocks. Make the scripts set `ParameterArgumentValues` by using structures instead of comma-separated lists. If you do not modify the scripts, they generate errors while trying to use comma-separated lists.

Project and File Management

Simulink Project Upgrade: Easily update all the models in your Simulink Project to the latest release

Easily upgrade all the models in your project using the Upgrade Project tool in Simulink Project, by selecting **Run Checks > Upgrade**. You can upgrade all models in your project to the latest release using a simple workflow. The tool can apply all fixes automatically when possible, upgrade all model hierarchies in the project at once, and produce a report. You do not need to open the Upgrade Advisor.

For details, see Upgrade All Project Models.

Missing Product Identification: Fix models with unresolved library links and unknown block types by finding and installing missing products

Simulink now tries to help you find and install missing products that a model needs to run. If you open a model that contains built-in blocks or library links from missing products, you see labels and links to help you fix the problem.

- Blocks are labeled with missing products (for example, **SimEvents not installed**)
- Tooltips include the name of the missing product
- Messages provide links to open Add-On Explorer and install the missing products
- Simulink Project dependency analysis reports missing products required by a project.

To find a link to open Add-On Explorer and install the product:

- For built-in blocks, open the Diagnostic Viewer, and click the link in the warning.
- For unresolved library links, double-click the block to view details and click the link.

The image shows a Simulink model with a block labeled "Aerospace Blockset" not installed. A tooltip for this block reads: "Cannot find library called 'aerolibconvert2'. 'Aerospace Blockset' is not installed. Double-click block for more information." Below the model is the Diagnostic Viewer window, which displays a warning: "Cannot find library called 'aerolibconvert2'. To use the library, install 'Aerospace Blockset'. [26 similar]". To the right is the Block Parameters dialog for "Angle Conversion", which shows the "Reference" field as "Unresolved library link. Specify valid library block path as the source block." and the "Details" field as "Cannot find library called 'aerolibconvert2'. To use the library, install 'Aerospace Blockset'." The "Source block" field is set to "aerolibconvert2/Angle Conversion" and the "Source type" is "Angle Conversion".

For details, see the Unresolved Link block reference page, and Check Dependency Results and Resolve Problems.

Git Pull: Fetch and merge in one step

You can now use Pull for Git from the Simulink Project toolstrip tab. Pull fetches the latest changes and merges them into your current branch. Previously, you had to fetch and merge separately before you could see changes.

For details, see Pull, Push, and Fetch Files with Git.

Project Creation API: Set up projects programmatically, including shortcuts and referenced projects

New functions enable you to create Simulink projects programmatically. You can set up referenced projects, shortcuts, and the path. Previously you had to use Simulink Project to set up projects interactively. New functions:

- `slproject.create`
- `addPath`
- `removePath`
- `addReference`
- `removeReference`
- `addShortcut`
- `removeShortcut`

For an example, see [Creating Simulink Projects Programmatically](#).

Referenced Project Change Management: Compare components with checkpoints

You can create a checkpoint for a referenced project. You can then compare the referenced project against the checkpoint to detect any changes.

For details, see [Manage Referenced Project Changes Using Checkpoints](#).

Source Control Toolstrip: Simplified workflow for working with source control

You can now use the Simulink Project toolstrip tab to perform source control action. Built-in SVN or Git and other source control integrations now have toolstrip buttons for operations such as Commit, Update, Push, Pull, and other actions.

For details, see [Source Control in Simulink Project](#).

Custom Task Tool: Improved interface for managing custom tasks and creating reports from results

You can run custom tasks on files in Simulink Project with a simplified workflow. The new custom task tool makes it easier to select files, run tasks, and create and save reports. Previously you had to select the Batch Job node in the project tree to run custom tasks.

For details, see [Create a Custom Task Function](#).

Git Remote Repositories: Connect existing project to a remote repository

In an existing Simulink project under local Git source control, you can now specify a remote repository for the project by clicking Remote on the Simulink Project tab. Previously you could only specify a remote Git repository when you created the project.

For details, see [Add a Project to Git Source Control](#).

Start Page Example Search: Find featured examples

You can now search for examples in the Simulink start page examples tab. Enter search terms to find examples titles and descriptions of interest, or open further examples on the web.

For details, see [Create a Model](#).

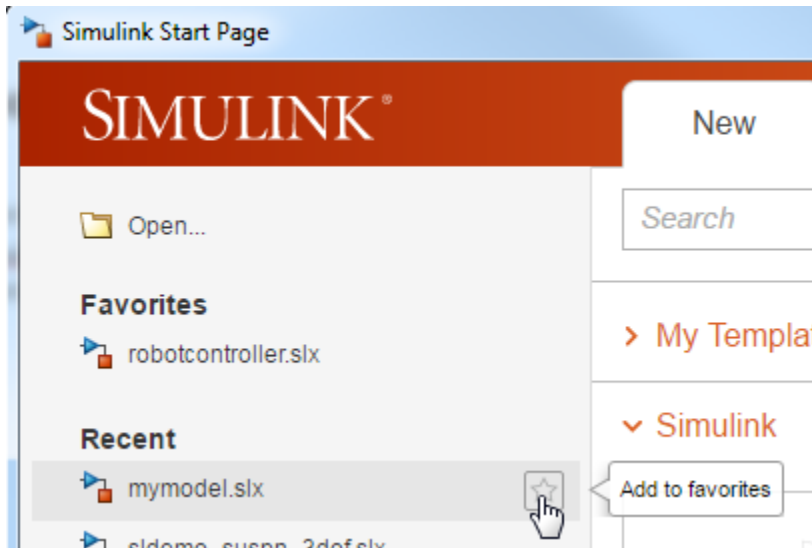
Model Templates: Simplified workflow for exporting models to templates

Exporting models to templates is simplified and you can add thumbnail images for your templates. The dialog box now prepopulates fields with model information that you can edit if needed.

For details, see [Create a Template from a Model](#).

Start Page Favorites: Easily get back to your favorite models and projects

In the Simulink start page recent files list, you can add files to favorites. The Favorites list then appears above recent files in the start page, so that you can easily reopen your favorite models and projects.



For details, see [Open a Model](#).

Project Componentization: Include referenced projects in templates for sharing components

When you make a template from a project, you can now include referenced projects so that template users can get all the components. You can choose whether to share references together with main project.

For details, see [Create a Template from the Current Project](#).

bdIsDirty Function: Programmatically check whether models contain unsaved changes

You can use the new `bdIsDirty` function to check whether or not loaded block diagrams contain unsaved changes.

For details, see [bdIsDirty](#).

listRequiredFiles Function: Get project file dependencies programmatically

You can use the new `listRequiredFiles` function to get the downstream dependencies of a file in a Simulink Project. The function returns the files that the specified file requires to run.

For details, see [listRequiredFiles](#).

Data Management

Simulation Data: Easily access simulation output data in the MATLAB Variable Editor and MATLAB Command Window

You can use the Variable Editor to inspect simulation data that is stored as `Simulink.SimulationData.Dataset` or `Simulink.SimulationOutput` objects. Also, for `SimulationOutput` objects, you can view simulation metadata that provides context about the simulation, such as information about the model. For details about the Variable Editor, see [Editing Workspace Variables](#).

In the MATLAB Command Window

, you can use dot notation to access an element of a `SimulationOutput` object. For example, to return the `xout` element of a `SimulationOutput` object called `simout`, you can enter:


```
xout = simout.yout;
```

Before R2017a, you had to use a `get` function (which continues to work in R2017a):

```
xout = simout.get('yout');
```

Management of workspace variables and mask parameters from block parameters

Before R2017a, you could create or navigate to a workspace variable by right-clicking the value of a block parameter, for example, the **Gain** parameter of a Gain block.

In R2017a, right-clicking a parameter value does not initiate creation or navigation. Instead, you left-click the new button  next to the value of the block parameter. The button appears in block dialog boxes, the Property Inspector, the Model Data Editor, and the Model Explorer. For more information, see [Manage Variables from Block Parameters](#).

- In the Model Data Editor, on the **Parameters** tab, the **Defined In** column no longer appears. To navigate to a workspace variable from a block parameter that appears on the **Parameters** tab, use the new button.
- In the Model Explorer, when you edit the value of a workspace variable or parameter object, the **Open Variable Editor** button no longer appears. To open the Variable Editor, use the new button.

Association of root-level Outport block with Simulink.Signal object

Before R2017a, you could not associate a root-level Outport block with a `Simulink.Signal` object.

In R2017a, you can use the Model Data Editor (see [Configure Data Properties by Using a Table](#)) to make this association.

Initial State: Log and load initial states using Dataset format

If you set the `SaveFormat` model parameter to `'Dataset'`, the `Simulink.BlockDiagram.getInitialState` function returns the initial state as a `Simulink.SimulationData.Dataset` object.

To load an initial state, you can specify a `Dataset` object for the **Data Import/Export > Initial state** configuration parameter.

Root Inport Mapper Tool Updates

The tool has the following updates:

- Root Inport Mapping has been renamed to Root Inport Mapper.
- To use strong data typing when mapping data from spreadsheets, select **Options > Use Strong Data Typing with Spreadsheets**. For more information, see Set Options for Mapping.

Legacy Code Tool `StartFcnSpec` and `InitializeConditionsFcnSpec` accept outputs as arguments

Previously, when you specified the `StartFcnSpec` and `InitializeConditionsFcnSpec` functions of the Legacy Code Tool data structure, the functions could not accept outputs as arguments. In R2017a, these functions can access output ports if the S-Function option `outputsConditionallyWritten` is set to true. With this option setting, the generated S-Function specifies that the memory associated with each output port cannot be overwritten and is global (`SS_NOT_REUSABLE_AND_GLOBAL`).

Utility to generate Simulink representations of custom data types defined by external C code

Before R2017a, to integrate your external C code into Simulink by using an S-function, you needed to manually create:

- `Simulink.Bus` objects to represent the structure types (`struct`) that your code defined. For example, you could use the Bus Editor.
- Enumeration definitions to represent the enumerated types (`enum`) that your code defined. For example, you could use the function `Simulink.defineIntEnumType`.
- `Simulink.AliasType` objects to represent primitive `typedef` statements in your code.

These manual techniques led to data entry errors. Also, maintaining the objects and definitions in Simulink was difficult.

In R2017a, you can use a function, `Simulink.importExternalCTypes`, that parses your external header files for `struct`, `enum`, and `typedef` type definitions and generates corresponding objects and enumeration classes.

Direct representation of fixed-point data types by `Simulink.AliasType`

A fixed-point data type consists of a word length and a scaling (for example, a 16-bit word length with a 14-bit binary fraction length).

Before R2017a, to specify a custom name for a fixed-point data type, you used one of these techniques that involve `Simulink.NumericType` objects:

- Use both `Simulink.NumericType` and `Simulink.AliasType` objects.

- 1 Create a `Simulink.NumericType` object in the base workspace or a data dictionary.
 - 2 Configure the `Simulink.NumericType` object to represent the fixed-point data type.
 - 3 Create a `Simulink.AliasType` object.
 - 4 Configure the `Simulink.AliasType` object to use the `Simulink.NumericType` object as a base data type.
- Use a standalone `Simulink.NumericType` object.
 - 1 Create a `Simulink.NumericType` object in the base workspace or a data dictionary.
 - 2 Configure the `Simulink.NumericType` object to represent the fixed-point data type.
 - 3 Set the `IsAlias` property of the object to `true`.

In R2017a, a `Simulink.AliasType` object can directly represent a fixed-point data type. You can use `Simulink.AliasType` to rename numeric data types including integer, floating-point, and fixed-point types. You do not need to create a `Simulink.NumericType` object.

Display of alias, base, or both data types in a model

Before R2017a, when you displayed signal data types on a block diagram by selecting **Display > Signals and Ports > Port Data Types**, you were not able to control the display of data type aliases. For example, if you used a `Simulink.AliasType` object to set the data type of a signal, the diagram displayed only the alias. The diagram did not display the underlying base data type (such as `int16`).

In R2017a, you can choose whether to display data type aliases, base data types, or both for all of the signals in a model. In the model, choose an option for **Display > Signals and Ports > Port Data Type Display Format**. You can temporarily or permanently adjust this setting, which is saved with the model file, to:

- Control the appearance of the model to users who do not need to know the underlying numeric data types.
- Inspect the numeric data types that signals use on a high level, especially while designing and debugging fixed-point models.

More accurate comparison of nondouble data to specified minimum and maximum values

You can specify minimum and maximum values for signals and block parameters in a model. For example, in a Constant block, you can set **Output minimum** and **Output maximum** to specify design limits for the block output signal. You must set such a design limit by using a literal number (with implicit data type `double`) or an expression that yields a `double` number.

Before R2017a, if the data item (signal or block parameter) used a data type other than `double`, Simulink:

- 1 Cast the nondouble value of the data item to `double`.
- 2 Compared the `double` value of the data item to the `double` value of the design limit.

If the storage and typecasts of the design limit or the data item incurred losses of accuracy (quantization) or information (overflow), the comparison could unexpectedly report a violation of the design limit.

In R2017a, before comparison, Simulink casts the data item and the design limit to the same data type (the data type of the data item). Simulink does not cast the data item to `double` unless you specify it. This more accurate technique can prevent the generation of unnecessary, misleading errors and warnings. However, Simulink still stores the design limit as `double` before comparison.

Version History

The more accurate comparison technique can cause new warnings or errors in R2017a when you:

- Simulate an existing model.
- Run an existing script that sets the value, minimum, or maximum of a block parameter in a model.
- Run an existing script that sets the `Value`, `Min`, or `Max` properties of a parameter object (such as `Simulink.Parameter`).

For example, the R2016b comparison technique could have caused Simulink to overlook limit violations when you used data types that have greater precision or range than `double`. In R2017a, the new comparison technique can cause these limit violations to generate warnings or errors.

- Use the Fixed-Point Tool or the Data Type Assistant to autoscale data items in a fixed-point model. For each data item, these tools can propose a scaling that enables the data item to represent the real-world minimum and maximum values that you specify. See Fixed-Point Tool (Fixed-Point Designer) and Specify Fixed-Point Data Types.
- Reduce the number of errors that you encounter while interacting with the model by:
 - Using the Fixed-Point Tool to temporarily override data types to `double`. See Control Data Type Override.
 - Adjusting the setting for the configuration parameter **Simulation range checking** from error to warning or none.
- Round a design limit to the next number furthest from zero that `double` can represent. This technique can resolve a limit violation when the data type of the data item has higher precision than `double` (for example, a fixed-point data type with a 128-bit word length and a 126-bit fraction length) and `double` cannot exactly represent the value of the design limit.

For example, if an existing model generates a new error in R2017a with the maximum value of a signal set to `98.8847692348509014`, at the command prompt, calculate the next number furthest from zero that `double` can represent.

```
format long
98.8847692348509014 + eps(98.8847692348509014)

ans =

    98.884769234850921
```

Use the resulting number, `98.884769234850921`, to replace the maximum value.

Deep copy of handle objects by `Simulink.ModelWorkspace.assignin`

Before R2017a, the `assignin` method of a `Simulink.ModelWorkspace` object did not deeply copy handle objects. As a result, modifications that you made to a handle object in the source workspace (for example, the base workspace or a function workspace) also affected the model workspace.

Suppose you used this code to create a `Simulink.Parameter` object named `myVar` and assign `myVar` into the model workspace of a model named `myModel`:

```
myVar = Simulink.Parameter(5.2);  
mdlwks = get_param('myModel', 'ModelWorkspace');  
assignin(mdlwks, 'myVar', myVar);
```

Then, if you modified `myVar` in either the base workspace or the model workspace, the modification affected `myVar` in both workspaces. Each instance of `myVar` stored a handle to the same `Simulink.Parameter` object.

In R2017a, the `assignin` method performs a deep copy so modifications that you make in the source workspace do not affect the model workspace. With this deep copy, in the example, `myVar` in the model workspace is independent of `myVar` in the base workspace.

Use of From Workspace block in a model that uses a data dictionary

Prior to R2017a, if you linked a model to a data dictionary, the model could not contain any From Workspace blocks.

In R2017a, when a model is linked to a data dictionary, From Workspace blocks can acquire input data from variables that you store in the Design Data section of the dictionary or in the other workspaces that From Workspace already supports. To use variables that you store in the other workspaces, set the value of the **Data** parameter by using a call to the `evalin` function.

Because you cannot store a `timeseries` object in the dictionary, to drive the From Workspace block by using a `timeseries` object, you must:

- 1 Place the object in the base workspace.
- 2 In the From Workspace block, set the value of the **Data** parameter to, for example, `evalin('base', 'myTimeseriesObject')`. The argument `'base'` indicates that the object is in the base workspace.

For more information, see [From Workspace](#)

Specify 64-bit integer data types without a Fixed-Point Designer license

Beginning in R2017a, specifying a 64-bit integer data type no longer requires a Fixed-Point Designer license. To specify a 64-bit integer type, use the `fixdt` function. For example, to specify an `int64` or `uint64` data type, set the output data type of a block to `fixdt(1,64,0)`, or to `fixdt(0,64,0)`, respectively.

Block Enhancements

Support for Scopes in For Each Subsystems

You can place a Scope block within a For Each Subsystem block.

If your model has a Scope block attached to the output port of a For Each Subsystem block, you can move the Scope block into the subsystem block, attach signal lines, and then delete the Output block.

Scope Blocks: Support for nonvirtual bus and array of buses signals

You can connect nonvirtual bus signals and array of buses signals to a Scope block (and to a Time Scope block if you have a DSP System Toolbox license). To display the signals in the Scope block, use normal or accelerator simulation mode. For details, see [Nonvirtual Bus and Array of Buses Signals and Save Simulation Data Using a Scope Block](#).

Specify image file icons for MATLAB System block

You can specify a MATLAB System block icon as an image file using a new option in the MATLAB Editor. While editing your System object, specify the image file by selecting **System Block > Add Image Icon**. After specifying the image file, this code is added to the System object class:

```
function icon = getIconImpl(~)
    % Define icon for System block
    icon = matlab.system.display.Icon('image.png');
end
```

For more information, see [Customize System Block Appearance](#).

Copy scope to clipboard

To share the output of a signal simulation, copy the scope graphic to your clipboard by selecting **File > Copy to Clipboard**. The scope colors are converted to a print-friendly coloring. See [Share Scope Image](#).

Interactive legend for scopes

If you show a legend on your scope block or System object, you can use the legend to filter which signals are shown. Left-clicking a signal in the legend hides all other signals in the scope. Right-clicking a signal in the legend toggles whether the scope shows or hides the signal.

Stem plot option for Scope block

In the Scope block, you can visualize your signal as a stem plot. From the **View > Style** menu, select **Plot type > Stem**.

Simulink Blocks: Simulink implements same workflow when adding a block through the user interface or the command line

When you add a block through the command line, Simulink now executes the block copy function first and then sets the specified parameter values for the block.

Before R2017a, when you add a block through the command line, the parameter values are set first and then the block copy function is executed.

Slider Gain block: Minimum and Maximum values must not be same

In R2017a and higher, the minimum values specified for the Slider Gain block must be less than the maximum value specified. The minimum and maximum values must not be same.

Default input signal attributes for MATLAB System block

In R2017a and higher, the default input signal attributes are defined if a MATLAB System block has one or more inputs that are unconnected to another block's output port or connected to a port that has underspecified attributes.

Additional calls to Propagation Methods `getOutputDataTypeImpl`, `getOutputSizeImpl` and `isOutputComplexImpl` during the model pre-compile phase

In addition to the model compile phase, propagation methods `getOutputSizeImpl`, `getOutputDataTypeImpl` and `isOutputComplexImpl` are called during the model pre-compile phase.

Math Function block `rem`, `mod`, and `pow` function changes

The Math Function block `rem` and `mod` functions have these changes:

- For generated code, Simulink now applies output signedness for the `rem` and `mod` functions when the output is 0.
- You may experience improved performance when using the `rem` and `mod` functions.

The Math Function block `pow` function has this change:

- For generated code and simulation, the `pow` function now returns `1.0` for these cases:
 - 1^{inf}
 - $(-1)^{\text{inf}}$
 - $(1)^{(-\text{inf})}$
 - $(-1)^{(-\text{inf})}$

In previous releases, the `pow` function returned NaN for these cases.

Trigonometric Function block `asin`, `asinh`, `acos`, and `acosh` function changes

The Trigonometric Function block `asin`, `asinh`, `acos`, and `acosh` functions now perform correct branch cut behavior for generated code and simulation. In previous releases:

- The `asin` and `acos` functions returned NaN in the real part of a complex number.
- The `asinh` and `acosh` functions returned NaN in the imaginary part of a complex.

Dynamic memory allocation for unbounded arrays and large arrays

In R2017a, simulation and C/C++ code generation support dynamic memory allocation for arrays in a System object associated with a MATLAB System block. Dynamic memory allocation allocates memory as needed at run time, instead of allocating memory statically on the stack. Dynamic memory allocation is beneficial when:

- You do not know the upper bound of an array.
- You do not want to allocate memory on the stack for large arrays.

By default, dynamic memory allocation is enabled. To disable it, in the Configuration Parameters dialog box, on the **All Parameters** tab, in the **Simulation Target > Advanced parameters** category, clear the **Dynamic memory allocation in MATLAB Function blocks** check box.

When dynamic memory allocation is enabled, the code generator uses dynamic memory allocation for arrays whose size is equal to or greater than a threshold. The default value of this threshold is 64 kilobytes. To change the threshold, in the Configuration Parameters dialog box, on the **All Parameters** tab, in the **Simulation Target > Advanced parameters** category, set the **Dynamic memory allocation threshold in MATLAB Function blocks** parameter.

Dynamic memory allocation does not apply to:

- Input and output signals. Variable-size input and output signals must have an upper bound.
- Parameters or global variables. Parameters and global variables must be fixed-size.
- Fields of bus arrays. Bus arrays cannot have variable-size fields.
- Discrete state properties of System objects associated with a MATLAB System block.

See [Control Memory Allocation for Variable-Size Arrays in a MATLAB Function Block](#) and [Use Dynamic Memory Allocation for Variable-Size Arrays in a MATLAB Function Block](#).

Better handling of promoted parameter

From R2017a, any change to the path of a promoted mask parameter is resolved automatically. If the underlying block of the promoted mask parameter is deleted or is moved within another mask, Simulink displays a warning during model load, model simulation, and while using the `set_param` command on the promoted parameter.

Connection to Hardware

Wireless Connectivity: Use UDP and TCP/IP blocks to let Simulink hardware targets communicate with each other

The following support packages now support wireless communication using UDP and TCP/IP blocks:

- Simulink Support Package for Android Devices
- Simulink Support Package for Apple iOS Devices
- Simulink Support Package for Arduino Hardware
- Simulink Support Package for LEGO MINDSTORMS NXT Hardware
- Simulink Support Package for Raspberry Pi Hardware

When you add UDP or TCP/IP blocks to models created in two of these support packages, the models can communicate and transfer of data to each other directly during run time on the hardware.

Support for print and println on Arduino Serial Transmit block

The Serial Transmit block now supports the print and println options so that you can print data to the serial port.

Hardware plugin detection for Arduino boards in MATLAB, Simulink

Hardware plugin detection feature helps you to identify the devices that you can use within MATLAB and Simulink when you plug in a new hardware into the host computer.

Blocks added to LEGO EV3 support package

This table lists the support for these new blocks.

Block	Usage
TCP/IP Receive	Receive TCP packets from a remote host.
TCP/IP Send	Send TCP packets to a remote host.
UDP Receive	Receive UDP packets from a remote host.
UDP Send	Send UDP packets to a remote host.

Blocks added to Raspberry Pi support package

This table lists the support for these new blocks.

Block	Usage
I2C Master Write	Write data to I2C slave device or I2C slave device register.
I2C Master Read	Read data from I2C slave device or I2C slave device register.

Block	Usage
SPI Master Transfer	Write data to and read data from SPI slave device.
SPI Register Write	Write data to registers of an SPI slave device.
SPI Register Read	Read data from registers of SPI slave device.
Serial Write	Write data to serial device.
Serial Read	Read data from serial device.
TCP/IP Receive	Receive TCP packets from a remote host.
TCP/IP Send	Send TCP packets to a remote host.
HTS221 Humidity Sensor	Measure relative humidity and ambient temperature (Sense HAT block).
LPS25h Pressure Sensor	Measure barometric air pressure and ambient temperature (Sense HAT block).
LSM9DS1 IMU Sensor	Measure linear acceleration, angular rate and magnetic field along X, Y, and Z axis (Sense HAT block).
Joystick	Read the state of five-position joystick (Sense HAT block).
8x8 RGB LED Matrix	Control pixel color of 8x8 RGB LED Matrix (Sense HAT block).

Support for all Android smartphones and tablets

The Simulink Support Package for Android Devices supports all Android smartphones and tablets using Android version 4.2 and higher.

Blocks added to Android support package

This table lists the new blocks available in the Simulink Support Package for Android Devices.

Block	Usage
TCP/IP Send	Send TCP packets to a remote host.
TCP/IP Receive	Receive TCP packets from a remote host.
Audio File Read	Use an audio file from your desktop computer on your mobile app.

Blocks added to Apple iOS support package

This table lists the new blocks available in the Simulink Support Package for Apple iOS Devices.

Block	Usage
TCP/IP Send	Send TCP packets to a remote host.
TCP/IP Receive	Receive TCP packets from a remote host.

Block	Usage
Audio File Read	Use an audio file from your desktop computer on your mobile app.

Support for Scope block on Apple iOS and Android apps

The following support packages now support the Scope block:

- Simulink Support Package for Android Devices
- Simulink Support Package for Apple iOS Devices

When you add a Scope block to a model created using one of these support packages, an equivalent real-time scope display is added to the mobile app user interface.

MATLAB Function Blocks

Dynamic memory allocation for unbounded arrays and large arrays

In R2017a, simulation and C/C++ code generation support dynamic memory allocation for arrays in a MATLAB Function block, a Stateflow chart, or a System object associated with a MATLAB System block. Dynamic memory allocation allocates memory as needed at run time, instead of allocating memory statically on the stack. Dynamic memory allocation is beneficial when:

- You do not know the upper bound of an array.
- You do not want to allocate memory on the stack for large arrays.

By default, dynamic memory allocation is enabled for GRT-based targets and disabled for ERT-based targets. To change the setting, in the Configuration Parameters dialog box, on the **All Parameters** tab, in the **Simulation Target > Advanced parameters** category, clear or select the **Dynamic memory allocation in MATLAB Function blocks** check box.

When dynamic memory allocation is enabled, the code generator uses dynamic memory allocation for arrays whose size is equal to or greater than a threshold. The default value of this threshold is 64 kilobytes. To change the threshold, in the Configuration Parameters dialog box, on the **All Parameters** tab, in the **Simulation Target > Advanced parameters** category, set the **Dynamic memory allocation threshold in MATLAB Function blocks** parameter.

Dynamic memory allocation does not apply to:

- Input and output signals. Variable-size input and output signals must have an upper bound.
- Parameters or global variables. Parameters and global variables must be fixed-size.
- Fields of bus arrays. Bus arrays cannot have variable-size fields.
- Discrete state properties of System objects associated with a MATLAB System block.

See [Control Memory Allocation for Variable-Size Arrays in a MATLAB Function Block and Use Dynamic Memory Allocation for Variable-Size Arrays in a MATLAB Function Block](#).

Nested functions

In R2017a, you can use nested functions in MATLAB code in a MATLAB Function block. When you use nested functions, adhere to these restrictions:

- If the parent function declares a persistent variable, it must assign the persistent variable before it calls a nested function that uses the persistent variable.
- A nested recursive function cannot refer to a variable that the parent function uses.
- If a nested function refers to a structure variable, you must define the structure by using `struct`.
- If a nested function uses a variable defined by the parent function, you cannot use `coder.varsize` with the variable in either the parent or the nested function.
- You cannot use nested functions in MATLAB action language used by a Stateflow chart.

Also, you must adhere to the code generation restrictions for value classes and handle classes.

Handle classes in value classes

In R2017a, in MATLAB code in a MATLAB Function block, you can use value classes that contain handle classes. The handle class can be one that you define or a predefined handle class that is available with MATLAB or a MATLAB toolbox. Predefined handle classes, such as toolbox System objects, must be supported for C/C++ code generation. See [Functions and Objects Supported for C/C++ Code Generation — Category List](#).

For example, suppose that `myclass` is a value class and `myhandle` is a handle class. A MATLAB Function block can contain code such as:

```
obj = myclass;
obj.p1 = myhandle;
obj.p2 = dsp.Mean;
```

The code generation limitations for handle class objects apply to handle class objects in value classes. See [Code generation limitations for handle class objects](#).

Constant folding of value classes

In R2017a, you can use `coder.const` to constant-fold value classes.

The code generator tries to fold constant expressions into the generated code. Constant folding uses the value of a constant expression instead of the expression in the generated code. Constant folding can improve execution time because the generated code does not have to evaluate the expression multiple times. You can try to force the code generator to constant-fold an expression by using `coder.const`.

To constant-fold a value class object `obj`, use this syntax:

```
coder.const(obj)
```

To constant-fold the property `prop`, use this syntax:

```
coder.const(obj.prop)
```

Class properties and structure fields passed by reference to external C functions

To pass arguments by reference to an external C function, you use `coder.ref`, `coder.rref`, or `coder.wref` in a `coder.ceval` call. For example:

```
...
x = 1;
y = coder.ceval('myCFunction', coder.ref(x));
...
```

In previous releases, the argument that you passed by reference had to be a scalar variable or an element of an array. To pass a class property or structure field, you had to first assign the property or field to a variable. For example:

```
...
x = myClass;
x.prop = 1;
```



```
v = x;
coder.ceval('foo', coder.ref(v));
...
```

In R2017a, you can directly pass a class property or structure field by reference. For example:

- Pass a class property

```
...
x = myClass;
x.prop = 1;
coder.ceval('foo', coder.ref(x.prop));
...
```

- Pass a structure field

```
...
s = struct('s1', struct('a', [0 1]));
coder.ceval('foo', coder.wref(s.s1.a));
...
```

- Pass a field of an element of an array of structures

```
...
s = struct('c', [1 2], 'd', 2);
s1 = struct('a', [s s]);
coder.ceval('foo', coder.rref(s1.a(1).d));
...
```

Function specialization prevention with `coder.ignoreConst`

At compile time, if an input argument to a function call evaluates to a constant, the code generator can use the constant value to produce function specializations. A function specialization is a version of a function in which the input type, size, complexity, or value is customized for a particular invocation of the function. To prevent function specializations due to constant arguments, instruct the code generator to treat the value of the argument as a nonconstant value by using `coder.ignoreConst`.

With compile-time recursion, the code generator produces function specializations instead of a recursive call. If the specializations are due to a constant input argument to the recursive function, you might be able to force run-time recursion by using `coder.ignoreConst`. See [Force Code Generator to Use Run-Time Recursion](#).

New `coder.unroll` syntax for more readable code

In R2017a, `coder.unroll` has a new syntax that helps make your code more readable.

In previous releases, you put `coder.unroll` inside a `for`-loop. For example:

```
...
for i = coder.unroll(1:n)
    y(i) = rand();
end
...
```

With the new syntax, you put `coder.unroll` on a line by itself, immediately before the loop that it unrolls. For example:

```
...
coder.unroll();
for i = 1:n
    y(i) = rand();
end
...
```

Here is an example of the new syntax with the `flag` argument:

```
...
unrollflag = n < 10;
coder.unroll(unrollflag);
for i = 1:n
    y(i) = rand();
end
...
```

Both the new syntaxes and the syntaxes from previous releases are supported. For more readable code, use the new syntax.

For more information about `coder.unroll` and for-loop unrolling, see `coder.unroll` and `Unroll for-Loops`.

Size argument for `coder.opaque`

In R2017a, you can specify the size of a variable that you declare with `coder.opaque`. The syntax with the size argument is:

```
x = coder.opaque(type,value,'Size', size)
```

Specify the size in bytes. For example, declare `x1` to be a 4-byte integer with initial value 0.

```
x1 = coder.opaque('int','0','Size', 4);
```

Code generation for more MATLAB functions

- `cholupdate`
- `histcounts`
- `ismethod`

Code generation for more Audio Toolbox System objects

`audioPlayerRecorder`

For C/C++ code generation usage notes and limitations, see the reference page.

Code generation for more Communications System Toolbox System objects

`comm.RBDSWaveformGenerator`

For C/C++ code generation usage notes and limitations, see the reference page.

Code generation for more DSP System Toolbox System objects

- `dsp.HampelFilter`
- `dsp.AsyncBuffer`

For C/C++ code generation usage notes and limitations, see the System object reference page.

Code generation for more Phased Array System Toolbox System objects

- `bw2range`
- `diagbfweights`
- `scatteringchanmtx`
- `waterfill`
- `phased.BackScatterSonarTarget`
- `phased.DopplerEstimator`
- `phased.IsoSpeedUnderWaterPaths`
- `phased.IsotropicHydrophone`
- `phased.IsotropicProjector`
- `phased.MultipathChannel`
- `phased.RangeEstimator`
- `phased.RangeResponse`
- `phased.ScatteringMIMOChannel`

For C/C++ code generation usage notes and limitations, see the function or System object reference page.

Code generation for more Robotics System Toolbox functions and classes

- `robotics.AimingConstraint`
- `robotics.Cartesianbounds`
- `robotics.GeneralizedInverseKinematics`
- `robotics.InverseKinematics`
- `robotics.Joint`
- `robotics.JointPositionBounds`
- `robotics.PoseTarget`
- `robotics.PositionTarget`
- `robotics.OrientationTarget`
- `robotics.RigidBody`
- `robotics.RigidBodyTree`
- `transformScan`

For C/C++ code generation usage notes and limitations, see the function or class reference page.

Code generation for more Signal Processing Toolbox functions

- `alignsignals`
- `cconv`
- `convmtx`
- `corrmtx`
- `envelope`
- `finddelay`
- `hilbert`
- `sgolayfilt`
- `sinc`
- `xcorr2`
- `xcov`

For C/C++ code generation usage notes and limitations, see the function or class reference page.

Statistics and Machine Learning Toolbox Code Generation: Generate C code for prediction by using linear models, generalized linear models, decision trees and ensembles of classification trees

You can generate C code that predicts responses by using trained linear models, generalized linear models (GLM), decision trees, or ensembles of classification trees. The following prediction functions support code generation.

- `predict` — Predict responses or estimate confidence intervals on predictions by applying a linear model to new predictor data.
- `predict` or `glmval` — Predict responses or estimate confidence intervals on predictions by applying a GLM to new predictor data.
- `predict` or `predict` — Classify observations or estimate classification scores by applying a classification tree or ensemble of classification trees, respectively, to new data.
- `predict` — Predict responses by applying a regression tree to new data.

For C/C++ code generation usage notes and limitations, see the function reference page.

Enhancement to synchronous subsystem support

For a MATLAB Function block inside a synchronous subsystem, you can now use the combinational and sequential logic portions in one MATLAB function. Previously, you created two separate MATLAB Function blocks, one for the combinational logic, and the other for the sequential logic.

To use the combinational and sequential logic portions inside one MATLAB Function block, in the Ports and Data Manager dialog box, select the **Allow direct feedthrough** checkbox. The output function can then depend on inputs and persistent variables. For example, you can now use this MATLAB function that has two outputs, with one output depending on the input, and the other output depending on a persistent variable.

```
function [y1, y2] = fcn(u, v)
```

```
persistent p;  
if isempty(p)  
    p = uint8(0);  
end  
  
y1 = p;  
y2 = v;  
  
p = u;
```

If you have HDL Coder™, you can use the MATLAB Function block inside a synchronous subsystem to generate cleaner HDL code and use fewer hardware resources. See also State Control.

Support for tunable structure array parameters

You can now use a tunable structure array parameter with the MATLAB Function block. Previously, you used a tunable structure of scalar values or nontunable parameters with structure arrays.

State behavior specification for function-call input events

If you define a function-call input event for a MATLAB Function block, you can now specify the state behavior when this event reenables the block. To specify this behavior, use the **States When Enabling** block parameter located in either:

- The **Properties > Advanced** section of the Property Inspector
- The Ports and Data Manager dialog box

When you set **States When Enabling** to `held`, the simulation maintains the most recent values of the states when the function-call event reenables the MATLAB Function block. If you set **States When Enabling** to `reset`, the function-call event reverts states to their initial conditions.

S-Functions

Functionality being Removed or Changed

Functionality	Result	Use Instead	Compatibility Considerations
Level-1 Fortran S-Functions	Still runs.	Level-2 Fortran S-Functions	Level-1 Fortran S-Functions will continue to work but no documentation support will be provided. Use the Level-2 Fortran S-Functions instead.

R2016b

Version: 8.8

New Features

Bug Fixes

Version History

Simulation Analysis and Performance

Just-in-Time Acceleration Builds: Quickly build the top-level model for improved performance when running simulations in Accelerator mode

When you simulate a model in accelerator mode, Simulink now uses Just-in-Time (JIT) acceleration to speed up the building of the accelerator target for the top-level model. Using JIT, Simulink generates an execution engine in memory instead of generating C code or MEX files during simulation. JIT provides the best performance for the generation of the accelerator target for a model.

For more information, see [Accelerator Mode](#), or watch this video to learn more.

If you want to simulate your model using the classic, C-code generating, accelerator mode, run the following command:

```
set_param(0, 'GlobalUseClassicAccelMode', 'on');
```

Dataset Signal Plot: View and analyze dataset signals directly from the MATLAB command line

Use `plot` to plot dataset signals in the Signal Preview window for `Simulink.SimulationData.Dataset` and `Simulink.SimulationData.DatasetRef` objects.

Watch this video to learn more.

Multi-State Image Dashboard Block: Display different images based on the signal value

The Dashboard block library now includes a Multi-state Image block that you can use to display changes in signal values. After adding the block to your model, double-click the block and select signals whose states you want to capture visually. For each state, select a value and an image to display. When the value of a signal changes, the block displays the image that you set up for that particular value.

Simplified tasking mode setup

Previously, you set the tasking mode for your model using the **Tasking mode for periodic sample times** drop-down in the **Configuration Parameters** dialog box. You were also able to set the tasking mode programmatically using the `SolverMode` parameter.

In R2016b, a simple check box labeled **Treat each discrete rate as a separate task** replaces the drop-down. The command-line equivalent of this parameter is `EnableMultiTasking`.

	Tasking mode for periodic sample times	Treat each discrete rate as a separate task
Command-line parameter	<code>SolverMode</code>	<code>EnableMultiTasking</code>
Parameter values	Auto, Multitasking	On (enables multitasking mode)

	Tasking mode for periodic sample times	Treat each discrete rate as a separate task
	SingleTasking	Off (enables single-tasking mode)

If you select the check box, single-tasking mode is used in these cases:

- Your model contains one sample time.
- Your model contains a continuous and a discrete sample time, and the fixed step size is equal to the discrete sample time.

For more information, see [Treat each discrete rate as a separate task](#).

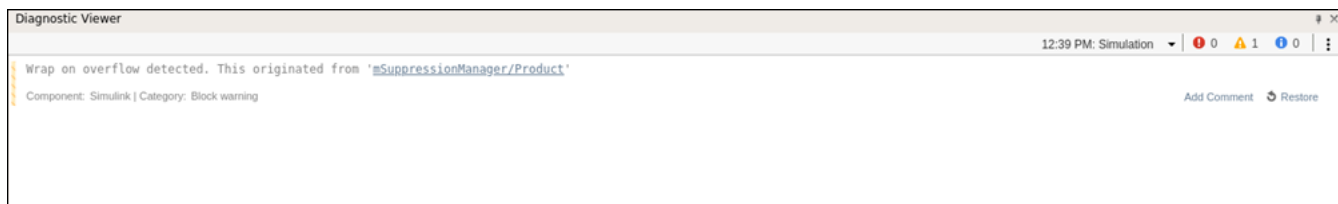
Version History

By default, new models now have the single-tasking mode enabled, whereas they had the Auto tasking mode enabled previously.

Further, if you have references to `SolverMode` in your scripts, replace those references with `EnableMultiTasking`. Use the permissible values for `EnableMultiTasking` to specify the tasking mode.

Diagnostic Suppressor: Suppress specific simulation warnings on particular blocks

The Diagnostic Viewer in Simulink now includes an option to suppress certain diagnostics. This feature enables you to suppress warnings for specific objects in your model. Click the **Suppress this warning** button next to the warning in the Diagnostic Viewer to suppress the warning from the specified source. You can restore the warning from the source by clicking **Restore this warning**.



You can also control the suppressions from the command line. For more information, see [Suppress Diagnostic Messages Programmatically](#).

Diagnostic Viewer: Improved build diagnostics display

Diagnostic Viewer detects compiler errors and warnings generated during model build and reports them with the ability to navigate to the source of the problem.

Export functions allow periodic function calls

Export functions have been enhanced to allow periodic function calls. In addition, you can now use the sample time of each export function call in your model to control the scheduling of triggers. The order of the function-call triggers is determined by the sample times of their corresponding export-

function blocks. Using the sample time provides more fine-grained control over how function-call triggers are scheduled in your model.

These enhancements provide the following advantages:

- Multiple exported functions can have the same execution period.
- Simulink now checks for sample time consistency in export functions.
- Export-function models support `auto` step size and dataset logging.
- Sample time optimization results in more efficient code.

For more information, see [Export-Function Models](#).

Version History

As a result of this enhancement, the root output logging behavior for export-function models in standalone simulations has changed. Previously, logging of root output ports for export-function models was inconsistent with the multitasking behavior of regular Simulink models, resulting in missing periodic rates in the logged output. Now, export-function models supports multitasking logging and dataset logging, which enable correct logging of all periodic rates from root output ports.

If you rely on the previous logging behavior, you can enable dataset logging to get the old output back. Note that dataset logging does not support asynchronous rate. In those cases, specify the base rate in the root input ports.

Simulink Editor

Property Inspector: Edit parameters and properties of model elements using a single interface

The Simulink Editor enables you to use a single interface to edit parameters and properties for any model element. When you display the Property Inspector, it becomes part of the model editing window. Select **View > Property Inspector** to open this interface. You can leave the Property Inspector open as you build your model, and it updates with your selection. To learn more about setting properties, see [Setting Properties and Parameters](#), or watch this video to learn more.

You can use the Property Inspector to edit:

- Block parameters and properties
- Stateflow elements
- Annotations, areas, and images
- Signals
- Model properties

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

The **Errors and Warnings** option enables edit-time checking, providing you with visual cues about issues with your model. To display information about an issue, hover the cursor over the highlighted block and click the error or warning icon.

Simulink detects the following block errors and warnings:

- Goto and From block mismatches.
- Duplicate data store blocks. The value of the Duplicate data store names parameter determines if there is a visual cue, and if the cue is an error or warning.

The **Errors and Warnings** option, is enabled by default. To turn off this option, in the model window, select **Display > Errors & Warnings**.

For more information, see [Model Design Error Detection](#).

Finder: Search for model elements using improved interface

The interface for searching by way of **Edit > Find (Ctrl+F)** is improved in R2016b. To learn how to search using this interface, see [Find Model Elements in Simulink Models](#).

Annotations in Libraries: Add annotations from libraries into models

In R2016b, you can add annotations to custom libraries for the user of the library to add to their model. To learn how to add an annotation to a library that appears in the Library Browser, see [Create a Custom Library](#). Add annotations to your model from a library using the same techniques you use for adding blocks.

Library Browser: Expand or collapse libraries by default

You can customize the display of any library in the Library Browser tree to expand or collapse by default. See [Customize Library Browser Appearance](#).

Library Browser API: Programmatically refresh the Library Browser

Using the `refresh` method with the `Library.LibraryBrowser2` class, you can programmatically refresh the library browser. To learn more, see `LibraryBrowser.LibraryBrowser2`.

Version History

In previous releases, these two commands returned `LibraryBrowser.LibraryBrowser2`:

```
lb = slLibraryBrowser
lb = LibraryBrowser.LibraryBrowser2
```

In R2016b, each returns `LibraryBrowser.LBStandalone`. However, the API to access Library Browser operations such as `show` and `refresh` has not otherwise changed.

Annotations: Click once to select annotation

In previous releases, a single click on an annotation prompted you to edit the annotation text instead of selecting the annotation frame. In R2016b, click one time to select the annotation frame, which you can then drag or right-click to access the content menu. Click a selected annotation to edit the text.

Default Model Font: Specify default font for model elements

You can now set the default font for models. Setting the default font affects any existing model elements whose font you have not set manually as well as new model elements. In your model, with no selection, select **Diagram > Format > Font Styles for Models**. See [Specify Fonts in Models](#). To use that font in new models, set the default font in the default template. See [“Default Model Template: Use your own customized settings when creating new models”](#) on page 12-12.

Simulink Preferences: Simplified and reorganized interface

Simulink Preferences now includes three panes:

- **General** pane for generated file folders, background colors for printing and exporting, and model display options
- **Editor** pane for customizing the Simulink Editor
- **Model File** pane for file saving options, including version notifications

Settings that are saved with the model no longer appear in Preferences. Use default templates for these settings instead of preferences. For more information, see [“Default Model Template: Use your own customized settings when creating new models”](#) on page 12-12.

For more information on the new Simulink Preferences panes, see [Set Simulink Preferences](#).

Simulink Editor Fonts: FreeType font engine replaces Windows GDI font engine

On Windows system, Simulink now uses the FreeType font engine. In previous releases, it used the Windows graphics device interface (GDI) font engine. This change improves the appearance of text in the user interface of the Simulink Editor.

If you are using a language that uses non-Latin characters, you might need to change the font in the editor to one that supports your language. To set defaults for a model, with no selection in the Simulink Editor, select **Diagram > Format > Font Styles for Model**. Change the font using the dialog box.

Component-Based Modeling

Initialize and Terminate Function Blocks: Respond to events to model dynamic startup and shutdown behavior

Use an Initialize Function block with a State Writer block to set the state of a block in response to an initialize event. By default, the Initialize Function block includes a State Writer block and an Event Listener block with the **Event** parameter set to `Initialize`.

Use the Terminate Function block with a State Reader block to read the state of a block in response to a terminate event. By default, the Terminate Function block includes a State Reader block and an Event Listener block with the **Event** parameter set to `Terminate`.

Applications for these new blocks include:

- Starting up and shutting down an application component
- Initial condition calculations
- Save and restore state from nonvolatile memory

For more information, see Initialize Function, Terminate Function, Event Listener, State Reader, and State Writer block reference. Watch this video to learn more

Variant Subsystem Condition Propagation: Automatically assign variant conditions to blocks outside the subsystem for improved performance

When you specify variant conditions in models containing Variant Subsystem blocks, Simulink propagates these conditions outside to determine which components of the model are active during simulation.

A variant condition can be either a condition expression or a variant object.

For more information, see Condition Propagation with Variant Subsystem, or watch this video to learn more.

Simulink Units Updates

These are changes to the Simulink Units:

- Allowed Units list:
 - The ASCII symbol of the `reduced_Planck_constant` unit has changed to `h_bar`. In R2016a, this symbol was `hbar`. Starting in R2016b, `hbar` refers to hectobar.
 - Starting in R2016b, the dalton unit no longer allows the use of SI prefixes. In R2016a, the dalton unit incorrectly allowed the use of SI prefixes.
 - Starting in R2016b, the `deg` unit is listed in both the English and SI (extended) units lists. In R2016a, the `deg` unit was listed in only the SI (extended) list.
 - Starting in R2016b, the `au` unit refers to astronomical unit. In R2016a, `au` incorrectly referred to attodalton.

- Unit matching — As you type your unit in the **Unit** parameter, the software now suggests close, but not exact, unit matches highlighted with a star.
- Unit mismatching handling — Now when you click on the mismatched units warning badge, the dialog lets you go to either the source or destination where the unit is specified.
- Undefined units handling — The Model Advisor check now has a column with suggested unit alternatives that are a close match to the Simulink unit database.

Additional SimStruct Functions to Specify Units for Input and Output Ports

To specify units for S-function input and output ports, use these `SimStruct` functions:

- `ssGetInputPortUnit`
- `ssGetOutputPortUnit`
- `ssRegisterUnitFromExpr`
- `ssSetInputPortUnit`
- `ssSetOutputPortUnit`

Additional heterogeneous targets supported for concurrent execution

When configuring a model for concurrent execution, you can now build and download partitions of the model to these additional heterogeneous targets:

- Altera® Cyclone® SoC Rev. C development kit target
- Altera Cyclone SoC Rev. D development kit target
- Arrow® SoCKit development board target

For more information, see `Specify a Target ArchitectureExport-Function Models`.

Simulink.BusElement: SamplingMode property removed to support having blocks specify whether to treat inputs as frame-based signals

The `SamplingMode` property of `Simulink.BusElement` objects has been removed in R2016b. Specify the sampling mode (sample-based and frame-based) of input signals at the block level instead of at the signal level.

Version History

Scripts that use the `SamplingMode` property of `Simulink.BusElement` objects continue to work in R2016b. However, support for `SamplingMode` will be removed in a future release.

Export functions allow periodic function calls

Export functions have been enhanced to allow periodic function calls. In addition, you can now use the sample time of each export function call in your model to control the scheduling of triggers. The order of the function-call triggers is determined by the sample times of their corresponding export-function blocks. Using the sample time provides more fine-grained control over how function-call triggers are scheduled in your model.

These enhancements provide the following advantages:

- Multiple exported functions can have the same execution period.
- Simulink now checks for sample time consistency in export functions.
- Export-function models support `auto` step size and dataset logging.
- Sample time optimization results in more efficient code.

For more information, see [Export-Function Models](#).

Version History

As a result of this enhancement, the root output logging behavior for export-function models in standalone simulations has changed. Previously, logging of root output ports for export-function models was inconsistent with the multitasking behavior of regular Simulink models, resulting in missing periodic rates in the logged output. Now, export-function models supports multitasking logging and dataset logging, which enable correct logging of all periodic rates from root output ports.

If you rely on the previous logging behavior, you can enable dataset logging to get the old output back. Note that dataset logging does not support asynchronous rate. In those cases, specify the base rate in the root input ports.

Variant Refresh: Improved performance with removal of live refresh

Simulink models offer better performance with the removal of live refresh of active models that contain Variant Subsystem blocks.

To update or refresh such models, click **Diagram > Refresh Blocks** (Ctrl + K) or **Simulation > Update Diagram** (Ctrl + D) in Simulink.

Variant Subsystem: Convert Subsystems with physical ports to Variant

You can convert a Subsystem block that contains physical ports to a Variant Subsystem block.

To do so, right-click the Subsystem block, in the context menu, click **Subsystem & Model Reference > Convert Subsystem to > Variant Subsystem**.

Variant Reducer: Additional model reduction modes in Variant Reducer (requires SLDV product license)

Use the Variant Reducer to reduce a variant model based on any one of these reduction modes:

- Variant control variable values in the global workspace.
- Variant control values specified as a comma-separated list.
- Variant configuration data object.

For more information, see [Reduce Models Containing Variant Blocks](#).

Enhanced `find_mdhrefs` function: Keep models loaded that the function loads

The `find_mdhrefs` function loads models in the model reference hierarchy of the model that you specify. By default, the function closes those models, except for models that were already loaded before execution of the function. You can now use the `KeepModelsLoaded` name-value pair to keep all the models loaded that the function loads.

Subsystem conversion to referenced models: Automatic subsystem wrapper and improved `Goto` and `From` block handling

The Model Reference Conversion Advisor and `Simulink.SubSystem.convertToModelReference` function conversion creates a wrapper subsystem automatically if the conversion modifies the Model block interface by adding ports. The wrapper subsystem preserves the layout of the top model.

The conversion process no longer generates an error when `Goto` and `From` block pairs cross model reference boundaries. The conversion automatically fixes the model, regardless of how you set the **Fix errors automatically** (`'Autofix'`) option. The conversion now produces a message highlighting the changes it made, so that you can determine whether the changes produce the desired results.

For more information, see [Convert a Subsystem to a Referenced Model](#) and `Simulink.SubSystem.convertToModelReference`.

Disallow multiple iterations of root `Inport` function-call with discrete sample time

For a function-call root `Inport` block with a discrete sample time vector, you cannot have multiple times at the same sample time.

Project and File Management

Default Model Template: Use your own customized settings when creating new models

In R2016b, you can specify a model template to use for all new models. Create a model with the configuration settings and blocks you want, then export the model to a template. To reuse these settings in every new model, make the new template your default model template using the Simulink start page or the `Simulink.defaultModelTemplate` function.

In R2016b, after setting a default model template, every new model uses that template, for example, when you press **Ctrl+N**, when you use new model buttons, or when you use `new_system`. In the Simulink Editor, the default template name appears at the top of the list when you select **File > New > MyDefaultTemplateName**.

For details, see [Use Customized Settings When Creating New Models](#), or watch this video to learn more.

Version History

Action	R2016a	R2016b
Create new models using: <ul style="list-style-type: none"> • Ctrl+N in the Editor or start page. • New model toolbar buttons in the Editor and Library Browser • <code>new_system</code> 	Factory default blank model	You can specify your default template. Until you choose a new default template, the default template remains the blank model.
Specify configuration setting defaults	You could use Simulink Preferences to specify default parameters for new models.	The configuration defaults, font defaults, and display defaults are removed from the Simulink Preferences dialog box in R2016b. Use a model template instead. If you previously saved configuration defaults in the Simulink Preferences, in R2016b, Simulink converts these settings to a model template, called Converted Simulink Preferences in the start page. If you previously specified configuration defaults and ran scripts on the root block diagram, then to use the old behavior, run the script <code>enablePreferencesCompatibilityMode</code> .
Specify model fonts	You could not change fonts in existing models.	In the Simulink Editor, with no model element selected, choose Diagram > Format > Font Styles for Model . To use the specified fonts as defaults for new models, export the model to a template.

Upgrade Advisor API: Automate the process of upgrading large model hierarchies

In R2016b, you can call `upgrade` or `analyze` on the output argument of the `upgradeadvisor` function. You can use the API to analyze only or also perform automatic fixes where available.

For details, see `upgradeadvisor`, or watch this video to learn more.

Project-Wide Search: Search inside all models and supporting files

Search across all your project files in one place. You can find matches inside model files, MATLAB files, and other project files such as PDF and Microsoft Word files. On the **Simulink Project** tab, click **Search** and enter some characters to search for.

Open search results to locate specific items, e.g., highlight blocks in models, specific lines in MATLAB files. Filter results by file type, status, or label.

For details, see `Search Inside Project Files` and `Filter File Views`.

Refactoring Tools: Rename folders and automatically replace all references

In a Simulink project, when you rename a folder, the project checks for impact in other project files and offers fixes. You can find and fix impacts and avoid refactoring pain tracking down other affected files. Automatic renaming is helpful in preventing errors that result if you change names or paths manually and overlook or mistype one or more instances of the name.

For details, see `Automatic Updates When Renaming, Deleting, or Removing Files`.

Project Toolbox Analysis: Find products and toolboxes used by a project

In a Simulink project, in the Impact graph view, you can find the required toolboxes for the whole project or for selected files. You can see which products a new team member requires to use the project, or find which file is introducing a product dependency.

For details, see `Find Required Toolboxes`.

Project Derived File Analysis: Find out-of-date .p, .slxp, and .mex files in a project

In a Simulink project, you can run checks to ensure your derived files are up to date, and choose to rebuild the files.

After running dependency analysis, the Impact view shows:

- Relationships between source and derived files (such as `.m` and `.p` files, `slx/slxp`, `ssc/sscp`, `c/mex` files).
- Warnings when any derived file is out of date compared to its source file.

New project checks help you manage derived files:

- You can automatically rebuild out-of-date P-code files by running the project checks.
- If you rename a source file, the project detects impact in the derived file and prompts you to update it.

For details, see [Check Dependency Results and Resolve Problems](#).

Project Export Profiles: Share specified files to zip archive

You can specify export profiles to control the project files you send to a zip archive. You can choose to exclude files, for example, when sharing a release version.

For details, see [Archive Projects in Zip Files](#).

Project Batch Job Report: Archive results in a document

After running a batch job on project files, you can view and archive results in a batch job file. For details, see [Run a Simulink Project Batch Job and Publish Report](#).

Git Submodules: Include submodules in your project

You can specify Git submodules to include in your project, to reuse code from another repository.

For details, see [Add Git Submodules](#).

C/C++ file dependency analysis: View dependencies between C/C++ source and header files in the Impact graph

Use Simulink Project dependency analysis to explore dependencies between C/C++ source and header files in the impact graph.

For details, see [Perform Impact Analysis](#).

Updated source control SDK: Write a source control integration providing file-based actions and annotations

You can use the source control Software Development Kit (SDK) to integrate Simulink Project with third-party source control tools. You can now use the updated SDK to write a source control integration that can provide file-based actions and annotations. An integration written using this API works with R2016b onwards. If you require a backwards-compatible integration, you can continue to use the old interfaces.

For details, see [Write a Source Control Integration with the SDK](#).

Diff Tools: Customize external source control tools to use MATLAB to compare and merge

Use MATLAB Comparison tool to review changes to files such as MAT data and data dictionaries from your external source control client.

For details, see [Customize External Source Control to Use MATLAB for Diff and Merge](#).

SVN Cleanup: Fix problems with working copy locks

Remove stale working copy locks using SVN Cleanup. See [Get SVN File Locks](#).

Command-line Impact Analysis: Update and analyze the dependencies graph programmatically

In R2016b, you can access the dependency graph using the Simulink project API. You can perform command-line project impact analysis and get required files.

For details, see [Get File Dependencies](#).

Data Management

Model Data Editor: Configure model data properties using a table within the Simulink Editor

The Model Data Editor allows you to inspect and edit data items (signals, data stores, and parameters) in a list that you can sort, group, and filter. You can then configure properties and parameters without having to locate the items in the block diagram.

While creating and debugging a model, you can configure multiple data items at once by selecting the corresponding signals and blocks in the block diagram. Work with the selected items in the Model Data Editor instead of opening individual dialog boxes.

Use the Model Data Editor to configure:

- Instrumentation for signals and data stores, which means you want to view and collect the simulation values. For example, you can configure signals and data stores to stream values to the Simulation Data Inspector.
- Design attributes such as data type, minimum and maximum value, and physical unit. You specify these attributes to control:
 - The values of numeric block parameters, such as the **Gain** parameter of a Gain block.
 - The interaction (interface) between components through Inport and Outport blocks and data stores.
- Storage classes for code generation. See Model Data Editor for applying storage classes to Inport blocks, Outport blocks, signals, and Data Store Memory blocks.

To use the Model Data Editor, in a model, select **View > Model Data**. See Configure Data Properties by Using a Table, or watch this video to learn more.

Output Logging: Log data incrementally, with support for rapid accelerator mode and variant conditions

For logging output (**Configuration Parameters > Data Import/Export > Output**) using Dataset format:

- If you enable **Configuration Parameters > Data Import/Export > Log Dataset data to file**, output data is logged to a MAT-file, using only a small constant amount of memory. Before R2016b, when you accessed the output logged data, Simulink loaded the entire data at once. Now Simulink loads that data into the workspace incrementally. For information about logging to a MAT-file, see Log Data Using Persistent Storage.
- In addition to the simulation modes previously supported, you can now log to memory or to a MAT-file in rapid accelerator mode.
- For the active variant condition, Simulink creates a Dataset object with the logged data. For inactive variant conditions, Simulink creates MATLAB timeseries with zero samples.

Logging Inside For Each Subsystem: Log signals inside a For Each subsystem by marking lines with antennas

Performing signal logging in For Each subsystems no longer requires adding an Outport block outside of the subsystem for each signal that you want to log. You can mark for signal logging nonbus signals inside For Each subsystems. Directly logging signals in a For Each subsystem simplifies the model layout and editing. For details, see Log Signals in For Each Subsystems.

Logged Dataset Data Analysis: Call same function for all timeseries objects in logged Dataset data

Use the new `Simulink.SimulationData.forEachTimeseries` function to call a built-in or user-defined function on each MATLAB `timeseries` object that you specify as input. For example, you can use this function to resample every element of a structure of `timeseries` objects obtained by logging a bus signal. You can use `forEachTimeseries` with a function that returns a scalar. You can specify a `timeseries` object, an array of `timeseries`, a structure with `timeseries` at leaf nodes, or an array of structures with `timeseries` at leaf nodes.

Scalar expansion of initial value for data store

Before R2016b, if a data store (Data Store Memory block or `Simulink.Signal` object) represented an array (nonscalar) signal, you specified the initial value of the data store by using an array of the same dimensions. You could not use scalar expansion by specifying a scalar initial value. For several other blocks, scalar expansion enables you to specify the same initial value for each element of a nonscalar signal.

In R2016b, you can take advantage of scalar expansion for data stores with some exceptions. For more information, see Specify Initial Value for Data Store.

Technique to determine whether signal has variable size

You can use the programmatic parameter `CompiledPortDimensionsMode` of a block output port to determine whether the associated signal is a variable-size signal. For more information, see Programmatically Determine Whether Signal Line Has Variable Size.

View your model configuration parameters as a group on the All Parameters tab

Previously, in the Configuration Parameters dialog box, the **All Parameters** tab grouped configuration parameters only by the category that was displayed in the first column. In R2016b, this tab groups the parameters for each category into the same groups used on the **Commonly Used Parameters** tab. Headings that describe the category and group replace the **Category** column. The new **Advanced Parameters** group for each category contains parameters only available on the **All Parameters** tab.

For more information, see Configuration Parameters Dialog Box Overview.

Enhanced error reporting and extended syntax for specifying argument dimensions for function specifications in Legacy Code Tool

In R2016b, these features enhance the usability of the Legacy Code Tool:

- Improved error reporting — provides more specific information about argument specification errors.
- Extended syntax for specifying argument dimensions — In addition to the `size` function, you can specify function argument dimensions using expressions with:
 - The `numel` function
 - Arithmetic operators: `+`, `-`, `*`, `/`
 - Integer and floating-point literals

Class to package and share breakpoint and table data for lookup tables

You can use the new classes `Simulink.LookupTable` and `Simulink.Breakpoint` to store table and breakpoint data for lookup tables.

When you share data between lookup table blocks by using variables in a workspace or data dictionary, use this technique to improve model readability by:

- Clearly identifying the data as parts of a lookup table.
- Explicitly associating breakpoint data with table data.

For more information, see [Package Shared Breakpoint and Table Data for Lookup Tables](#).

Simulink Coder and Embedded Coder enable you to use this technique to package the data in the generated code for calibration according to the ASAP2 and AUTOSAR standards. See [Storage of lookup tables for calibration according to ASAP2 and AUTOSAR standards](#).

These blocks have additional parameters to support the use of `Simulink.LookupTable` and `Simulink.Breakpoint` objects:

- n-D Lookup Table
- Prelookup
- Interpolation Using Prelookup

Root Inport Mapping Tool Updates

The Root Inport Mapping Tool has the following updates:

- Select a subset of scenarios for mapping and choose a mapping mode different from other scenarios (for more information, see [Map Signal Data to Root Inports](#)).
- If you import from a sheet whose name does not follow MATLAB variable name rules, the Root Inport Mapping tool now uses a modified sheet name. For more information, see [Supported Microsoft Excel File Formats](#).

Option to disable resolution of signals and states to Simulink.Signal objects

To configure signal or state properties such as data type, minimum and maximum values, and storage class, you can create a `Simulink.Signal` object in a workspace or data dictionary. The signal property **Signal name must resolve to Simulink signal object**, the block parameter **State name must resolve to Simulink signal object**, and the model configuration parameter **Signal resolution** control how the signal or state name corresponds to the object.

In R2016b, **Signal resolution** has an additional option, `None`. When selected, this setting prevents the signals, states, Stateflow data, and MATLAB Function block data in the model from acquiring settings from `Simulink.Signal` objects. Use this setting to reduce the dependency of the model on variables and objects in workspaces and data dictionaries, which can improve model portability, readability, and ease of maintenance.

However, selecting `None` does not affect data stores that you define by creating `Simulink.Signal` objects (instead of Data Store Memory blocks).

For more information about **Signal resolution**, see [Signal resolution](#).

Help fixing configuration errors from Diagnostic Viewer

For some configuration error messages, the Diagnostic Viewer provides these methods for fixing the error:

- To fix simple configuration errors, click the **Fix** button. The software updates the parameter. Green or red text in the diagnostic viewer reports if the change is successful. To see the highlighted parameter in the Configuration Parameters dialog box before fixing, click the parameter link.

Note If you are modifying a referenced configuration, or if the parameter change results in a model reference incompatibility, a dialog box warning allows you to cancel the operation.

- To fix a configuration error with a more complex solution or multiple solutions, click the **Open** button next to your chosen solution. Then, update the highlighted parameter in the Configuration Parameters dialog box.

Metadata for Logging to Persistent Storage: Simulation metadata contains persistent storage logging settings to facilitate analysis of data from multiple simulations

R2016a introduced the option for you to log `Dataset` format simulation data to persistent storage in a MAT-file. To do so, select **Configuration Parameters > Data Import/Export > Log Dataset data to file**. In R2016b, Simulink stores metadata about logging to persistent storage. When you run multiple simulations using parallel simulations or batch processing, specify a different persistent storage MAT-file for each simulation. Then you can use the metadata to provide context for analyzing the logged data stored in each file.

If you use single simulation output, Simulink creates a `Simulink.SimulationMetadata` object. In R2016b, this object includes in its `ModelInfo` structure a new `LoggingInfo` structure that contains two fields:

- `LoggingToFile` — Indicates whether logging to persistent storage is enabled ('on' or 'off')
- `LoggingFileName` — Specifies the resolved file name for the persistent storage MAT-file (if `LoggingToFile` is 'on')

The MAT-file used for persistent storage contains a new `SimulationMetadata` variable that stores the same simulation metadata as the `Simulink.SimulationMetadata` object.

For details, see [Save Logged Data from Successive Simulations](#).

Improved display of large arrays by Model Explorer and Simulink.Parameter property dialog boxes

Before R2016b, for numeric MATLAB variables and `Simulink.Parameter` objects, the Model Explorer **Value** column displayed large arrays as read-only text, such as `<3x5x3 double>`. The property dialog box for `Simulink.Parameter` objects also used this read-only text.

In R2016b, the Model Explorer and property dialog boxes display the entire value of large arrays. To modify the elements in the array, you can edit the displayed text.

Arrays with three or more dimensions appear as an expression that contains a call to the `reshape` function. To edit the values in the array, modify the arguments of this `reshape` call.

For more information, see [Edit and Manage Workspace Variables Used by Models](#).

Configuration set in base workspace resolves variables in base workspace

For a model that references a configuration set in the base workspace, if the configuration set uses a variable, Simulink resolves the variable in the base workspace.

Connection to Hardware

Raspberry Pi 3 Support: Run Simulink models on Raspberry Pi 3 hardware

The Simulink Support Package for Raspberry Pi Hardware now supports Raspberry Pi 3 hardware.

Watch this video to learn more.

Arduino: Improved External mode over serial communication

The external mode feature on the Simulink Support Package for Arduino Hardware is now improved with a faster serial communication protocol. The new protocol reduces data drop during data logging. With this change, increasing the baud rate also increases the data logging performance.

Simulink Support Package for Samsung GALAXY Android Devices renamed to Simulink Support Package for Android Devices

The Simulink Support Package for Samsung GALAXY® Android Devices has been renamed to the Simulink Support Package for Android Devices.

Google Nexus Support: Run Simulink models on Google Nexus Android devices

The Simulink Support Package for Android Devices now supports Google® Nexus Android devices.

Watch this video to learn more.

Block Enhancements

State Reader and Writer Blocks: Reset and record states during model execution

Use the State Writer block with an Initialize Function block to set the state of a block in response to an initialize event. Use the State Reader block with a Terminate Function block to read the state of a block in response to a terminate event.

Blocks with state include:

- Discrete State-Space
- Discrete-Time Integrator
- Discrete Transfer Fcn
- Discrete Filter
- Delay
- Unit Delay
- S-Function (with one data type work vector declared as a discrete-state vector)

For more information, see Initialize Function, Terminate Function, Event Listener, State Reader, and State Writer block reference.

MATLAB System Block Support for Global Data: Access Simulink data stores from System objects using global variables

The MATLAB System block now has ability to share data between multiple MATLAB System blocks. For more information, see Data Sharing with the MATLAB System Block.

MATLAB System block now supports enumerated data types

The MATLAB System block now supports enumerated data types.

MATLAB System Block Support for LAPACK: Generate faster standalone code for linear algebra in a MATLAB System block

For improved simulation speed of certain linear algebra function calls in a System object associated with a MATLAB System block, the simulation software can call LAPACK functions. In R2016b, if you use Simulink Coder to generate C/C++ code from your model, you can specify that the code generator produce LAPACK function calls. If you specify that you want to generate LAPACK function calls, and the input arrays for the linear algebra functions meet certain criteria, the code generator produces calls to relevant LAPACK functions. The code generator uses the LAPACKE C interface.

LAPACK is a software library for numerical linear algebra. MATLAB uses this library in some linear algebra functions, such as `eig` and `svd`. Simulink uses the LAPACK library that is included with MATLAB. Simulink Coder uses the LAPACK library that you specify. If you do not specify a LAPACK library, the code generator produces code for the linear algebra function instead of generating a LAPACK call.

To generate LAPACK function calls and link to a specific LAPACK library, follow the procedure in Speed Up Linear Algebra in Code Generated from a MATLAB Function Block in the Simulink Coder documentation.

In R2016a, when you configured your model to generate LAPACK calls for standalone code, the configuration applied to MATLAB Function blocks and Stateflow charts. In R2016b, the configuration also applies to MATLAB System blocks. If you previously configured your model to generate LAPACK calls for MATLAB Function blocks or Stateflow charts, the code generator now also generates LAPACK calls for qualifying linear algebra calls in MATLAB System blocks.

Simpler way to call System objects

Instead of using the `step` method to perform the operation defined by a System object, you can call the object with arguments, as if it were a function. The `step` method will continue to work. This feature improves the readability of scripts and functions that use many different System objects.

For example, if you create a `dsp.FFT` System object named `fft1024`, then you call the System object as a function with that name.

```
fft1024 = dsp.FFT('FFTLengthSource','Property', ...
    'FFTLength',1024);
fft1024(x)
```

The equivalent operation using the `step` method is:

```
fft1024 = dsp.FFT('FFTLengthSource','Property', ...
    'FFTLength',1024);
step(fft1024,x)
```

When the `step` method has the System object as its only argument, the function equivalent has no arguments. This function but must be called with empty parentheses. For example, `step(sysobj)` and `sysobj()` perform equivalent operations.

System objects support for additional inputs, global variables, and enumeration data types

- System objects in code generated using MATLAB Coder can have up to 1024 inputs.
- You can use global variables declared in System objects to exchange data with the Data Store Memory block in Simulink. You can use these variables in generated code.
- Enumeration data types for System objects included in Simulink using the MATLAB System block is supported. Enumerations restrict data to a finite set of data values that inherit from `int8`, `uint8`, `int16`, `uint16`, `int32`, or Simulink.IntEnumType data types, or a data type you define using `Simulink.defineIntEnumType`.

Prelookup and Interpolation Using Prelookup Block Bus Support: Simplify and extend use of index and fraction signals

The Prelookup and the Interpolation Using Prelookup blocks use two signals as output and input, respectively:

- An index of the breakpoint set element

- An interval fraction that represents the normalized position on the breakpoint interval between the index and the next index value if the input is in range.

In R2016b, you can use a bus to combine index and fraction signals. Benefits of using a bus for these signals include:

- Simplifies the model by tying these two related signals together.
- Creates a testpoint `DpResult` structure for the AUTOSAR 4.0 library.
- For the AUTOSAR 4.0 library, avoids the creation of extra copies during code generation when the Prelookup and Interpolation Using Prelookup blocks are in separate models.

For more information, see the Prelookup and Interpolation Using Prelookup blocks.

From Spreadsheet block updates

The From Spreadsheet block now has a **Range** parameter. Use this parameter to select ranges of data in your spreadsheet.

Property inspector available for Simulink blocks

Property inspector view is now available for Simulink blocks. For more information on the property inspector, see Setting Properties and Parameters.

Manual Variant Source and Sink: Switch manually between different variants without using conditions

Use the Manual Variant Source and Sink blocks to switch between choices at input and output ports. You can double-click the block to toggle between the choices.

For more information, see Manual Variant Source and Manual Variant Sink.

Block Mask: Improved performance while evaluating mask parameter in fast restart mode

Once a model is initialized, only the tunable mask parameters are evaluated thus resulting in improved performance while evaluating the mask parameters.

Slider Range Parameter: Dynamically change the range of slider and dial parameter

You can change the range of the slider and dial parameters on a mask using callback code.

For more information, see the Change Slider range Dynamically model in `slexMaskParameterOptionsExample`.

Some types of unit delay blocks obsoleted

These unit delay blocks were removed from the Discrete library in R2016b. In new models, use the Delay block (with parameters set appropriately). For example, you can use the Resettable Delay block

to replace the Unit Delay Resettable External IC block. The Resettable Delay is the Delay block configured to reproduce the behavior of the Unit Delay Resettable External IC block.

Existing models that contain these blocks continue to work for backward compatibility.

- Unit Delay Enabled
- Unit Delay Enabled Resettable
- Unit Delay Enabled External IC
- Unit Delay Enabled Resettable External IC
- Unit Delay External IC
- Unit Delay Resettable
- Unit Delay Resettable External IC
- Unit Delay With Preview Enabled
- Unit Delay With Preview Enabled Resettable
- Unit Delay With Preview Enabled Resettable External RV
- Unit Delay With Preview Resettable
- Unit Delay With Preview Resettable External RV

Enhanced discrete block behavior

Previously, if your model used a variable-step solver, Simulink reported an error when it was unable to assign an appropriate sample time to the discrete blocks.

In this release, this discrepancy has been addressed so that discrete blocks always receive an appropriate sample time. The error message no longer appears.

In addition, for the following blocks, the default sample time setting has changed from 1 to -1:

- Discrete-Time Integrator
- Zero-Order Hold
- Discrete Zero-Pole
- Discrete PID Controller
- Discrete PID Controller (2DOF)
- Discrete State-Space

MATLAB Function Blocks

MATLAB Language Support: Use recursive functions and anonymous functions in a MATLAB Function block

Recursive functions

In R2016b, you can use recursive functions in MATLAB code that is intended for code generation. To generate code for recursive functions, MATLAB Coder uses compile-time recursion or run-time recursion. With compile-time recursion, the code generator creates multiple copies of the function in the generated code. The inputs to the copies have different sizes or constant values. With run-time recursion, the code generator produces recursive functions in the generated code. You can influence whether the code generator uses compile-time or run-time recursion by modifying your MATLAB code. You can disallow recursion or disable run-time recursion by modifying configuration parameters. See [Code Generation for Recursive Functions](#).

The block-level function in a MATLAB Function block cannot be a recursive function, but it can call a recursive function.

Anonymous functions

In R2016b, you can use anonymous functions in a MATLAB Function block. For example, a MATLAB Function block can contain the following MATLAB code that defines an anonymous function that finds the square of a number:

```
sqr = @(x) x.^2;  
a = sqr(5);
```

Anonymous functions are useful for creating a function handle to pass to a MATLAB function that evaluates an expression over a range of values. For example, this MATLAB code uses an anonymous function to create the input to the `fzero` function:

```
b = 2;  
c = 3.5;  
x = fzero(@(x) x^3 + b*x + c,0);
```

You cannot use anonymous functions for Simulink signals, parameters, or data store memory. For additional limitations, see [Code Generation for Anonymous Functions](#).

Variable-Size Cell Array Support: Use cell to create a variable-size cell array in a MATLAB Function block

To create a variable-size cell array in a MATLAB Function block, you can use the `cell` function. For example:

```
function z = mycell(n, j)  
assert(n < 100);  
x = cell(1,n);  
for i = 1:n  
    x{i} = i;  
end  
z = x{j};  
end
```


See Definition of Variable-Size Cell Array by Using cell.

Error for testing equality between enumeration and character array in a MATLAB Function block

For MATLAB code in a MATLAB Function block, an enumeration class must derive from a built-in numerical class. In R2016b, MATLAB introduces a new behavior for testing equality between these enumerations and a character array or cell array of character arrays. In previous releases, MATLAB compared the enumeration and character array character-wise. The behavior in a MATLAB Function block matched the MATLAB behavior. In R2016b, MATLAB compares the enumeration name with the character array. In R2016b, if a MATLAB Function block tests the equality of an enumeration and a character array, simulation or code generation ends with this error message:

Code generation does not support comparing an enumeration to a character array or cell array with the operators '==' and '~='

Consider this enumeration class:

```
classdef myColors < int8
    enumeration
        RED(1),
        GREEN(2)
    end
end
```

The following code compares an enumeration with the character vector 'RED':

```
mode = myColors.RED;
z = (mode == 'RED');
```

In previous releases, the answer in MATLAB and generated code was:

```
0 0 0
```

In R2016b, the answer in MATLAB is:

```
1
```

In R2016b, simulation or code generation ends with an error.

Version History

If you want the behavior of previous releases, cast the character array to a built-in numeric class. For example, use the built-in class from which the enumeration derives.

```
mode = myColors.RED;
z = (mode == int8('RED'));
```

Incremental build for relocation of MATLAB program files on the MATLAB path

In previous releases, for MATLAB files called by a MATLAB Function block, incremental builds treated a relocated file on the MATLAB path as an updated file, even if the contents or timestamp did not change. The relocation of an unchanged file called by a MATLAB Function block caused a rebuild.

In R2016b, incremental builds treat unchanged, relocated MATLAB files on the MATLAB path as unchanged files. Relocation of an unchanged MATLAB file called by a MATLAB Function block does not cause a rebuild.

To specify incremental builds, set **Simulation target build mode** to `Incremental build`.

Additional I/O Support: Generate code for fseek, ftell, fwrite

- `fseek`
- `ftell`
- `fwrite`

See [Data and File Management in MATLAB in Functions and Objects Supported for C/C++ Code Generation — Category List](#).

Code generation for additional MATLAB functions

- `cplxpair`
- `fminbnd`
- `inpolygon`
- `isenum`
- `polyeig`
- `repelem`

See [Functions and Objects Supported for C/C++ Code Generation — Alphabetical List](#).

Code generation for additional Audio Toolbox functions

- `integratedLoudness`
- `octaveFilter`
- `weightingFilter`

See [Audio System Toolbox in Functions and Objects Supported for C/C++ Code Generation — Category List](#).

Code generation for additional Computer Vision System Toolbox functions

- `cameraPoseToExtrinsics`
- `extrinsicsToCameraPose`

See [Computer Vision System Toolbox in Functions and Objects Supported for C/C++ Code Generation — Category List](#).

Statistics and Machine Learning Toolbox Code Generation: Generate code for prediction by using SVM and logistic regression models

You can generate C code that classifies new observations by using trained, binary support vector machine (SVM) or logistic regression models, or multiclass SVM or logistic regression via error-correcting output codes (ECOC).

- `saveCompactModel` compacts and saves the trained model to disk.
- `loadCompactModel` loads the compact model in a prediction function that you declare. The prediction function can, for example, accept new observations and return labels and scores.
- `predict` classifies and estimates scores for the new observations in the prediction function.
 - To classify by using binary SVM models, see `predict`.
 - To classify by using binary logistic regression models, see `predict`.
 - To classify by using multiclass SVM or logistic regression via ECOC, see `predict`.

Communications and DSP Code Generation: Generate code for additional functions

Communications System Toolbox

- `iqimbal`
- `comm.BasebandFileReader`
- `comm.BasebandFileWriter`
- `comm.EyeDiagram`
- `comm.PreambleDetector`

See Communications System Toolbox in Functions and Objects Supported for C/C++ Code Generation — Category List.

DSP System Toolbox

- `dsp.MovingAverage`
- `dsp.MovingMaximum`
- `dsp.MovingMinimum`
- `dsp.MovingRMS`
- `dsp.MovingStandardDeviation`
- `dsp.MovingVariance`
- `dsp.MedianFilter`
- `dsp.BinaryFileReader`
- `dsp.BinaryFileWriter`
- `dsp.Channelizer`
- `dsp.ChannelSynthesizer`

See DSP System Toolbox in Functions and Objects Supported for C/C++ Code Generation — Category List.

Phased Array System Toolbox

- `musicdoa`
- `pambgfun`
- `taylortaperc`
- `phased.GSCBeamformer`
- `phased.WidebandBackscatterRadarTarget`
- `phased.WidebandTwoRayChannel`
- `phased.MUSICEstimator`
- `phased.MUSICEstimator2D`

See Phased Array System Toolbox in Functions and Objects Supported for C/C++ Code Generation — Category List.

Conditional breakpoints for run-time debugging

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

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

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

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

Compiler optimization parameter support for faster simulation

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

Run-Time error stack in Diagnostic Viewer

In the Diagnostic Viewer, some error messages display the run-time stack information and include the line numbers for the MATLAB code. You can trace these errors that occur in MATLAB Function blocks to the line of code. When you select the line number, the link opens the editor and highlights the corresponding line of code. Previously, the error message did not provide information about the location of the error.

Modeling Guidelines

Modeling guidelines for high-integrity systems

The following are new modeling guidelines to develop models and generate code for high-integrity systems:

- hisl_0033: Usage of Lookup Table blocks
- hisl_0034: Usage of Signal Routing blocks
- hisl_0036: Configuration Parameters > Diagnostics > Saving
- hisl_0037: Configuration Parameters > Model Referencing
- hisl_0038: Configuration Parameters > Code Generation > Comments
- hisl_0039: Configuration Parameters > Code Generation > Interface
- hisl_0047: Configuration Parameters > Code Generation > Code Style
- hisl_0049: Configuration Parameters > Code Generation > Symbols

R2016a

Version: 8.7

New Features

Bug Fixes

Version History

Simulation Analysis and Performance

Automatic Solver Option: Set up and simulate your model more quickly with automatically selected solver settings

Starting with R2015b, you can use the auto solver to select a solver and step size for simulating a model. The auto solver suggests a fixed-step or variable-step solver along with maximum step size based on the dynamics of the model. Select the auto solver in the solver pane and accept recommended settings in the solver information dialog box. For more information, see [Use Auto Solver to Select a Solver](#).

Starting with R2016a, auto solver calculates the stiffness of a model. For stiff models, auto solver selects ode15s. For more information, see [Auto Solver Heuristics](#).

One-Click Display: Click a signal line when the simulation is running to view the current value

When you simulate your model, you can now display the port value label for a signal by clicking it. This option is selected by default.

For more information, see [Display Value for a Specific Port](#) or watch this video to learn more.

Simulation Metadata Diagnostics: Understand why a simulation has stopped in batch or individual runs

You can now use diagnostics in `SimulationMetadata` to understand why a simulation stopped. The metadata object has a new property called `ExecutionInfo`. This property contains information about stop events, error, and warning diagnostics. You can use the information in `ExecutionInfo` to troubleshoot individual runs or each run in a batch simulation.

See `Simulink.SimulationMetadata` to learn more.

Multi-Input Root Inport Mapping: Connect multiple sets of input signals to your Simulink model for interactive or batch simulation

The Root Inport Mapping tool now supports the connection of multiple sets of input signals to your Simulink model for interactive or batch simulation.

For more information, see [Map Root Inport Signal Data](#) or watch this video to learn more.

Simulation for Mixed Targets: Simulate system-level designs that integrate referenced models targeting an assembly of heterogeneous embedded devices

You can simulate a parent system model that includes referenced models configured for heterogeneous embedded devices. For example, you can simulate a parent model that has a referenced model **Configuration Parameters > Hardware Implementation > Device vendor** parameter set to `ARM Compatible` and another referenced model with that parameter set to `ASIC/`

FPGA. Simulink models can now simulate system engineering level models that target multiple hardware devices.

Time Out feature for Performance Advisor run time

You can now specify a run-time duration for Performance Advisor by selecting the **Time Out** option. For models with long simulation times, use this option to limit the time period for Performance Advisor runs. For more information, see Specify Runtime for Performance Advisor.

Solver Profiler to speed up simulation performance

You can use Solver Profiler to examine solver behavior of variable step solvers and model behavior to identify issues that can contribute to poor simulation performance. Run Solver Profiler on models that fail to simulate or that have long simulation times. When you pause or stop the simulation, Solver Profiler displays information gathered during runtime including an analysis of conditions that can slow down simulation. Use this data to examine the model simulation for solver resets, zero crossings and solver exceptions.

Solver Profiler highlights states in the model that contain solver errors. You can also launch States Explorer from Profiler to further investigate the state plot of a model state.

For more information, see Examine Model Dynamics Using Solver Profiler.

Diagnostic Viewer performance improvement

The Diagnostic Viewer performs better and faster when handling large number of similar warnings from Simulink operations.

Component-Based Modeling

Variant Source and Sink Blocks with Condition Propagation: Design variant choices and automatically remove unneeded functionality based on block connectivity

Simulink provides two blocks to visualize implementations of variant choices in a model graphically—Variant Source and Variant Sink.

When you compile the model, Simulink determines which variant control evaluates to `true`. Simulink then deactivates blocks that are not tied to the variant control being true and visualizes the active connections.

When you specify variant conditions in models containing Variant Source and Variant Sink blocks, Simulink propagates these conditions backward and forward from these blocks to determine which components of the model are active during simulation.

See Variant Condition Propagation with Variant Sources and Sinks or watch this video to learn more.

Scoping Simulink Functions: Call Simulink Function blocks within a subsystem hierarchy

Defining the scope of Simulink functions in a Simulink model allows you to modularize your model by limiting the visibility and access of functions. Simulink uses subsystems to define the local scope of a function. See Define Scope of Simulink Function Blocks and Simulink Functions in Simulink Models, or watch this video to learn more.

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

Simulink now supports the specification of physical units as part of signal attributes. This capability enables you to specify unit attributes at the boundaries of components such as subsystems and models. To learn more, see Unit Specification in Simulink Models or watch this video.

For a list of supported unit systems and their units, see Allowed Unit Systems.

Units in Simulink.Parameter and Simulink.Signal Objects

The `DocUnits` property is now `Units`. Use this property to specify units for signals.

Note You can still continue to use the `DocUnits` field to access or set the property. This capability maintains backward compatibility for existing MATLAB code, MAT-files, and Simulink data dictionaries that use the `DocUnits` field.

Specifying units in MATLAB Function blocks

Simulink supports the specification of a unit property for data inputs and outputs of MATLAB Function blocks. Specify units for input and output data by using the **Unit (e.g., m, m/s², N*m)** parameter on the Ports and Data Manager.

During model update, Simulink checks for inconsistencies in units between input and output data ports and the corresponding Simulink signals.

Units for logging and loading signal data

You can include units in signal data that you log or load.

Units for logging and loading are specified using `Simulink.SimulationData.Unit` objects. When you log using `Dataset` or `Timeseries` format, Simulink stores the unit information using `Simulink.SimulationData.Unit` objects. If you create MATLAB timeseries data to load, you can specify `Simulink.SimulationData.Unit` object for the `Units` property of the `timeseries` object.

For details, see [Log Signal Data That Uses Units](#) and [Load Signal Data That Uses Units](#).

New units blocks

The following blocks are new:

- Unit Conversion
- Unit System Configuration

Configuration parameters

The following configuration parameters are new, available on the **All Parameters** tab:

- Allowed unit systems
- Unitsinconsistency messages
- Allow automatic unit conversions

Model Advisor checks

The following Model Advisor checks are new for units:

- Identify undefined units in the model
- Identify unit mismatches in the model
- Identify disallowed unit systems in the model
- Identify automatic unit conversions in the model

Updated example

The [Exploring the Solver Jacobian Structure of a Model](#) has been updated with unit specifications in all components.

Version History

- The `DocUnits` property is now `Unit` for `Simulink.Parameter` or `Simulink.Signal` objects. If, in a previous release, you used the `DocUnits` parameter of a `Simulink.Parameter` or `Simulink.Signal` object to contain text that does not now comply with units specifications, simulation returns a warning when the model simulates.

To suppress these warnings, set the configuration parameter `Unitsinconsistency messages` to `none`. This setting suppresses all units inconsistency check warnings.

- If you have a class that derives from `Simulink.Parameter`, `Simulink.Signal`, or `Simulink.BusElement` with a previously defined `Unit` property, Simulink returns an error like the following:

```
Cannot define property 'Unit' in class 'classname' because the property
has already been defined in the superclass 'superclass'.
```

If you use this property to represent the physical unit of the signal, delete the `Unit` property from the derived class in the R2016a or later release. Existing scripts continue to work, unless you are assigning incorrectly formed unit expressions to the `Unit` field. In this case, replace the use of `Unit` with `DocUnits` to continue to be able to assign the unit expression.

Note If you store existing data in a MAT- or .sldd file, in a release prior to R2016a, copy the contents of the `Unit` property to the `DocUnits` first. Then, save the file in the earlier release before loading the model in R2016a or later release.

Mask Dialogs: Create masks with flexible layout options and new control parameters

You can use these controls from the Mask Editor dialog box to create a rich mask dialog box:

- Collapsible panel: Expand or collapse sections in the mask dialog box.
- Dynamic mask dialogs: Change the menu options of a dependent dialog box based on the value of the controlling parameter.
- Spinbox step size: Fine-tune and control a parameter value in the mask dialog box.
- Horizontal stretch property: Create a more flexible mask dialog box layout.
- Slider and dial parameter: Use variables to specify value in dial and slider parameters.

For more information, see Mask Editor Overview.

Mask Images: Quickly add images to masks and while keeping the port names visible

- Mask icons can leave port names visible.
- Mask icon preview is available in the Mask Editor.

For more information, see Mask Editor Overview.

Tracing Simulink Functions: Display connections between all Function Callers and a Simulink Function

Visually display connections between Simulink functions and their callers with tracing lines that connect callers to functions:

- Turning on/off tracing lines — From the **Display** menu, select **Function Connectors**.
- Direction of tracing lines — Lines connected at the bottom of a block are from a function caller. Lines connected at the top of a block are to a Simulink function or a subsystem containing the function.

- Navigation to functions — If a function caller is within a subsystem, you can open the subsystem, and then click a link to the called Simulink function. If the function is at the root level of a model, the function opens. If the function is within a subsystem, the subsystem containing the function opens.

Signal Label Propagation for Referenced Models: Propagate signal labels out of referenced models by default

To propagate signal labels out of referenced models more easily, in R2016a for new models, the **Configuration Parameters > Data Model Referencing > Propagate all signal labels out of the model** parameter is now enabled by default. The default for the `PropagateSignalLabelsOutOfModel` parameter is now 'on'. For details, see Propagate all signal labels out of the model.

Version History

If you open a model created before R2016a that has the **Propagate all signal labels out of the model** parameter cleared, that setting is preserved.

The new default can require updates to scripts written before R2016a that rely on the previous default `PropagateSignalLabelsOutOfModel` setting of 'off'. To have the scripts work as expected with models created in R2016a or later, update the code to reflect that the default for that parameter is now 'on'.

Simulink.SubSystem.convertToModelReference function for multiple subsystem conversion: Convert multiple subsystems with one command

You can convert multiple subsystems in a model to referenced models in one `Simulink.SubSystem.convertToModelReference` command.

Subsystem to Model Reference Conversion: Insert subsystem wrapper to preserve model layout

When you convert a subsystem to a referenced model, you can have the conversion process insert a wrapper subsystem to preserve the layout of a model. The subsystem wrapper contains the Model block from the conversion.

In the Model Reference Conversion Advisor, enable the **Create a wrapper subsystem** input parameter. In the `Simulink.SubSystem.convertToModelReference` function, use the `CreateWrapperSubsystem` name and value pair.

Model Reference Conversion Automatic Fix for Goto Blocks: Convert subsystems with Goto blocks more easily

The Model Reference Conversion Advisor and the `Simulink.SubSystem.convertToModelReference` function include additional checks and automatic fixes to make it easier to convert subsystems with Goto blocks to referenced models.

Virtual Bus Signals Across Model Reference Boundaries: Use virtual bus signals as inputs or outputs of a referenced model

In R2016a, when you specify that a bus signal input or output for a referenced model is a virtual bus, Simulink sets up the model so that, compared to previous releases, generated code generally:

- Has fewer copies of bus signals
- Executes faster

Use the Block Parameters dialog box for an Inport or Outport block to specify virtual bus output.

- Inport block — Clear the **Output as nonvirtual bus** parameter.
- Outport block — Clear the **Output as nonvirtual bus in parent model** parameter.

For information about the changes to the code that you generate from models, see Model Block Virtual Buses: Interface to Model blocks by using virtual buses, reducing data copies in the generated code.

Version History

The behavior of models that meet these criteria is different than it is in R2016a:

- The model was saved in a release earlier than R2016a.
- The model has referenced models with bus inputs and outputs configured to be treated as virtual buses.

Use the Upgrade Advisor **Check for virtual bus across model reference boundaries** check to avoid errors that the new behavior can trigger. Run the **Analyze model hierarchy and continue upgrade sequence** check on the top-level model and then down through the model reference hierarchy.

Bus Selector and Bus Assignment Block Signals: Display full signal path while editing a model

You can now display full paths to bus signals for Bus Selector and Bus Assignment blocks in the Simulink Editor, without opening the Block Parameters dialog boxes for the blocks. Interactively viewing the full signal path can simplify the process of editing signals with duplicate leaf names, by eliminating the need to match the block ports with the associated signals. Also, interactively viewing the full path for the signal provides a quick way to get context for understanding the model.

- To see the full path of a Bus Selector block output signal, hover over the output signal label.
- To see the full path of up to the first ten output signals of a Bus Selector block, hover over the block.
- To see the full path of up to the first ten signals in a Bus Assignment block, hover over the block.

Multi-Input Bus-Capable Block Ports: Simulate unconnected multi-input bus-capable block ports without error

Before R2016a, simulating a model that contains a multiport bus-capable block with an unconnected input port causes an error if one or more of the input ports is connected directly to *either*:

- A bus signal
- A Ground block

In R2016a, you can simulate under the same circumstances without error for these multiport bus-capable blocks:

- Manual Switch
- Multiport Switch
- S-Function
- Switch
- Variant Source
- Vector Concatenate

Using a Merge block in Simplified initialization mode in still causes an error for those situations.

Output Blocks with Bus Output: Simulate Output blocks with a bus output without error

Before R2016a, simulating a model that contains an Output block whose output data type is specified by a bus object caused an error if *either* of these conditions applied:

- The Output block has an unconnected port.
- The Output block is connected directly to a Ground block.

In R2016a, you can simulate without error in these situations.

Function-Call Split block with multiple outputs

Before R2016a, the Function-Call Split block was a circular block with one input and two outputs. In R2016a, this block can split an incoming function-call signal into more than two output signals. You can also change the shape of the block.

Function-Call Split block with no input signal

Before R2016a, the Function-Call Split block had to have an input signal. In R2016a, you can simulate your model without supplying an input signal to the Function-Call Split block.

Trigger port with inherited periodic function-call signal

Before R2016a, if the Trigger block of a triggered subsystem had the **Trigger type** parameter set to function-call and **Sample time type** set to periodic, you could not set **Sample time** to -1. In R2016a, you can specify a sample time of -1.

Standalone code generation for models with asynchronous function-call inputs

Before R2016a, you had to build a top model from which to reference asynchronous function-call inputs. In R2016a, models with asynchronous function-call inputs support code generation on standalone models. You can now build these models as is.

Additional component parameters saved with Simulink.ConfigSet.saveAs

In R2016a, the `Simulink.ConfigSet.saveAs` function saves all enabled parameters in the base configuration set. In addition, the function saves enabled parameters for the following components:

- Coder Target
- HDL Coder
- Polyspace
- SimEvents
- Simscape
- Simulink Coverage
- Simulink Design Verifier
- Simulink PLC Coder™
- Target Hardware Resources

Project and File Management

Start Page: Get started or resume work faster by accessing templates, recent models, and featured examples



The Simulink start page helps you get started faster by offering model and project templates and examples. You can use common design patterns or learn about new features.

If you want to pick up where you left off, the start page shows a list of recent models, and opens the project automatically if your selected model is part of a project. You can now also open recent models from the Simulink Editor or from the Library Browser.

For details, see [Create Models and Open Existing Models](#), or watch this video to learn more.

Version History

In previous releases, you could not access a list of recent models, and you could not find model and project templates and examples in the same place. The Simulink Template Gallery and the project template browser are merged into the start page.

Functionality	What Happens When You Use This Functionality?	Compatibility Considerations
simulink	Opens the start page	simulink no longer opens the Library Browser, but instead opens the start page. To open the Library Browser, use <code>slLibraryBrowser</code> instead, or click the Library Browser button in the Editor: 
 Simulink button on the MATLAB Home tab.	Opens the start page	The Simulink button in MATLAB no longer opens the Library Browser, but instead opens the start page.

Functionality	What Happens When You Use This Functionality?	Compatibility Considerations
Creating a new model, chart, library, or project from the MATLAB Home tab New menu, from the Editor, from the Library Browser, or from Simulink Project.	Opens the start page	<p>Choose a template from the start page. The start page shows an appropriate filtered list of models, charts, libraries, or projects.</p> <p>To recreate the previous workflow:</p> <ul style="list-style-type: none"> • In the start page, select Blank Model, Blank Library, Blank Project, or Chart, (or press Ctrl+N for a blank model), • In the Editor or Library Browser, select a new Blank Model (or press Ctrl+N). <p>To create a project using source control or from an archive, use the start page options.</p>
Use model templates.	Find all your templates on the start page.	The menu item From Template is no longer in the Editor or Library Browser. Instead, in the Simulink Editor, select File > New > Model and select your template in the start page. In the Library Browser, click the New Model button arrow and select Model . The Simulink Template Gallery is now merged into the start page.
Use project templates.	Find all your templates on the start page.	Project templates are no longer in the MATLAB Home tab New menu or in the Create Project dialog box. Instead, open the start page from MATLAB, or from the Editor by selecting File > New > Project , and select your template in the start page.
Simulink project templates created in R2014a or earlier (.zip files)	You cannot browse to legacy templates in the start page	Upgrade legacy templates to .sltx files using <code>Simulink.exportToTemplate</code> .

Automatic Renaming: Update all references in a project when you rename models, libraries, or MATLAB files

In a Simulink project, when you rename, delete, or remove a file, the project checks for impact in other project files. You can find and fix impacts such as changed library links, model references, and model callbacks. This tooling can help you avoid refactoring pain tracking down other affected files. Automatic renaming is helpful in preventing errors that result if you change names or paths manually and overlook or mistype one or more instances of the name.

Automatic renaming helps you refactor MATLAB code. Simulink project dependency analysis now finds dependencies on MATLAB code in packaged functions, classes, and superclasses. You can view the dependencies in the Impact graph and if you refactor the files, automatic renaming prompts you. For example, when renaming a class, the project offers to automatically update all classes that inherit from it. If you rename a `.m` or `.mlx` file, the project offers to automatically update any files and callbacks that call it.

For details, see [Automatic Updates When Renaming, Deleting, or Removing Files](#).

Three-Way Model Merge: Resolve conflicts between revisions and ancestor models using Simulink projects

In a Simulink project under source control, if your changes in a model conflict with another user, you can open the Three-Way Model Merge tool to resolve the conflicts. You can examine your local file compared to the conflicting revision and the base ancestor file, and decide which changes to keep. You can resolve the conflict and submit your changes. Three-Way Model Merge requires Simulink Report Generator™.

For details, see [Resolve Conflicts](#).

Template API: Programmatically create models and projects from custom templates

In R2016a, you can programmatically create models and projects from templates, and create custom templates. Model and project templates are starting points to apply common modeling approaches. They help you reuse settings and block configurations and share knowledge. Use model and project templates to apply best practices and take advantage of previous modeling solutions.

To use templates programmatically, see `Simulink.createFromTemplate`, `Simulink.findTemplates`, and `Simulink.exportToTemplate`.

Export function: Export to previous version using Simulink.exportToVersion

In R2016a you can use a new function, `Simulink.exportToVersion`, to export a model so that you can open it in a previous version of Simulink. The new function makes it easier to distinguish between exporting to a previous version and saving with a different name.

In previous releases you could use `save_system` with the `'ExportToVersion'` option. This option will also continue to work.

Dirty Model Management: Identify, save, or discard unsaved changes in project models

In a Simulink project, you can check for unsaved changes in project models, and decide whether to save or discard changes. In the previous release, you had to close the project to see warnings about unsaved changes. Now you can manage unsaved changes before closing the project.

For details, see [Manage Shadowed and Dirty Model Files](#).

Source Control API: Programmatically get modified files and revision information

In R2016a, you can programmatically get source control status information about Simulink project files. Use `simulinkproject` to get a project object, then you can get a list of modified files in the project. You can get the local status of a file (modified, unmodified, not under source control) and the revision if available.

For details, see `listModifiedFiles`.

Source Control Notifications: List changed files on update (SVN); find out if your branch is behind the origin (Git)

In R2016a, Simulink Project provides more notifications to help you with source control operations. With SVN, when you update a project from the repository, you see a list of all changed files in a dialog box. With Git, when you fetch from a remote repository, you can see if your current branch is behind or ahead of the remote tracking branch (the origin).

For details, see [Update Revisions with SVN and Fetch and Merge](#).

SVN Externals: Include files in projects from other repositories or repository locations

Use SVN externals to get files into your Simulink project from another repository or from a different part of the same repository. Right-click a project folder and select **Manage SVN Externals**. The project provides a dialog to help you browse, specify and validate the externals definitions. After you define the externals setting on a folder, other project users can get the same files included in their project.

For details, see [Manage SVN Externals](#).

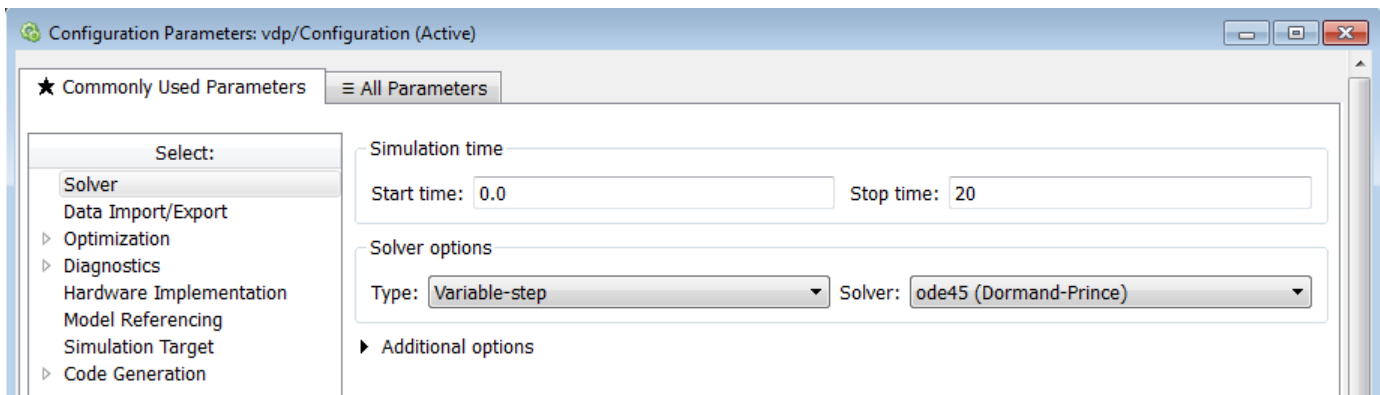
Custom Shortcut Icons: Personalize frequent task buttons on the toolstrip

In R2016a, you can specify an icon to use for your shortcut buttons on the Project Shortcuts tab. Icons such as “build” can aid other project users to recognize frequent tasks. For details, see [Customize Shortcut Icons](#).

Simplified Configuration Parameters: Configure model more easily using streamlined category panes

In the Configuration Parameters dialog box, streamlined category panes display only configuration parameters that you are most likely to use when configuring your model.

The category panes, previously referred to as the Category view, are now available on the **Commonly Used Parameters** tab. The **All Parameters** tab, previously referred to as the List view, provides the complete list of parameters in the model configuration set.



Version History

Following are the configuration parameters that have moved to the **All Parameters** tab or moved to a different pane.

Note Parameters that are removed from a pane are still available for configuration on the **All Parameters** tab. To locate a parameter on this tab, use either the search box or the **Category** filter.

Data Import/Export Pane

The **Enable live streaming of selected signal to Simulation Data Inspector** parameter is moved to the **All Parameters** tab.

The **Save simulation output as single object** parameter is now called **Single simulation output**

The following parameters are available by clicking **Additional Parameters** at the bottom of the pane:

- **Limit data points to last**
- **Decimation**
- **Output options**
- **Refine factor**

Diagnostics Pane

The following parameter is moved to the **All Parameters** tab:

- **Solver data inconsistency**

Diagnostics > Data Validity Pane

The following parameters are moved to the **All Parameters** tab:

- **Array bounds exceeded**
- **Model verification block enabling**
- **Check preactivation output of execution context**
- **Check runtime output of execution context**
- **Check undefined subsystem initial output**
- **Detect multiple driving blocks executing at the same time step**
- **Underspecified initialization detection**

Diagnostics > Saving Pane

The pane is removed and its parameters are moved to the **All Parameters** tab:

- **Block diagram contains disabled library links**
- **Block diagram contains parameterized library links**

Diagnostics > Solver Pane

The following parameters are moved to the **Diagnostics > Sample Time** pane:

- **Sample hit time adjusting**
- **Unspecified inheritability of sample time**

The following parameter is moved to the **Diagnostics > Compatibility** pane:

- **SimState object from earlier release**

Optimization Pane

The following parameters are moved to the **All Parameters** tab:

- **Remove code from floating-point to integer conversions with saturation that maps NaN to zero**
- **Compiler optimization level**
- **Verbose accelerator builds**
- **Implement logic signals as Boolean data (vs. double)**
- **Block reduction**
- **Conditional input branch execution**
- **Use memset to initialize floats and doubles to 0.0**

Optimization > Signals and Parameters Pane

The following parameters are moved to the **All Parameters** tab:

- **Signal storage reuse**

- **Enable local block outputs**
- **Reuse local block outputs**
- **Optimize global data access**
- **Reuse global block outputs**
- **Eliminate superfluous local variables (Expression folding)**
- **Simplify array indexing**

Simulation Target Pane

The following parameters are moved to the **All Parameters** tab:

- **Echo expressions without semicolons**
- **Simulation target build mode**
- **Ensure responsiveness**
- **Generate typedefs for imported bus and enumeration types**
- **Ensure memory integrity**

Simulation Target > Custom Code Pane

The pane is removed and its parameters are moved to the **Simulation Target** pane:

- **Header file**
- **Initialize function**
- **Source file**
- **Terminate function**
- **Parse custom code symbols**
- **Include directories**
- **Libraries**
- **Source files**
- **Defines**

Simulation Target > Symbols Pane

The pane is removed and its parameter is moved to the **Simulation Target** pane:

- **Reserved names**

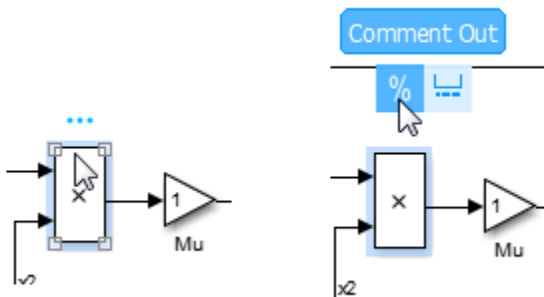
Simulink Editor

Single-Selection Actions: Access commonly used editing actions when clicking a block or signal line

When you select a block or a signal line in a Simulink model, a cue appears that lets you select a common action to perform. When you move your cursor over the cue, an action bar appears. Click the action you want to perform.

- For blocks, you can comment or uncomment the block or hide or display the block name using this cue.
- For signal lines, you can autoroute the line or enable or disable signal logging.

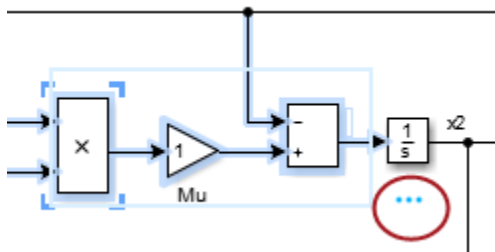
The figure shows how to use the cue to comment out a block.



Watch this video to learn more.

Multiple-Selection Cue: Selecting multiple blocks in the Simulink Editor shows new cue

When you select multiple blocks in your model, the Simulink Editor now displays a smaller cue, as shown in the figure. Move your mouse over the cue to display the action bar of common commands.



Single Click for Quick Insert: Click block name once to insert block from list

If you know the name of the block you want to add to a model, you can click in the model where you want to add the block and start typing the block name. The blocks that match appear in a list. In

previous releases, you double-clicked the block name to insert it in the model. In R2016a, you click the block name once to insert it. For an example, see [Add More Blocks](#).

Interactive Library Unlocking: Click lock symbol in custom libraries to unlock

Clicking the lock symbol in a custom library now unlocks the library so you can edit it. For information about library locks, see [Create a Custom Library](#).

Key Parameter Preference: Turn off parameter prompt during block creation. When you add a block to a model, a prompt appears so that you can enter a key parameter. To prevent this prompt from appearing, you can set a preference. In your model, select **File > Simulink Preferences**. In the Editor Preferences pane, clear the Edit key parameter when adding new blocks check box.

Improved block search usability

The Quick Block Insert search displays a description of the library making it easier to recognize the block in the Library Browser.

Data Management

Signal and State Logging to File: Log data directly to a MAT-file for long simulations

When logging large amounts of data that can cause memory issues for a long simulation, use the new **Configuration Parameters > Data Import/Export > Log Dataset data to file** parameter. With this feature enabled, Simulink logs data in Dataset format to a MAT-file for the following kinds of logging:

- Signal logging
- States
- Final states
- Output
- Data stores

Alternatively, you can enable this feature using the `LoggingToFile` and `LoggingFileName` name-value pairs with either the `sim` command or `set_param` command.

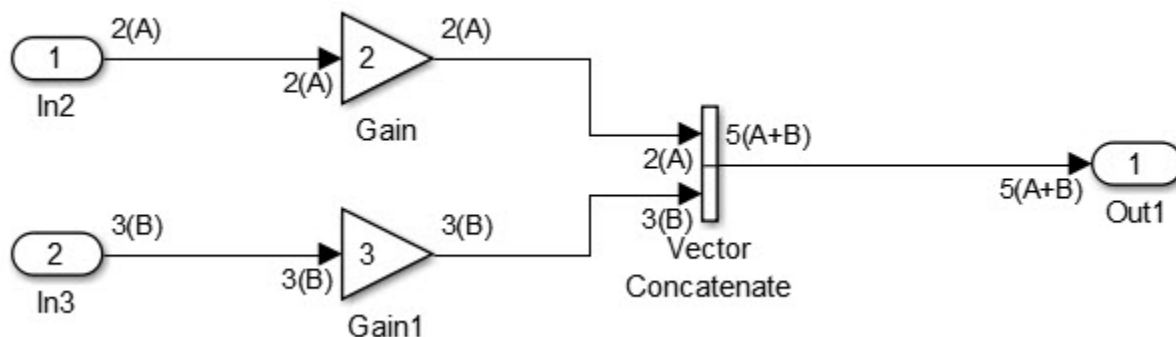
Using a `Simulink.SimulationData.DatasetRef` object to access signal logging and states logging data loads data into the model workspace incrementally. Loading and accessing data for other kinds of logging loads all of the data into memory at once.

For details, see [Log Data Using Persistent Storage](#).

Preserve symbolic constants in propagated signal dimensions

Previously, Simulink treated signal and parameter dimension specifications as numeric constants. In R2016a, you can use a `Simulink.Parameter` object as a symbol in a MATLAB expression to represent a dimension value. During simulation, Simulink propagates dimension symbols throughout the model and preserves these symbols in the propagated signal dimensions.

For example, Inport blocks `In1` and `In2` have symbolic constant dimension specifications. In the **Signals Attribute** tab in the Source Block Parameters dialog box for `In1`, the **Port dimension** parameter has the `Simulink.Parameter A`, which has a value of 2. In the **Signals Attribute** tab in the Source Block Parameters dialog box for `In2`, the **Port dimension** parameter has the `Simulink.Parameter B`, which has a value of 3. When you simulate the model, you see the symbolic constants and their values propagate throughout the model.



Embedded Coder preserves these dimension symbols and expressions in the generated code. If you use Embedded Coder, you can compile the same generated code into multiple applications of different sizes. For more information on generating code with symbolic constants, see [Compile-Time Dimensions: Generate compiler directives \(#define\) for implementing signal dimensions and Implement Dimension Variants for Array Sizes in Generated Code](#).

Dataset Format for Signal Logging: Log signals in format used for other logging

When you open a model in R2016a, signal logging uses the `Dataset` format for the logged data. The `SignalLoggingSaveFormat` parameter is set to `Dataset`.

`Dataset` is the default format for logging output and state information. Using the same format for signal logging data as well as other logged data simplifies writing scripts to process that data.

Version History

The `ModelDataLogs` class is supported for backwards compatibility. Starting in R2016a, you cannot log data in the `ModelDataLogs` format. In R2016a or later, when you open a model from an earlier release that had used `ModelDataLogs` format, the model is converted to use `Dataset` format.

You can convert signal logging data from `ModelDataLogs` to `Dataset` format. For more information, see [Convert Logged Data to Dataset Format](#).

If you have legacy code that uses the `Simulink.ModelDataLogs` API, you can encounter situations that require updates to your code or model. See [Migrate Scripts That Use ModelDataLogs API](#).

Unlimited Number of Data Points for Logging by Default: Log all data points by default

To simplify logging time, output, and state data, the default behavior is to have no limit on the number of data points logged during a simulation. In R2016a, by default the **Configuration Parameters > All Parameters > Limit data points** is now cleared.

Version History

The changed default can require updates to scripts written before R2016a that rely on the previous default of limiting the logged data points to 1000 or to a specified number of data points and that use:

- The `new_system` function to create an empty Simulink system
- `Simulink.ConfigSet` to access the model configuration set

Root Inport Mapping Tool Updates

The Root Inport Mapping Tool has the following updates:

- New interface
- Support for Excel spreadsheets on all platforms
- Support for the linking in of multiple scenario files at the same time

For more information, see [Map Root Inport Signal Data](#)

Function to convert MAT-file contents to Simulink.SimulationData.Dataset object

New `convertToSlDataset` function converts the contents of a MAT-file that contains Simulink inputs to a `Simulink.SimulationData.Dataset` object.

Functions to identify and close data dictionaries

Some commands and functions, such as `Simulink.data.dictionary.cleanupWorkerCache`, cannot operate when data dictionaries are open. Use the new function `Simulink.data.dictionary.getOpenDictionaryPaths` to identify open data dictionaries.

You can use the new function `Simulink.data.dictionary.closeAll` to close all connections to all open data dictionaries. However, it is a best practice to close each individual connection.

Navigation to variables from additional block dialog boxes

Beginning in R2015a, you can navigate to or create variables directly from blocks in the base Simulink libraries such as Discrete and Continuous. You can navigate to or create the variables in a workspace or data dictionary, or navigate to mask parameters, by right-clicking the variable name in the block dialog box. For more information, see [Manage Variables from Block Parameter](#).

In R2016a, the blocks in these additional libraries allow you to navigate to variables from block dialog boxes:

- **Communications System Toolbox**
- **Communications System Toolbox Support Package for RTL-SDR Radio**, except for the **Radio address** parameter in the RTL-SDR Receiver block
- **Communications System Toolbox Support Package for Xilinx FPGA-Based Radio**
- **SimEvents**, except for the **Port number** parameter in the Conn legacy block
- **Simulink Coder**
- **Simulink Control Design**
- **Simulink Design Optimization**
- **Simulink Design Verifier**
- **Simulink Extras**
- **Simulink Support Package for Apple iOS Devices**
- **Simulink Support Package for BeagleBoard Hardware**
- **Simulink Support Package for LEGO MINDSTORMS EV3 Hardware**
- **Simulink Support Package for LEGO MINDSTORMS NXT Hardware**
- **Simulink Support Package for Raspberry Pi Hardware**
- **Simulink Support Package for Samsung GALAXY Android Devices**
- **System Identification Toolbox**

Functionality Being Removed or Changed

Functionality	Result	Use Instead	Compatibility Considerations
<p>The programmatic parameter <code>DefaultDataPackage</code>, which corresponds to the Package option on the Data Management Defaults pane in the Simulink Preferences dialog box.</p>	<p>Warning. For example, this command generates a warning:</p> <pre>set_param(0, ... 'DefaultDataPackage', ... 'mpt')</pre>	<p>To select default classes for creating signal and parameter objects, on the Model Explorer toolbar, click the arrows next to the Simulink Signal and Simulink Parameter buttons.</p> <p>To select a default package when you apply a custom storage class in a Signal Properties dialog box or in a block dialog box, use the new drop-down list Signal object class.</p> <p>In the Data Object Wizard, click Change Class to select a class for each data object that the wizard proposes.</p>	<p>In your scripts, remove references to the programmatic parameter <code>DefaultDataPackage</code>.</p>

Connection to Hardware

Hardware implementation parameters enabled by default

In R2016a, the **Enable hardware specification** button is removed from the **Configuration Parameters > Hardware Implementation** pane. By default, the parameters on the pane are enabled.

Mac Support for LEGO EV3: Run Simulink models on LEGO EV3 hardware from a Mac

You can use the Simulink Support Package for LEGO MINDSTORMS EV3 Hardware on the Apple Mac OS X platform.

Block Enhancements

From Spreadsheet Block Updates

The From Spreadsheet has the following changes:

- The block has moved from the Simulink Extras/Additional Sources sublibrary to the Simulink Sources sublibrary.
- The block now supports Rapid Accelerator mode on all platforms.

System object enhancements to MATLAB System block

To implement these classes and methods for defining your own System objects, add them to your object's class definition file.

- Fixed-point data tab — The `showFiSettingsImpl` method adds a Data Types tab to the MATLAB System block dialog box. This tab includes options for fixed-point data settings.
- Model reference discrete sample time inheritance — The `allowModelReferenceDiscreteSampleTimeInheritanceImpl` method lets you specify whether a System object in a referenced model can inherit the sample time of the parent model. If your object uses discrete sample time in its algorithm, you set this method to `true` to allow inheritance.

Unit Delay block does not accept rate transitions

In R2016a, if the Unit Delay block input and output signals are at different rates, the block produces an error. For the rate transitions workflow, use a Rate Transition block.

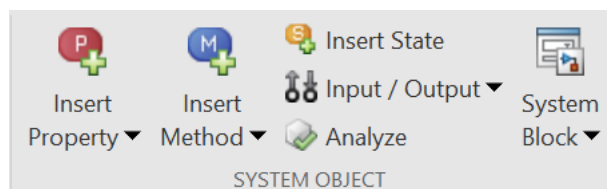
Matrix Interpolation Block for Multidimensional Lookup Table Data

The Matrix Interpolation block performs interpolation (or extrapolation) on a multidimensional table, where each data point can be a matrix. The block resides in the Simulink Extras library.

Enhanced System Object Development with MATLAB Editor

Create System objects in the MATLAB Editor using code insertion and visualization options.

- Define your System object with options to insert properties, methods, states, inputs, and outputs.
- View and navigate the System object code with the Analyzer.
- Develop System block and preview block dialog box interactively (with Simulink only).



These coding tools are available when you open an existing System object or create a new System object with **New > System object**.

Scope Block and Signal Viewer Enhancements

- Added ability to select signals for individual displays. This change removes an incapability from previous releases. See [Connect Signals to Floating Scope Block or Scope Viewer](#)
- Expand scope window layout from 4 x 4 to 16 x 16 displays. See [Select Number of Displays and Layout](#).

MATLAB Function Blocks

Cell Array Support: Use additional cell array features in a MATLAB Function block

In R2016a, support for cell arrays in a MATLAB Function block includes:

Use of `{end + 1}` to grow a cell array

You can write code such as `X{end + 1}` to grow a cell array `X`. For example:

```
X = {1 2};  
X(end + 1) = 'a';
```

When you use `{end + 1}` to grow a cell array, follow the restrictions described in Growing a Cell Array by Using `{end + 1}`.

Value and handle objects in cell arrays

Cell arrays can contain value and handle objects. You can use a cell array of objects as a workaround for the limitation that code generation does not support objects in matrices or structures.

Function handles in cell arrays

Cell arrays can contain function handles.

Non-Power-of-Two FFT Support: Generate code for fast Fourier transforms for non-power-of-two transform lengths

In previous releases, code generation required a power of two transform length for `fft`, `fft2`, `fftn`, `ifft`, `ifft2`, and `ifftn`. In R2016a, code generation allows a non-power-of-two length for these functions.

Faster Standalone Code for Linear Algebra: Generate code that takes advantage of your own target-specific LAPACK library

To improve the simulation speed of MATLAB Function block algorithms that call certain linear algebra functions, the simulation software can call LAPACK functions. In R2016a, if you use Simulink Coder to generate C/C++ code for these algorithms, you can specify that the code generator produce LAPACK function calls. If you specify that you want to generate LAPACK function calls, and the input arrays for the linear algebra functions meet certain criteria, the code generator produces calls to relevant LAPACK functions. The code generator uses the LAPACKE C interface.

LAPACK is a software library for numerical linear algebra. MATLAB uses this library in some linear algebra functions, such as `eig` and `svd`. Simulink uses the LAPACK library that is included with MATLAB. Simulink Coder uses the LAPACK library that you specify. If you do not specify a LAPACK library, the code generator produces code for the linear algebra function instead of generating a LAPACK call.

See LAPACK Calls for Linear Algebra in a MATLAB Function Block.

To specify that you want to generate LAPACK function calls and link to a specific LAPACK library, see [Speed Up Linear Algebra in Code Generated from a MATLAB Function Block in the Simulink Coder documentation](#).

Concatenation of variable-size, empty arrays

In R2016a, the MATLAB Function block treatment of an empty array in a concatenation more closely matches the MATLAB treatment.

For concatenation of arrays, MATLAB and the MATLAB Function block require that corresponding dimensions across component arrays have the same size, except for the dimension that grows. For horizontal concatenation, the second dimension grows. For vertical concatenation, the first dimension grows.

In MATLAB, when a component array is empty, the sizes of the nongrowing dimensions do not matter because MATLAB ignores empty arrays in a concatenation. In previous releases, the MATLAB Function block required that the sizes of nongrowing dimensions of a variable-size, empty array matched the sizes of the corresponding dimensions in the other component arrays. A dimension size mismatch resulted in a simulation error.

In R2016a, for most cases of empty arrays in concatenation, the MATLAB Function block behavior matches MATLAB behavior. In some cases, if the MATLAB Function block does not recognize the empty array and treats it as a variable-size array, a dimension size mismatch results in a simulation error.

Consider the function `myconcat` that concatenates two arrays A and B.

```
function D = myconcat(n, B)
%#codegen
assert(n <= 5);
A = zeros(n,3);
C = [A, B];
D = numel(C);
end
```

The size of A is :5-by-3. The first dimension has a variable size with an upper bound of 5. The second dimension has fixed size 3. Suppose that B is a 5-by-5 array. If n is 0, A is an empty array whose size is 0-by-3. In previous releases, `myconcat` in a MATLAB Function block resulted in a size mismatch error because the size of dimension 1 of A did not match the size of dimension 1 of B. In R2016a, in a MATLAB Function block, the output from `myconcat` is 25. This output is the same as the output from `myconcat` in MATLAB.

Version History

When the result of the concatenation is assigned to a variable that must be a fixed-size variable, support for a variable-size, empty array in a concatenation introduces an incompatibility.

In previous releases, it was possible that a concatenation that included a variable-size array produced a fixed-size array because concatenation rules were stricter in the MATLAB Function block than in MATLAB. In R2016a, a concatenation that includes a variable-size array produces a variable-size array. If the result of the concatenation is assigned to a variable that must be a fixed-size variable, an error occurs.

Consider the function `myconcat1` that concatenates two arrays X and Y.

```
function c = myconcat1(n, Y)
%#codegen
assert(n <= 2);
X = zeros(n,2);
Z.f = [X Y];
c = numel(Z.f);
```

X has size :2-by-2. The first dimension has a variable size with an upper bound of 2. The second dimension has fixed size 2. Suppose that Y is a 2-by-4 array. In previous releases, the MATLAB Function block determined that the result of [X Y] had fixed size 2-by-6. In R2016a, the result of [X Y] has a size of 2-by-:6. The first dimension has a fixed size of 2 and the second dimension has a variable size with an upper bound of 6. This size accommodates an empty and nonempty X. If X is empty, the size of the result is 2-by-4. If X is nonempty, the size of the result is 2-by-6.

Consider the function myconcat2.

```
function c = myconcat2(n, Y)
%#codegen
assert(n <= 2);
X = zeros(n,2);
Z.f = ones(2,6);
myfcn(Z);
Z.f = [X Y];
c = numel(Z.f);
```

```
function myfcn(~)
```

myconcat2 assigns a 2-by-6 value to Z.f. The size of Z.f is fixed at 2-by-6 because Z is passed to myfcn. The result of the concatenation [X Y] is variable-size. The assignment Z.f = [X Y] results in an error because the left side of the assignment is fixed-size and the right side is variable-size.

To work around this incompatibility, you can use `coder. varsize` to declare that Z.f is variable-size.

```
function c = myconcat2(n, Y)
%#codegen
assert(n <= 2);
coder. varsize('Z.f');
X = zeros(n,2);
Z.f = ones(2,6);
myfcn(Z);
Z.f = [X Y];
c = numel(Z.f);
```

```
function myfcn(~)
```

xcorr Code Generation: Generate faster code for xcorr with long input vectors

For long input vectors, code generation for `xcorr` now uses a frequency-domain calculation instead of a time-domain calculation. The resulting code can be faster than in previous releases.

To use the `xcorr` function, you must have the Signal Processing Toolbox software.

More keyboard shortcuts for the MATLAB Function report

In R2016a, you can use keyboard shortcuts to perform the following actions in a MATLAB Function report.

Action	Default Keyboard Shortcut
Zoom in	Ctrl+Plus
Zoom out	Ctrl+Minus
Evaluate selected MATLAB code	F9
Open help for selected MATLAB code	F1
Open selected MATLAB code	Ctrl+D
Step backward through files that you opened in the code pane	Alt+Left
Step forward through files that you opened in the code pane	Alt+Right
Refresh	F5
Find	Ctrl+F

Your MATLAB preferences define the keyboard shortcuts associated with these actions. You can also select these actions from a context menu. To open the context menu, right-click anywhere in the report.

Zoom In	Ctrl+Plus
Zoom Out	Ctrl+Minus
Evaluate Selection	F9
Help on Selection	F1
Open Selection	Ctrl+D
Back	Alt+Left
Forward	Alt+Right
Refresh	F5
Find...	Ctrl+F
Page Source	

Code generation for Audio Toolbox functions and System objects

See Audio System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

Code generation for additional Computer Vision System Toolbox functions and objects

See C code generation support in the Computer Vision System Toolbox™ release notes.

Image Processing Toolbox Code Generation: Generate code for additional functions

See C-code generation support for more than 20 functions, including `regionprops`, `watershed`, `bweuler`, `bwlabel`, `bwperim`, and `multithresh` using MATLAB Coder in the Image Processing Toolbox™ release notes.

Code generation for additional MATLAB functions

Specialized Math in MATLAB

- `airy`
- `besseli`
- `besselj`

See Specialized Math in MATLAB in Functions and Objects Supported for C and C++ Code Generation — Category List.

Trigonometry in MATLAB

- `deg2rad`
- `rad2deg`

See Trigonometry in MATLAB in Functions and Objects Supported for C and C++ Code Generation — Category List.

Interpolation and Computational Geometry in MATLAB

- `interp`

See Interpolation and Computational Geometry in MATLAB in Functions and Objects Supported for C and C++ Code Generation — Category List.

Changes to code generation support for MATLAB functions

- Code generation now supports the `nanflag` option for `sum`, `mean`, `median`, `min`, `max`, `cov`, `var`, and `std`.
- Code generation for `ismember` no longer requires that the second input be sorted.

See Functions and Objects Supported for C and C++ Code Generation — Alphabetical List.

Code generation for additional Communications System Toolbox functions

- `convenc`

- `dpskdemod`
- `dpskmod`
- `qammod`
- `qamdemod`
- `vitdec`

See Communications System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

Code generation for additional DSP System Toolbox

- `audioDeviceWriter`
- `dsp.Differentiator`
- `designMultirateFIR`
- `dsp.SubbandAnalysisFilter`
- `dsp.SubbandSynthesisFilter`

DSP System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

Code generation for additional Phased Array System Toolbox functions

- `fogpl`
- `gaspl`
- `rainpl`
- `phased.BackscatterRadarTarget`
- `phased.LOSChannel`
- `phased.WidebandLOSChannel`

See Phased Array System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

Code generation for WLAN System Toolbox functions and System objects

See WLAN System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List

Units for MATLAB Function blocks

Simulink supports the specification of a unit property for data inputs and outputs of MATLAB Function blocks. Specify units for input and output data by using the **Unit (e.g., m, m/s², N*m)** parameter on the Ports and Data Manager.

During model update, Simulink checks for inconsistencies in units between input and output data ports and the corresponding Simulink signals.

In/Out Arguments: Specify same variable name for in/out arguments

In a MATLAB Function block, you can specify the same name for input and output arguments. For example, the block supports this code.

```
function y = fcn(y)
%#codegen
```

```
y = y + 1;
```

The corresponding input and output ports on the block have the same name, `y`.

UserData parameter available for storing values

Use `set_param()` and `get_param()` to store and retrieve values in the `UserData` parameter for MATLAB Function blocks.

Modeling Guidelines

High-Integrity Systems: Model object, file, and folder names

When you develop models for high-integrity systems, use the following guidelines for model object, file, and folder names:

- hisl_0031: File and folder names
- hisl_0032: Model object names

If you have a Simulink Verification and Validation™ license, you can use the following Model Advisor standards checks to verify compliance with high-integrity guideline hisl_0032: Model object names. The checks are available in the applicable Model Advisor **By Task** folder.

- DO-178C/DO-331: Check model object names
- IEC 61508, EN 50128 and ISO® 26262: Check model object names

Model Advisor

Additional functionality for Model Advisor check that checks for usage of partial structure

In R2016a, the Model Advisor check **Check for partial structure parameter usage with bus signals** has a new name, **Check structure parameter usage with bus signals**. The check uses a new programmatic ID, `mathworks.design.MismatchedBusParams`.

If you have Simulink Coder software, before you generate code from a model, use this check to discover potential inefficient typecasts due to mismatched data types. For information about the changes to the check functionality, see Model Advisor check for data type mismatches between bus elements and structure fields.

Version History

If you have scripts that programmatically run this check, consider using the new programmatic ID. The old programmatic ID is `mathworks.design.PartialBusParams`. For example, your scripts might use this command:

```
SysResultObjArray = ModelAdvisor.run({'myModel'}, {'mathworks.design.PartialBusParams'})
```

Your scripts continue to work with the old ID. The new ID is `mathworks.design.MismatchedBusParams`.

In a future release, the check will appear in the **By Product > Simulink Coder** folder instead of the **By Product > Simulink** folder. You will need Simulink Coder software to run the check. If you do not have Simulink Coder software, consider removing the check from your scripts in R2016a.

S-Functions

ssSetSolverNeedsReset updates

The `ssSetSolverNeedsReset` macro is now needed for both fixed-step and variable-step ODE solvers. Simulink now monitors S-function continuous state changes without solver resets in normal and accelerated simulation modes for fixed-step simulation.

To improve performance, consider updating existing S-functions that correctly use `ssSetSolverNeedsReset` with the new macro, `ssSetSkipContStatesConsistencyCheck`.

ssSetSkipContStatesConsistencyCheck

The `ssSetSkipContStatesConsistencyCheck` macro is a new macro that lets you skip continuous state consistency checks. Consider using this macro for an S-function with continuous states that either does not change the state vector or changes it only in conjunction with `ssSetSolverNeedsReset`.

R2015aSP1

Version: 8.5.1

Bug Fixes

R2015b

Version: 8.6

New Features

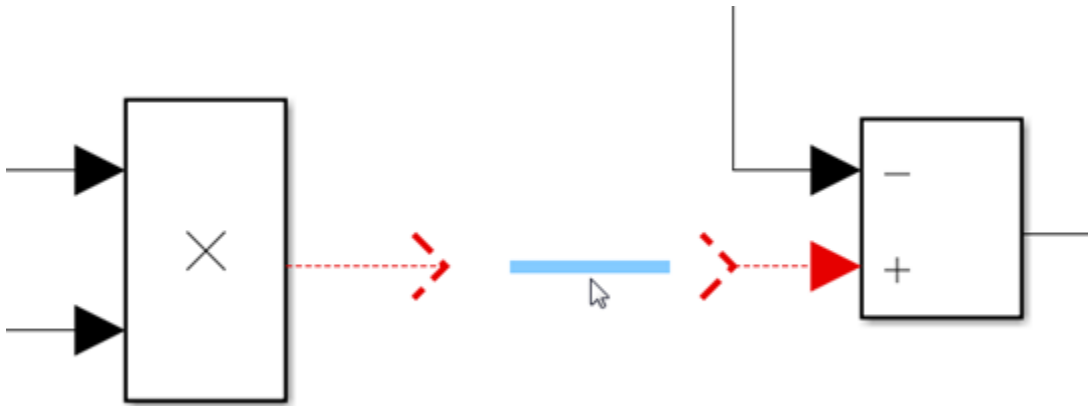
Bug Fixes

Version History

Simulink Editor

Signal Line Healing: Click once to repair broken signal lines after deleting blocks

You can connect signal lines that broke when you deleted a block. When you delete a block that was connected to one input signal and one output signal, a prompt appears in its place. Click the prompt to connect the broken input and output signal lines to each other.



Multilingual Names and Comments: Use any language to write block names, signal names, and MATLAB Function comments

You can use Unicode characters for most textual content in Simulink, including names of blocks and signals as well as comments in MATLAB scripts or functions.

Programmatic removal of mask dialog box controls and mask parameters

You can use the following functions to remove dialog control elements and parameters from the mask dialog box:

- `removeParameter`: Removes a parameter from mask dialog box. For more information, see `Simulink.Mask.removeParameter`.
- `removeDialogcontrol`: Removes a dialog control element from mask dialog box. For more information, see `Simulink.Mask.removeDialogControl`.

Alternative view of library contents in Library Browser

You can now view blocks in the Simulink Library Browser in alphabetical order or in the order specified by the developer of the library. Right-click the blocks pane in the Library Browser, and select **Sort in library model order**. The blocks appear in the order specified by the developer of the library in the underlying library model. This setting takes effect on all your libraries and stays in effect from session to session.

To return to alphabetical order, right-click and select **Sort in alphabetical order**. For more information, see Reorder Blocks in Libraries.

Prompt to set key parameter when dragging a block from the Library Browser

When you drag a block from the Library Browser into your model, you are prompted to enter the key parameter. Setting the value this way lets you set the parameter without opening the dialog box.

Printing to Postscript and EPS file formats

You now cannot print Simulink models or Stateflow charts to postscript and EPS file formats.

Version History

If you want to print models or charts to a file, specify a .pdf extension for the output file.

Programmatic addition of areas and images in models

You can now add labeled areas and images to your Simulink model programmatically. For more information, see `add_block`.

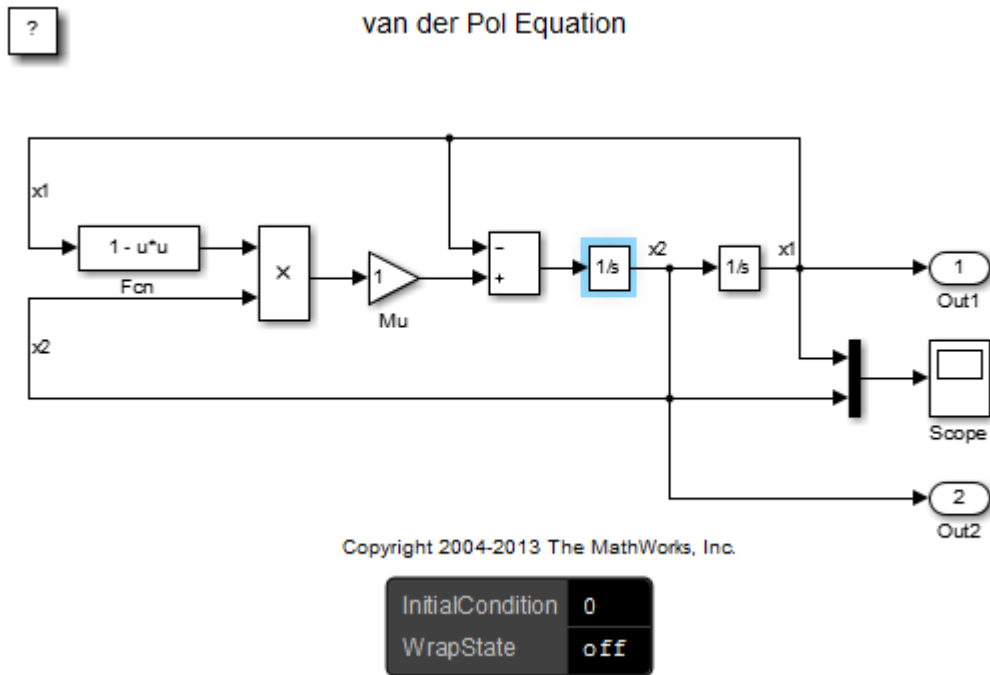
Redesigned interface for Model Dependency Viewer

The Model Dependency Viewer tool has a new interface to view a model's dependencies on models and libraries. For more information, see Model Dependency Viewer.

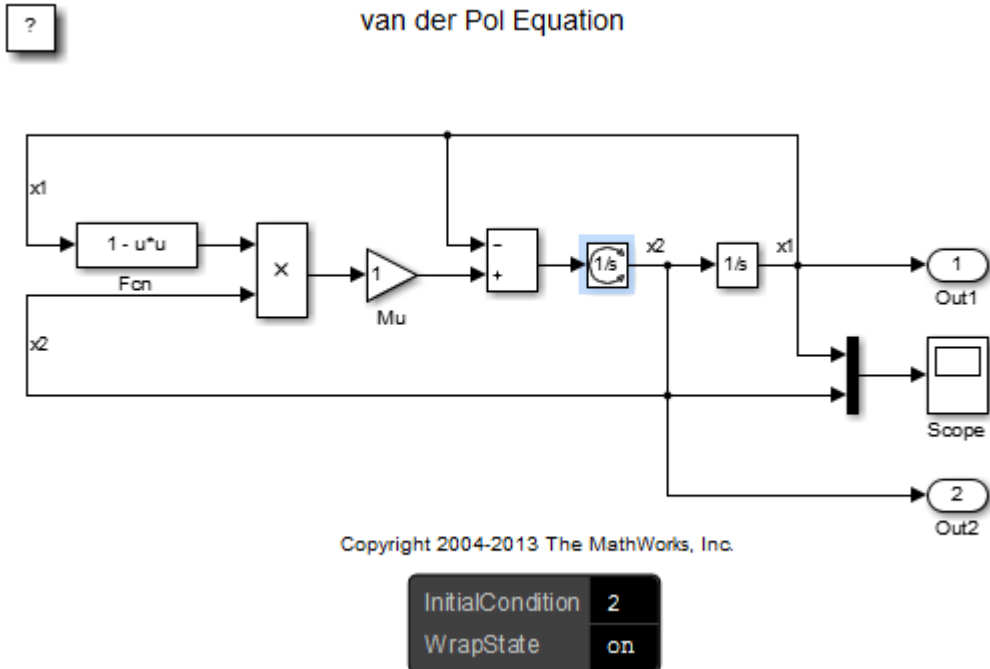
Visual cue for undo and redo of block parameter value changes

After you undo or redo block parameter changes, a visual cue appears that shows the current values of the affected parameters. For example:

- 1 In the `vdp` model, for an Integrator block, change the **Initial condition** parameter value to 2 and select the **Wrap state** check box. Apply the changes.
- 2 Select **Edit > Undo Parameter Changes**. The cue shows that the parameters returned to their original values.



- 3 Select **Edit > Redo Parameter Changes**. The cue shows the restored parameters values.



For more information on Undo and Redo, see Interactive Model Building.

Simulation Analysis and Performance

New Interface for Scopes: View and debug signals with cursors and measurements

The Simulink Scope and Floating Scope blocks with the Scope Viewer were significantly upgraded. Simulink upgrades the scope blocks in a model that you previously created and saved when you reopen it.

Major new features include simulation data analysis and debugging tools directly within the Scope window, and tools similar to those of modern hardware bench-top oscilloscopes. See Scope block reference for a list of features.

Note The Signal Statistics, Bilevel Measurements, and Peak Finder panels require a DSP System Toolbox or Simscape license.

Features from the previous scope were retained with some differences.

Capability	Previous Simulink Scope block	New Simulink Scope block
Number of input ports	No limit	Maximum 96
Number of axes (displays)	No limit	Maximum 16
Mapping axes (displays) to input ports	Number of axes equal to the number of input ports.	Number of displays and number of input ports are independent. Ports are assigned to displays sequentially, one port per display. If there are more ports than displays, the extra signals are plotted in the last display.
Mapping axes (displays) to signals	For the Floating Scope, select a specific axis and assign signals to axis using the Signal Selector.	For the Floating Scope, the number of displays and the number of attached signals are independent. Signals are assigned to displays sequentially, one signal per display. If there are more signals than displays, the extra signals are plotted in the last display.
Default logging format	Structure with time	Dataset
Default line color	Color order yellow, magenta, cyan, red, green, light gray.	Color order yellow, blue, orange, green, purple, cyan, magenta.
Model Explorer Search	Search by Property Name, by Property Value, or by Dialog Prompt.	Not possible to search property names or values.

Capability	Previous Simulink Scope block	New Simulink Scope block
Block and dialog parameters	The command <code>get_param(<scope block path>, 'ObjectParameters')</code> returns a list of block parameters and dialog parameters.	The command <code>get_param(<scope block path>, 'ObjectParameters')</code> returns a list of block parameters. There are no dialog parameters. The command <code>get_param(<scope block path>, 'DialogParameters')</code> returns empty.
Last buffer Data Archiving	Last data buffer in external mode data archiving saved to the workspace by default	Data saved to workspace when Write Intermediate Results to WorkSpace is selected
Decimation and Sample Time	Sample time and decimation settings are not independent.	Sample time and decimation settings are independent.
Select signals for Floating Scope	Several Signal Selectors opened at once. Selection from the model diagram and from the Signal Selector poorly coordinated.	Only one Signal Selector opened at a time. Selecting signals with the Signal Selector disables selection from the model diagram. To select signals from the model diagram, unlock the Floating Scope, and then select signals.
Convert Scope block to Floating Scope block and back	Convert Scope to Floating Scope using a Floating Scope check box parameter.	Conversion not available. Get a Scope or a Floating Scope from the sinks library.
Floating property	Possible to convert a Scope block to a Floating Scope block using the command <code>set_param([model '/ Scope'], 'Floating', 'on')</code>	The Floating property is read-only and set to 'off'. Not possible to convert a Scope to a Floating Scope.
Hybrid signal plot style	Continuous elements of a hybrid signal (mux of continuous and discrete) plotted using a line and discrete elements plotted using stairs.	Signal elements of a hybrid signal plotted using lines. Behavior is consistent with a Dataset logging timeseries plot of hybrid signals.
Programmatic access and scripting	Use <code>set_param</code> and <code>get_param</code> . Changing Scope dialog parameters using <code>set_param</code> can put the Scope in an undefined state.	Use Scope Configuration object. For backward compatibility, <code>set_param</code> and <code>get_param</code> continue to support dialog parameters in the old Scope. However, these parameters are hidden.

Capability	Previous Simulink Scope block	New Simulink Scope block
Legend location	Location of a legend on a scope serialized and restored when a model is re-opened.	Placed legends anywhere within a scope, but the location of the legend is not serialized.
Zoom Mode	ZoomMode serialized and restored when a model is re-opened.	ZoomMode not serialized and not restored when a model is re-opened.
Scope blocks in conditionally executed subsystems	Scope data is not acquired when subsystem is not enabled. Lines drawn connecting gaps in plotted data.	Scope data is not acquired when subsystem is not enabled. Gaps in plotted data visible.

Fast Restart API: Programmatically run consecutive simulations more quickly

You can now enable fast restart from the command line using `set_param`. See [Get Started with Fast Restart](#) for more details.

You can also simulate a model in fast restart using `sim` and `cvsim` commands.

Previously, you could not use `cvsim` or `signalbuilder` with fast restart. In R2015b, you can now use these commands.

Auto solver that chooses solver for a model

Starting with R2015b, you can use the auto solver to select a solver and step size for simulating a model. The auto solver suggests a fixed-step or variable-step solver along with maximum step size based on the dynamics of the model. Select the auto solver in the solver pane and accept recommended settings in the solver information dialog box. For more information, see [Use Auto Solver to Select a Solver](#).

Tunability of struct parameters in rapid accelerator mode

You can now tune struct parameters when you simulate a model in rapid accelerator mode.

Previously you tuned struct parameters only in normal and accelerator modes. You do not need to regenerate the rapid accelerator code when you tune struct parameters.

Port value labels for nonvirtual buses and bus signals

To monitor bus signal data during simulation, you can use port value labels for nonvirtual buses. You can also display port labels for individual signals in a bus through a new interface. For more information, see [Display Value for a Specific Port](#).

Visualization of inserted rate transition blocks

When Simulink performs automatic rate transition, you can now see the rate transition blocks that Simulink inserts in your model. You can also set the **Initial Condition** of these blocks and change block parameters for rate transfers. For more information, see [Visualize Inserted Rate Transition Blocks](#).

Common format for saving states, output, and final states data and other logging and loading techniques

The default for the **Configuration Parameters > Data Import/Export > Format>** parameter is now Dataset, which:

- Simplifies postprocessing of logged data
- Makes it easier to take advantage of features that require Dataset format

Extended support for root Inport loading using Dataset format in rapid accelerator

Loading root Inport data using Dataset format in rapid accelerator mode is now supported for underspecified and complex buses and arrays of buses.

Free MinGW-w64 compiler for running simulations on 64-bit Windows®

You can now use the MinGW-w64 compiler from TDM-GCC to run simulations on 64-bit Windows® hosts. You can run simulations in Accelerator and Rapid Accelerator modes, build model reference simulation targets and S-functions, and simulate MATLAB Function blocks. To download and install the compiler, see [Install MinGW-w64 Compiler](#).

Component-Based Modeling

More flexible configuration of Application lifespan (days) parameter in a model reference hierarchy for simulation

The **Configuration Parameters > Optimization > Application lifespan (days)** parameter can now be different for a parent and child model in a model reference hierarchy during simulation. This flexibility can be useful during model reference development. However, for code generation, the settings for this parameter must be consistent throughout a model reference hierarchy.

Model Advisor checks for simplified initialization mode

The Model Advisor **Check consistency of initialization parameters for Outport and Merge blocks** check has been replaced with four new checks. These checks help to migrate your model to simplified initialization mode. For more information, see the Model Advisor task Migrating to Simplified Initialization Mode Overview.

Changes to export-function models

Export-function models include these changes for R2015b.

Configuration Parameter for Scheduling Checks

A new model referencing configuration parameter, **Enable strict scheduling checks for referenced export-function models**, lets you enable or skip checks on scheduling order and sample-time consistency. This check applies to referenced export-function models. For more information, see Enable strict scheduling checks for referenced export-function models.

Triggered Sample Time for Function-Call Subsystems

You can now specify a discrete sample time for the function-call root-level Inport block for function-call subsystems whose trigger block has **Sample time type** set to Triggered. For more information, see Sample Time for Function-Call Subsystems in Export-Function Models.

Execution Order of Function-Call Root-Level Inport Blocks

The algorithm to determine block execution order has changed. Root-level function-call Inport block execution order is now determined first by priority, then by sample time, and then by the port number. For more information, see Execution Order for Function-Call Root-level Inport Blocks.

Version History

The algorithm to determine block execution order has changed. Specify low priority on the blocks you want to execute first.

Saving of list view parameters with Simulink.ConfigSet.saveAs

Previously, `Simulink.ConfigSet.saveAs` saved a configuration set that included the enabled parameters that appear in the category view in the Configuration Parameters dialog box. In R2015b, the method also saves the enabled parameters that appear only in the list view.

Project and File Management

Referenced Projects: Create reusable components for large modeling projects

You can organize large projects into components through the use of referenced projects. Componentization facilitates reuse, modular development, unit testing, and independent release of components. In R2015b, you can:

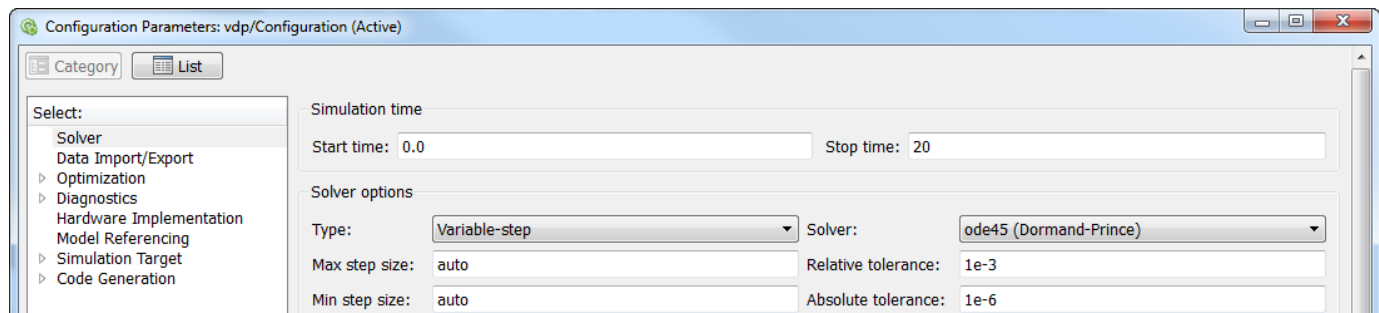
- Add new components to your project by referencing other projects.
- Use shortcuts to view and run files that belong to the referenced project.
- Extract a folder from a project and convert the folder into a referenced project.

For more information, see:

- [Componentization of Large Projects](#)
- [Airframe Project Reference Example](#)

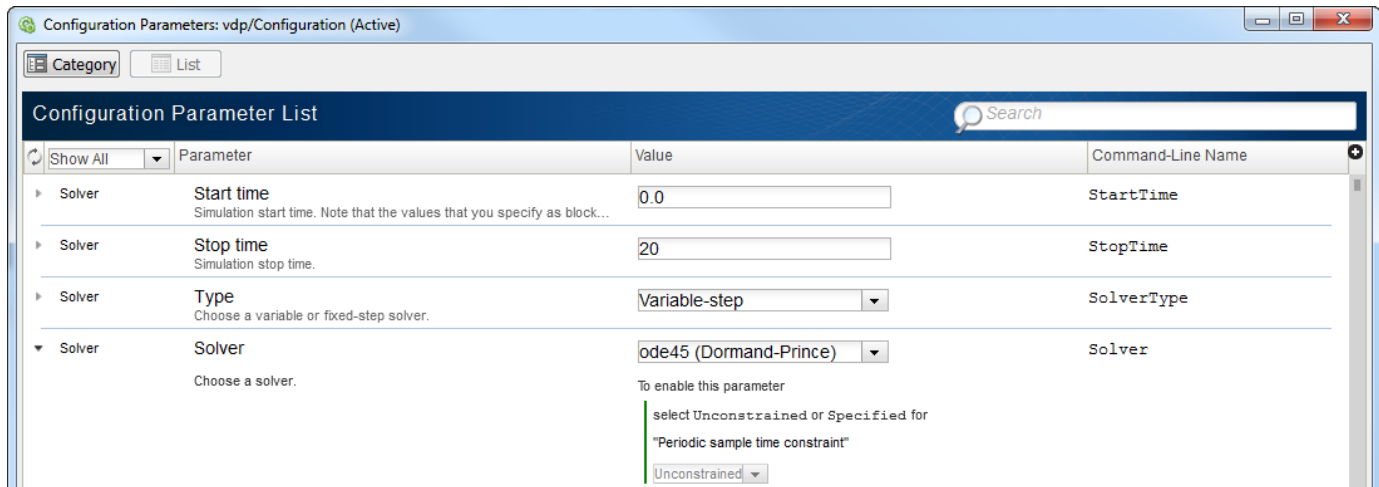
Configuration Parameters List View: List, edit, and search all configuration parameters within your model

The Configuration Parameters list view provides a complete list of parameters in the model configuration set. The list view includes parameters available in the category view and parameters previously available only from the command line. To use the list view, click the **List** button at the top of the Configuration Parameters dialog box.



You can use the list view to:

- Search for specific parameters or filter parameters by category.
- Sort parameters.
- Edit parameter values.
- View parameter dependencies.
- Get parameter names for use in scripts.



For more information on the Configuration Parameters dialog box and the list view, see Configuration Parameters Dialog Box Overview.

Project Creation from a Model: Quickly organize your model and all dependent files

Put your model and all dependent files into a Simulink project in three clicks. If your model has supporting files, then a project makes it easier to manage. If you have more than one model, version, or user, a project can help you. The project provides these advantages:

- Automate setup and shutdown tasks such as loading data and running scripts.
- Manage any required files (not just models and libraries, but all your supporting files, such as data, images, scripts, functions, data dictionaries, requirements docs, etc.).
- Manage your path when you open and close the project.
- Visualize all file dependencies.
- Use source control tools and compare versions.

In a Simulink model, select **File > Simulink Project > Create Project from Model**.

Simulink runs dependency analysis on your model to identify required files and suggest a project root location that contains all dependencies.

For details, see Create a Project from a Model.

Faster, Improved Dependency Analysis: Analyze projects several times faster, identify referenced project files, and view library blocks

Dependency analysis of typical projects is faster, and includes new capabilities:

- Identify dependencies on files in referenced projects.
- Trace library link dependencies to view library block names in the graph.

Analyzing all project files is now the default, which helps to ensure that you explore all dependencies.

The dependency analysis workflow is simplified so that you can perform all operations in the graph view using one toolstrip tab. The Simulink Project tree now has a single **Dependency Analysis** view instead of three views separating file selection for analysis, the table of results, and the Impact graph. The table view of files is still available as an option, but you can perform all operations from the Impact graph view.

For details, see Perform Impact Analysis.

Management of shadowed and dirty files

Simulink Project now helps you to avoid working on the wrong files by detecting shadowed files when you open a project. If loaded model files shadow your project model files, you are prompted to inspect or close them. To avoid working on the wrong files, close the files that shadow your project files before continuing to open your project. You can choose to show or close individual files, or close all files that shadow with one click. Previously, you only saw a warning in the command window when you opened a model that another loaded file shadowed.

Simulink Project also now helps you to manage unsaved changes when closing a project. When you close a project, it closes any project models that are open, unless they are dirty. If model files have unsaved changes, then a prompt appears. You can see all dirty files, grouped by project if you have referenced projects. To avoid losing work, you can save or discard changes by file, by project, or globally.

For details, see Manage Shadowed and Dirty Model Files.

Comparison of any pair of file revisions

In a Simulink project under source control, you can now select any pair of revisions to compare. This ability is useful for investigating differences in older changelists. You can now sort revisions by column headers in the Compare to Revision dialog box (e.g., by date, revision number, or author). The Compare to Revision dialog box has an improved layout, placing revision columns above the details pane, making it much easier to read.

Previously you could compare a revision only with your local file, making it more difficult to investigate older changes. You could not compare between other revisions, and you could not sort revisions.

For details, see Compare Revisions.

Updated power window example

The Power Window Control Project example has been updated to exercise more of the logic in the power window controller. The example uses the From Spreadsheet block to provide multiple sets of inputs to the controller. For more information, see Power Window.

Case-sensitive model and library names

Function Element Name	What Happens When You Use This Function Element	Use This Instead	Compatibility Considerations
Case-insensitive model or library name	Simulink uses the closest case-insensitive match, and warns	Specify exact case	Case-insensitive matching will be removed in a future release.

Warning for Model Info Configuration Manager

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
Model Info block Configuration Manager	When a model's configuration manager is active, you see a new warning when loading or saving the model, or when you access the Model Info block.	For a more flexible interface to source control tools, use Simulink Project instead of the Model Info block.	The Configuration Manager will be removed in a future release. For help, see the Upgrade Advisor checks: Identify Model Info blocks that use the Configuration Manager and Identify Model Info blocks that can interact with external source control tools.

Data Management

Interval Logging: Specify start and stop time intervals to log only the data you need

You can use the new **Configuration Parameters > Data Import/Export > Logging intervals** parameter to specify an interval for logging data for:

- Time
- States
- Output
- Signal logging
- The To Workspace block
- The To File block

Limiting logging to specified intervals:

- Allow you to view specific logged data without changing the model or adding complexity to a model
- Simplifies analysis of logged data
- Reduces simulation time
- Reduces memory consumption

Always-On Tunability: Tune all block parameters and workspace variables during a simulation

Prior to R2015b, you selected **Configuration Parameters > Optimization > Signals and Parameters > Inline parameters** to increase simulation speed by treating block parameter values as constants. In this case, you could not tune these parameters during simulation. To preserve tunability for individual block parameters, you used data objects or the Model Parameter Configuration dialog box. Alternatively, clearing **Inline parameters** enabled block parameter tunability by default while reducing simulation speed.

In R2015b, you can tune block parameters during simulation, and you retain the simulation speed benefit. You do not need to use data objects or the Model Parameter Configuration dialog box to preserve parameter tunability during simulation. **Inline parameters** is now called **Default parameter behavior**. This configuration parameter does not affect simulation.

Prior to R2015b, when you displayed sample time colors, magenta indicated constant sample time. Now, magenta indicates blocks whose output values are constant. The term constant refers to blocks whose output values change only when you tune block parameters.

The table describes some block and system level changes caused by these enhancements.

Description of change	Behavior in R2015a	Behavior in R2015b
Constant block with sample time set to <code>inf</code>	Inline parameters selected: The block receives constant sample time. When you display sample time colors, the block appears magenta in color.	The block computes its output at the start of simulation and whenever you tune a block parameter. When you display sample time colors, the block appears magenta in color.
	Inline parameters cleared: The block inherits a sample time from its context. The block color indicates the inherited sample time.	
Relationship between sample time and parameter tunability	If you use a tunable parameter, such as a <code>Simulink.Parameter</code> object with a storage class other than <code>Auto</code> , to specify a numeric block parameter, the block inherits a sample time from its context.	The block computes its output at the start of simulation and whenever you tune a block parameter. When you display sample time colors, the block appears magenta in color.
Weighted sample time math block	Inline parameters selected: If any of the block inputs are constant, Simulink displays an error.	The block inherits a sample time from its context.
	Inline parameters cleared: The block inherits a sample time from its context.	

For more information about changes to the configuration parameter **Inline parameters** with respect to code generation, see the R2015b Simulink Coder release notes.

For more information about constant sample time, see Constant Sample Time.

Version History

- An error occurs in conditionally executed subsystems where:
 - A divide-by-zero operation occurs and all the blocks in the computation have constant output values.
 - The control input of an Index Vector block specifies an index that is out of range, and the block input has a constant value.

Workaround: Use Upgrade Advisor to run **Check model for parameter initialization and tuning issues**. For more information, see Check model for parameter initialization and tuning issues.

- Prior to R2015b, if a model contained Level-2 MATLAB S-Function blocks, and you selected **Inline parameters**, these blocks allowed constant sample time by default. In R2015b, Level-2 MATLAB S-Function blocks inherit a sample time from their context by default.

Workaround: To enable these blocks to allow constant sample time, explicitly set `SetAllowConstantSampleTime` to `true`.

- If you set **Default parameter behavior** to **Inlined**, and you log blocks that have tunable parameters, a mismatch can occur in the number of data points logged between the simulation and code generation workflows. However, the value logged remains constant where extra points are logged.
- In the code generation workflow, when the Merge block receives a constant value and non-constant sample times, one of these conditions must hold. Otherwise Simulink displays an error.
 - The source of the constant value is a grounded signal.
 - The source of the constant value is a constant block with a non-tunable parameter.
 - There is only one constant block that feeds the Merge block.
 - All other input signals to the Merge block are from conditionally executed subsystems.
 - The Merge block and output blocks of all conditionally executed subsystems should not specify any initial outputs.

For more information, see Merge.

- If you generate subsystem code for a subsystem that has any constant inputs, an error can occur.

Workaround: Change the sample time at the source to an inherited or periodic sample time.

- Models that had **Inline parameters** cleared and simulated without error in releases prior to R2015b can have rate transition issues in R2015b. These issues occur because all constant blocks now receive constant sample time regardless of the setting for **Default parameter behavior**. This change can produce different sample time propagation results.

Workaround: Use Upgrade Advisor to run **Check model for parameter initialization and tuning issues**. For more information, see Check model for parameter initialization and tuning issues.

- Suppose you generate code for a model where these conditions are true:
 - The default parameter behavior is tunable.
 - A constant block executes with a downstream rate that is not the fastest rate.

The logged data appears to be delayed. There is a delay equal to one step of the slower rate that the block executes at. Simulink displays a warning during code generation.

This issue does not affect signal logging during simulation.

- If you write TLC code to generate code from an inlined S-function, and if the TLC code contains an **Outputs** function, you must modify the TLC code if all of these conditions are true:
 - An output port uses or inherits constant sample time. The output port has a constant value.
 - The S-function is a multirate S-function or uses port-based sample times.

In this case, the TLC code must generate code for the constant-valued output port by using the function **OutputsForTID** instead of the function **Outputs**. For more information, see Specifying Constant Sample Time (Inf) for a Port.

- Previously, if you selected **Inline parameters**, parameters that used the storage class **Auto** (including numeric MATLAB variables) were not tunable during simulation. The code generated for models referenced in accelerator mode and the code generated for top models or freestanding models in rapid accelerator mode inlined the numeric values of the variables. In R2015b, these parameters are tunable, and the code preserves them.

- These parameters must exist throughout simulation. For example, if you reference a model in accelerator mode, you cannot use the referenced model `CloseFcn` callback to clear the parameters. Consider using the callback of the parent model instead. Alternatively, store the parameters in a Simulink data dictionary.
- Suppose that you use a variable `myVar` in an inconsistent way, for example:
 - You use `myVar` as the value of the **Gain** parameter of a Gain block in a referenced model `submodel1`. The Gain block applies a data type, such as `int32`, to `myVar`.
 - You also use `myVar` in a Gain block in the referenced model `submodel2`. This Gain block applies a different data type, such as `int16`, to `myVar`.

In R2015b, you cannot use `myVar` in an inconsistent way across these referenced models because `myVar` is tunable. To simulate the models, resolve the inconsistent usage. For example, modify the referenced models `submodel1` and `submodel2` so they apply the same data type to `myVar`.

- Previously, the code generated in rapid accelerator mode did not preserve expressions, such as `myVar + myOtherVar * 5`, that you used to specify block parameter values. In R2015b, the code preserves these expressions.
 - If a block parameter value references workspace variables, you cannot change the block parameter value during rapid accelerator simulation. Instead, you can tune the values of the referenced variables. If you use a script to change the block parameter value during simulation, modify the script so it changes the values of the variables instead.
 - Suppose that a For Each subsystem has a mask parameter `a` whose value is the three-element array `[1 2 3]`. Suppose that the subsystem partitions the parameter `a` three times so that each iteration of the subsystem uses one of the elements in the array. You cannot use the expression `15.23^a` to specify a block parameter value inside the subsystem because the expression contains an operator that the code generator does not support.

To simulate the model, consider implementing the expression as a block algorithm instead. For more information about the operators that the code generator supports in tunable expressions, see [Tunable Expression Limitations](#).

- Previously, you could select **Inline parameters** to prevent mask initialization code from executing during simulation. This technique prevented code that made changes to the model from executing. If the changes generated errors during simulation, for example by attempting to add blocks to the model, selecting **Inline parameters** prevented the errors.

In R2015b, all mask initialization code executes during simulation. To avoid errors during simulation, consider wrapping the mask initialization code in additional code that prevents it from executing during simulation.

Arrays of structures as parameters



You can use arrays of structures as parameters to:

- Initialize arrays of bus signals that pass through blocks like Unit Delay. Arrays of buses help you to reduce signal line density in model diagrams. For more information, see [Specify Initial Conditions for Bus Signals](#).
- Specify the **Constant value** parameter in a Constant block. You can use this technique to compactly represent multiple constant-valued signals as an array of buses. For an example, see the Constant block.

- Group workspace variables into a single variable whose value is an array of structures. You can use this technique to organize related variables and reduce workspace clutter. For more information about structure parameters, see [Organize Related Parameters in Structures and Arrays of Structures](#).
- Parameterize a For Each Subsystem, which can help you repeat an algorithm over multiple inputs. The subsystem can partition mask parameters that are arrays of structures. For an example, see [Repeat an Algorithm Using a For Each Subsystem](#).

Improved methods to create custom data objects

Model Explorer

Previously, to create data objects such as `Simulink.Parameter` and `Simulink.Signal`, you used the Model Explorer buttons **Add Parameter**  and **Add Signal** . To create data objects from a package other than `Simulink`, you specified the package using Simulink Preferences.

In R2015b, to select a data class other than those in the package `Simulink`, you can use the new arrows beside these buttons. You can select any data class whose definition is on the MATLAB path, including custom classes such as `myPackage.myParameter`.

For more information about creating data objects, see [Data Objects](#).

Data Object Wizard

Previously, to create custom data objects using the Data Object Wizard, you specified a default data class package using Simulink Preferences.

In R2015b, to create objects from data classes in any package, including your own packages, you can select the classes using the Data Object Wizard.

For more information about creating data objects using the Data Object Wizard, see [Create Data Objects for a Model Using Data Object Wizard](#).

No creation of parameter objects for mask initialization code

You now cannot use mask initialization code to create parameter objects. Parameter objects are objects of the class `Simulink.Parameter` and of any of its subclasses that you create. For more information about block masking and using MATLAB code to initialize a mask, see [Initialize Mask](#).

Version History

If you use existing mask initialization code that creates parameter objects, you must edit the code so that it does not create parameter objects.

Sample time for signal logging

In the Signal Properties dialog box, you can use the new **Sample Time** option for a signal marked for signal logging. This option:

- Maintains the separation of design and testing, because you do not need to insert a Rate Transition block to have a consistent sample time for logged signals

- Reduces the amount of logged data for continuous time signals, for which setting decimation is not relevant
- Eliminates the need to postprocess logged signal data for signals with different sample times

For details, see Set Sample Time for a Logged Signal.

Same format for logging states, output, and final states as used for other logging and loading techniques

The default for the **Configuration Parameters > Data Import/Export > Format** parameter is now **Dataset**, which:

- Simplifies post processing of logged data
- Makes it easier to take advantage of features that require **Dataset** format

Root Inport loading in rapid accelerator mode using Dataset format

In rapid accelerator mode, using **Dataset** format to load root Inport blocks now supports:

- Specifying a single **Dataset** object, as an alternative to specifying a long comma-separated list using the **Configuration Parameters > Data Import/Export > Input** parameter
- Loading of buses and arrays of buses, including underspecified buses and arrays of buses
- Loading of fixed-point and enum data
- No code regeneration for new data

Logged signals with propagated names

Signal logging and root Outport block logging data for a signal captures the propagated signal name if the logging format is **Dataset** and:

- For signal logging, you:
 - Mark one or more signals for signal logging and in the Signal Properties dialog box select **Show Propagated Signals**.
 - Enable **Configuration Parameters > Data Import/Export > Signal logging**.
- For root Outport block logging, you select **Configuration Parameters > Data Import/Export > Output**.

The `Simulink.SimulationData.Signal` class has a new `PropagatedName` property for displaying the propagated signal name. The `Simulink.SimulationData.Dataset` class displays propagated signal names as a comment (the propagated signal name preceded by a percent sign (%)). In the logged data, the propagated signal name does not include angle brackets (<>).

Tolerance for data type mismatch between bus elements and structure fields

Previously, when you used a MATLAB structure to initialize a bus signal, or to drive a bus signal using a Constant block, you matched the data types of the structure fields with those of the bus signal elements.

In R2015b, you do not need to match the data types when you simulate a model. You can use doubles to specify the structure field values, and use the bus signal elements to control the data types. Prior to simulation, the bus elements cast the structure field values.

However, for other applications such as creating tunable initial conditions in the generated code, you must match the data types.

To decide whether to explicitly specify field data types, see [Decide Whether to Specify Data Types for Structure Fields](#).

Summary of changes made to data dictionary

If you use a Simulink data dictionary to store the variables that a model uses, you can display a summary of all of the unsaved changes that you made to the dictionary. You can use this technique to:

- Track your changes while you create and modify dictionary entries.
- Decide which changes to keep or discard before you close an unsaved dictionary.
- Recover variables that you deleted.
- Recover dictionary references that you removed.

For an example, see [View and Revert Changes to Entire Dictionary](#).

Rename All in Goto blocks

You can use the **Rename All** button in Goto block dialog boxes to quickly rename the corresponding tag in From and Goto Tag Visibility blocks. For more information, see the [Goto block](#).

Change to visibility of SamplingMode property of signal objects

The `Simulink.Signal` property `SamplingMode` is hidden if you set it to `'auto'`, the default value. Previously, when you used a `Simulink.Signal` object to define a data store, you set `SamplingMode` to `'Sample based'`. You can now leave the property at the default value, `'auto'`.

If you use existing code that creates `Simulink.Signal` objects and changes the value of the `SamplingMode` property, the property is not hidden for these objects.

This change supports recent changes to frame-based processing with DSP System Toolbox. For more information, see [Sample- and Frame-Based Concepts in the DSP System Toolbox documentation](#).

Continued availability of Simulink.saveVars

The R2014a Release Notes state that the function `Simulink.saveVars` will be removed in a future release. As of R2015b, you can continue to use this function. Existing scripts that use `Simulink.saveVars` do not generate warnings.

However, `Simulink.saveVars` is not recommended. To save workspace variables to a MATLAB script, use `matlab.io.saveVariablesToScript` instead of `Simulink.saveVars`.

Simulink.SimulationData.Dataset updates

`Simulink.SimulationData.Dataset.find` is a new method that finds elements with specified property names and values. For documentation on this method, in the MATLAB Command Window, type `help Simulink.SimulationData.Dataset.find`.

Edit Input button is now Connect Input

The **Input** > **Edit Input** button on the Configuration Parameters Data Import/Export pane is now **Connect Input**.

Legacy Code Tool support for conditional outputs

When you use the `legacy_code` function, you can now specify whether the legacy code conditionally writes the output ports. Use the `outputsConditionallyWritten` S-function option. If `true`, the generated S-function specifies that the memory associated with each output port cannot be overwritten and is global (`SS_NOT_REUSABLE_AND_GLOBAL`). If `false`, the memory associated with each output port is reusable and is local (`SS_REUSABLE_AND_LOCAL`). By default, the value is `false` (0).

Connection to Hardware

Raspberry Pi 2 Support: Run Simulink models on Raspberry Pi 2 Model B hardware

You can use the Simulink Support Package for Raspberry Pi Hardware with Raspberry Pi 2 Model B hardware.

Arduino Yun: Design and run Simulink models on Arduino Yun hardware

You can use the Simulink Support Package for Arduino Hardware with Arduino Yun hardware.

Hardware Implementation Selection: Quickly generate code for popular embedded processors

Specification of hardware configurations has been simplified. Top-level Configuration Parameters dialog box panes, **Run on Target Hardware** and **Coder Target**, have been removed. Parameters previously available on those panes now appear on the **Hardware Implementation** pane. A parameter has also moved from the **Code Generation** pane to the **Hardware Implementation** pane.

This list summarizes the R2015b changes and new behavior:

- By default, the **Hardware Implementation** pane lists **Hardware board**, **Device vendor**, and **Device type** parameter fields only.
- If you use Simulink without a Simulink Coder license, initially parameters on the **Hardware Implementation** pane are disabled. To enable them, click **Enable hardware specification**. The parameters remain enabled for the current MATLAB session.
- By default, the **Hardware board** list includes: None or Determine by Code Generation system target file, and Get Hardware Support Packages. After installing a hardware support package, the list also includes corresponding hardware board names.
- If you select a hardware board name, parameters for that board appear in the dialog box display.
- Lists for the **Device vendor** and **Device type** parameters have been updated to reflect hardware that is available on the market. The default **Device vendor** and **Device type** are Intel and x86-64 (Windows64), respectively.
- If Simulink Coder is installed, the revised **Hardware Implementation** pane identifies the system target file that you selected on the **Code Generation** pane.
- A **Device details** option provides a way to display parameters for setting details such as number of bits and byte ordering.
- To specify target hardware for a Simulink support package, select a value from **Configuration Parameters > Hardware Implementation > Hardware board**. Before R2015b, you selected **Tools > Run on Target Hardware > Prepare to run**. Then, you selected a value from **Configuration Parameters > Run on Target Hardware > Target hardware**.
- To specify target hardware for an Embedded Coder support package, select a value from **Configuration Parameters > Hardware Implementation > Hardware board**. Before R2015b, you selected a value from **Configuration Parameters > Code Generation > Target hardware**.

- The **Test hardware** section was removed. Configure test hardware from the Configuration Parameters list view. Set `ProdEqTarget` to `off`, which enables parameters for configuring test hardware details.
- If you set **Configuration Parameters > Code Generation > System target file** to `ert.tlc`, `realtime.tlc`, or `autosar.tlc`, the default setting for **Configuration Parameters > Hardware Implementation > Hardware board** is `None`. If you set **System target file** to value other than `ert.tlc`, `autosar.tlc`, or `realtime.tlc`, the default setting for **Hardware board** is `Determine by Code Generation system target file`.

For more information, see Hardware Implementation Pane.

Version History

Starting in R2015b:

- By default, the **Hardware Implementation** pane lists **Hardware board**, **Device vendor**, and **Device type** parameter fields only. To view parameters for setting details, such as number of bits and byte ordering, click **Device details**.
- The following devices appear on the **Hardware Implementation** pane only for models that you create with a version of the software earlier than R2015b. These devices are considered legacy devices.
 - Generic, 32-bit Embedded Processor
 - Generic, 64-bit Embedded Processor (LP64)
 - Generic, 64-bit Embedded Processor (LLP64)
 - Generic, 16-bit Embedded Processor
 - Generic, 8-bit Embedded Processor
 - Generic, 32-bit Real-Time Simulator
 - Generic, 32-bit x86 compatible
 - Intel, 8051 Compatible
 - Intel, x86-64
 - SGI, UltraSPARC Iii

In R2015b, if you open a model configured for a legacy device and change the **Device type** setting, you cannot select the legacy device again.

- Device parameter **Signed integer division rounds to** is set to `Zero` instead of `Undefined`. For some cases, numerical differences can occur in results produced with `Zero` versus `Undefined` for simulation and code generation.

This change does not apply to legacy devices.

- To associate a new model with an existing configuration set that has the following characteristics, configure the model to use the same hardware device as the existing model.
 - The model consists of a model reference hierarchy. Models in the hierarchy use different configuration sets.
 - The existing configuration set was saved as a script and associated with a configuration set variable.

If the code generator detects differences in device parameter settings, a consistency error occurs. To correct the condition, look for differences in the device parameter settings, and make the appropriate adjustments.

Signal Management

Virtual bus signal inputs to blocks that require nonbus or nonvirtual bus input

Starting in R2015b, virtual bus signal input to blocks that require nonbus or nonvirtual bus input can cause an error. The error occurs when a virtual bus input signal is generated by a block that specifies a bus object as its output data type. Examples of blocks that can specify a bus object as its output data type include a Bus Creator block or a root Inport block. The blocks that cause an error when they have a virtual bus input in this situation are:

- Assignment
- Delay

This block causes an error only if you set an initial condition from the dialog that is a MATLAB structure or zero and you specify a value for **State name**.

- Permute Dimension
- Reshape
- Selector
- Unit Delay

This block causes an error only if you set an initial condition from the dialog that is a MATLAB structure or zero and you specify a value for **State name**.

- Vector Concatenate

Generating an error when this situation occurs helps to promote consistent output by requiring proper input to these blocks.

To check for proper virtual bus usage and for Mux blocks used to create bus signals, use the new Upgrade Advisor **Check bus usage** check.

The new Upgrade Advisor check inserts a Bus to Vector block to attempt to convert virtual bus input signals to vector signals. For issues that the Upgrade Advisor identifies but cannot fix, modify the model manually. For details, see [Correct Buses Used as Muxes and Prevent Bus and Mux Mixtures](#).

Version History

In R2015b, a virtual bus input signal to the affected blocks causes an error message, regardless of the **Configuration Parameters > Diagnostics > Connectivity > Bus signal treated as vector** setting.

In models created in releases earlier than R2015b, if the **Bus signal treated as vector** diagnostic setting is **error**, there is no compatibility issue when you run the model in R2015b.

To help you to address this issue, use the Upgrade Advisor **Check bus usage** check.

Entire nested bus assignment for Bus Assignment block

If a bus signal input to a Bus Assignment block contains a nested bus signal, then you can assign or select the nested bus signal as a whole. However, the nested bus cannot be nested inside of an array of buses. Before R2015b, you had to assign each signal in the nested bus.

Support for entire nested bus signal assignment reduces:

- The number of blocks in a diagram
- The maintenance effort

For details, see [Bus Assignment](#).

Block Enhancements

Waveform Generator Block: Define and output arbitrary waveform signals

The Waveform Generator block outputs waveforms using signal notations. This block is located in the Sources sublibrary.

From Spreadsheet Block: Read signal data into Simulink from a spreadsheet

The From Spreadsheet block reads data from spreadsheets. This block is located in the Simulink Extras/Additional Sources sublibrary.

MATLAB System block support for nonvirtual buses

The MATLAB System block now supports nonvirtual buses. For more information, see Nonvirtual Buses and MATLAB System Block. For an example, see Using Buses with MATLAB System Blocks.

Inport block update

The Inport block now has a **Connect Input** button. Use this button to import, visualize, and map signal and bus data to root-level inports using the Root Inport Mapping tool.

From File updates for file name and signal preview

The From File block **File name** parameter has been updated to include:

- A file browse button
- A view button that lets you plot and inspect signals.

Inheriting of continuous sample time for discrete blocks

The **Configuration Parameters > Diagnostics > Sample Time > Discrete used as continuous** diagnostic was removed. If a discrete block (such as the Unit Delay block) inherits a continuous sample time, the block returns an error.

Evenly spaced breakpoints in Lookup Tables

A new format is available to specify evenly spaced breakpoints in the Prelookup and n-D Lookup Table blocks. For more information, see the **Specification** parameter in the Prelookup block reference page and the **Breakpoints specification** parameter in the n-D Lookup Table reference page.

Integrator block: Wrapped states for modeling rotary and cyclic state trajectories

The Integrator block has been enhanced to support wrapped states when modeling rotary, cyclic, or periodic state trajectories. This support for wrapping states provides these advantages.

- It eliminates simulation instability when your model approaches large angles and large state values.
- It reduces the number of solver resets during simulation and eliminates the need for zero-crossing detection, improving simulation time.
- It eliminates large angle values, speeding up computation of trigonometric functions on angular states.
- It improves solver accuracy and performance and enables unlimited simulation time.

Variant Subsystem block: Enhanced option for generating preprocessor conditionals

The option **Generate preprocessor conditionals** in the Variant Subsystem block parameters dialog box has been replaced with the option **Analyze all choices during update diagram and generate preprocessor conditionals**. When you select this option, Simulink analyzes all variant choices during an update diagram or simulation. This analysis provides early validation of the code generation readiness of all variant choices.

Version History

Previously, when the option to generate preprocessor conditionals was switched on, Simulink analyzed all variant choices only during the code generation phase. Now, Simulink performs this analysis during the update diagram phase. As a result, errors that you would normally see during code generation appear earlier, during an update diagram.

Constant sample time in S-function blocks

MATLAB S-function blocks no longer support a constant sample time (Inf) for their ports by default.

Version History

To allow ports in your MATLAB S-function blocks to have a sample time of Inf, use the `SetAllowConstantSampleTime` command.

MATLAB Function Blocks

Calling of Simulink Functions

You can call a Simulink Function block from inside of a MATLAB Function block. You can also call Stateflow functions with **Export Chart Level Functions (Make Global)** and **Allow exported functions to be called by Simulink** checked in the chart Properties dialog box.

Nondirect feedthrough in MATLAB Function blocks

By default, MATLAB Function blocks have direct feedthrough enabled. To disable, in the Ports and Data Manager, you can now clear the **Allow direct feedthrough** check box. Nondirect feedthrough enables semantics to ensure that outputs rely only on current state.

To use nondirect feedthrough, do not program outputs to rely on inputs or updated persistent variables. For example, do not use the following code in a nondirect feedthrough block:

```
counter = counter + 1;      % update state
output = counter;          % compute output based on updated state
```

Instead, use code such as:

```
output = counter;          % compute output based on current state
counter = counter + 1;     % update state
```

Also, nondirect feedthrough semantics require function inlining. Do not disable inlining.

Using nondirect feedthrough enables you to use MATLAB Function blocks in a feedback loop and prevent algebraic loops.

Overflow and data range detection settings unified with Simulink

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

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

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

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

See Diagnostics Pane: Data Validity.

Version History

When you open a model with a MATLAB Function block saved in a previous release, a change in behavior is possible. These options are no longer valid.

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

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

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

Set each configuration parameter to `none`, `warning`, or `error`. The software displays a warning if the previous MATLAB Function block options saved with the model were set differently than the current Simulink options.

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

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

No frame-based sampling mode for outputs

In R2015b, you can no longer set the sampling mode of outputs to frame based in MATLAB Function blocks. Models created in previous releases using frame-based sampling mode continue to behave as they did before this release.

Code generation for cell arrays

In R2015b, you can generate code from MATLAB code that uses cell arrays.

The code generation software classifies a cell array as homogeneous or heterogeneous. This classification determines how a cell array is represented in the generated C/C++ code. It also determines how you can use the cell array in MATLAB code from which you generate C/C++ code. See [Homogeneous vs. Heterogeneous Cell Arrays](#).

As long as you do not specify conflicting requirements, you can control whether a cell array is homogeneous or heterogeneous. See [Control Whether a Cell Array is Homogeneous or Heterogeneous](#).

For information about restrictions when you use cell arrays in a MATLAB Function block, see [Cell Array Requirements and Limitations for Code Generation](#).

LAPACK calls during simulation for algorithms that call linear algebra functions

To improve the simulation speed for MATLAB Function block algorithms that call linear algebra functions, the simulation software can now call LAPACK functions. If the input arrays for the linear algebra functions meet certain criteria, the simulation software generates calls to relevant LAPACK functions.

LAPACK is a software library for numerical linear algebra. MATLAB uses this library in some linear algebra functions such as `eig` and `svd`. The simulation software uses the LAPACK library that is included with MATLAB.

For information about the open source reference version of LAPACK, see [LAPACK — Linear Algebra PACKage](#).

Code generation for additional Image Processing Toolbox and Computer Vision System Toolbox functions

Image Processing Toolbox

<code>bwareaopen</code>	<code>houghpeaks</code>	<code>immse</code>	<code>integralBoxFilter</code>
<code>grayconnected</code>	<code>imabsdiff</code>	<code>imresize</code>	<code>psnr</code>
<code>hough</code>	<code>imcrop</code>	<code>imrotate</code>	
<code>houghlines</code>	<code>imgaborfilt</code>	<code>imtranslate</code>	

See Image Processing Toolbox in [Functions and Objects Supported for C and C++ Code Generation — Category List](#).

Computer Vision System Toolbox

- `insertText`
- `extractLBPFeatures`

See Computer Vision System Toolbox in [Functions and Objects Supported for C and C++ Code Generation — Category List](#).

Code generation for additional Statistics and Machine Learning Toolbox functions

- `kmeans`
- `randsample`

See Statistics and Machine Learning Toolbox in [Functions and Objects Supported for C and C++ Code Generation — Category List](#).

Code generation for additional MATLAB functions

Data Types in MATLAB

- `cell`
- `fieldnames`
- `struct2cell`

See Data Types in MATLAB in Functions and Objects Supported for C and C++ Code Generation — Category List.

String Functions in MATLAB

- `iscellstr`
- `strjoin`

See String Functions in MATLAB in Functions and Objects Supported for C and C++ Code Generation — Category List.

Code generation for additional Communications System Toolbox, DSP System Toolbox, and Phased Array System Toolbox functions and System objects

Communications System Toolbox

`comm.CoarseFrequencyCompensator`

See Communications System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

DSP System Toolbox

- `dsp.IIRHalfbandDecimator`
- `dsp.IIRHalfbandInterpolator`
- `dsp.AllpassFilter`

See DSP System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

Phased Array System Toolbox

- `phased.TwoRayChannel`
- `phased.GCCEstimator`
- `phased.WidebandRadiator`
- `phased.SubbandMVDRBeamformer`
- `phased.WidebandFreeSpace`
- `gccphat`

See Phased Array System Toolbox in Functions and Objects Supported for C and C++ Code Generation — Category List.

R2015a

Version: 8.5

New Features

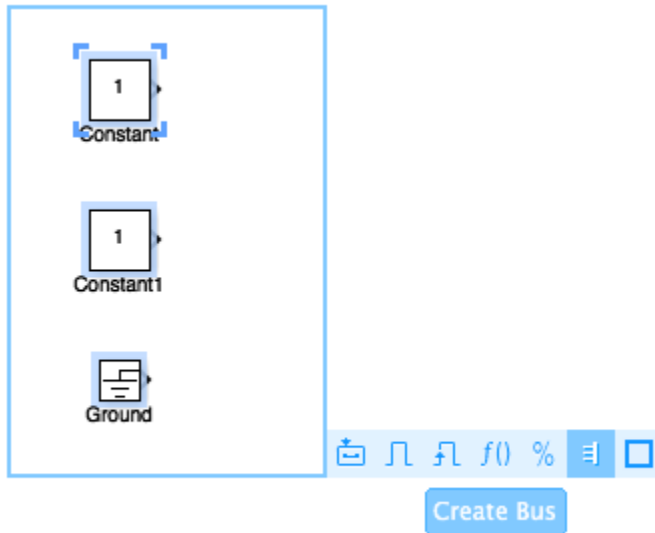
Bug Fixes

Version History

Simulink Editor

Bus Smart Editing Cue: Automatically create a bus from a set of signals

You can select multiple blocks or signals in an area of a model to create a bus for.



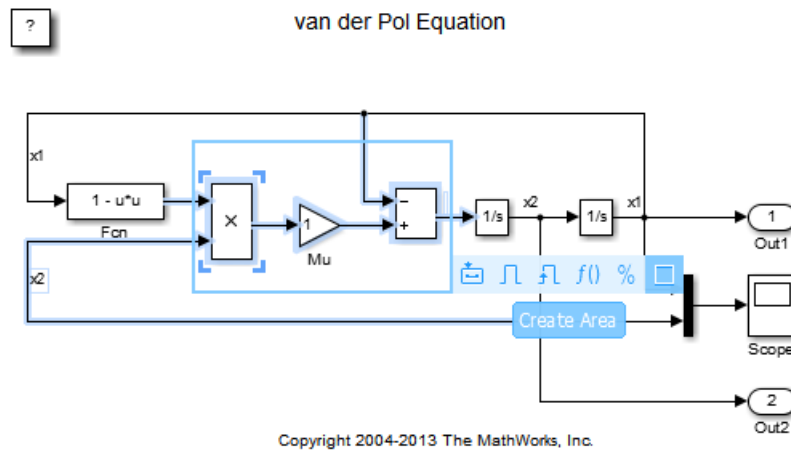
Using the **Create Bus** button to create a bus automatically performs these actions:

- Creates a Bus Creator block with the right number of inputs
- Resizes the Bus Creator block to an appropriate size for the number of input signals
- Orients the Bus Creator block to the direction of the input signals
- Connects the signals to the Bus Creator block

Area Annotations: Call out and separate regions of interest in model

You can select multiple objects in an area of a model to annotate an area of interest in your model.

- 1 In the Simulink Editor, drag to select the area of the model you want to create an area around.
- 2 In the action bar, click **Create Area**.

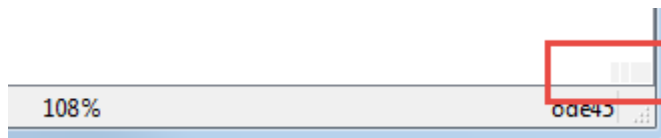


For more information, see [Box and Label Areas of a Model](#).

Perspectives Controls: Access alternative views of your model, such as harness and interface views

You can switch to different views of a model, such as harness view to view test harnesses and interface view to view interfaces of your system.

- 1 In the Simulink Editor, click the control in the lower-right corner.



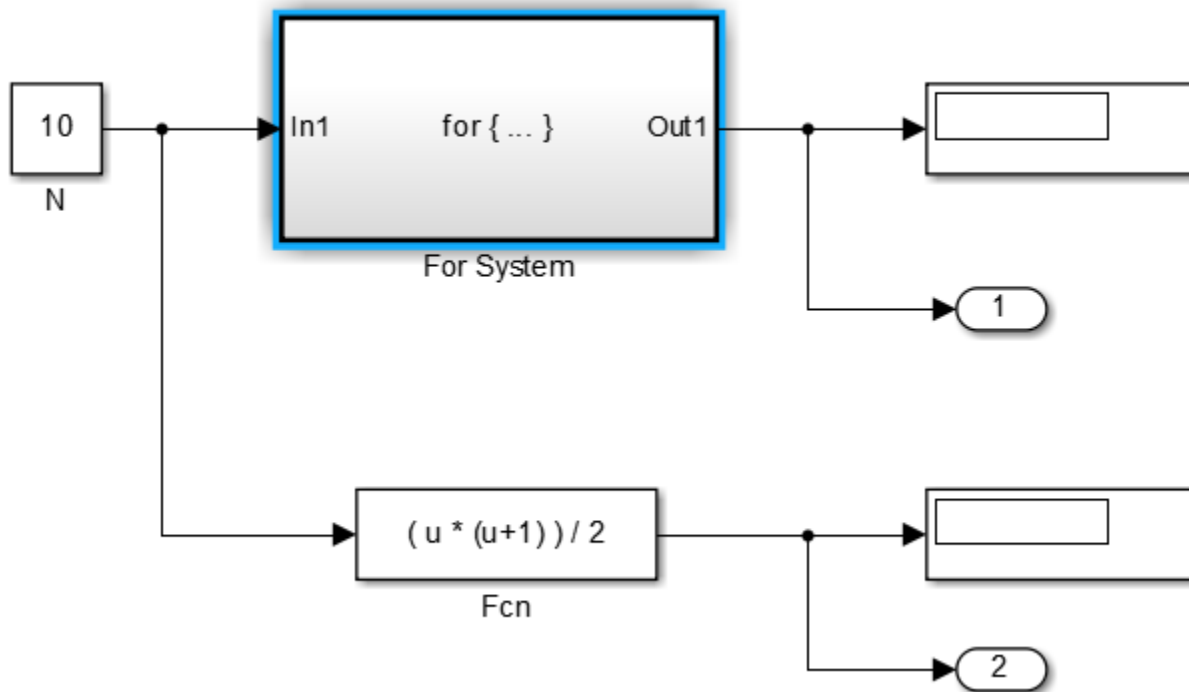
- 2 Click the view you want to see.

Saving of viewmarks in Simulink models

You can now store viewmarks with your model. For more information, see [Use Viewmarks to Save Views of Models](#).

Highlighting of the subsystem you navigated from

When you navigate out of a subsystem, Simulink now highlights that subsystem temporarily. The highlight helps you to identify the subsystem you were working in most recently.



Annotation connector colors and width

You can now specify the color and width of annotation connectors. Right-click the annotation connector and use the **Format** menu.

To create an annotation connector, see [Add Lines to Connect Annotations to Blocks](#)

Undo and redo of block parameter value changes

Block parameter changes using block dialog boxes are now undoable. When you use **Undo** after you make changes to block parameter values, the values revert and the affected block briefly appears highlighted. A **Redo** command as the next action restores the changes and also briefly highlights the block.

Display of product name in model title bar

The Simulink Editor now displays the word **Simulink** next to the model name in the model title bar.

Simulation Analysis and Performance

Dashboard Block Library: Tune and test simulations with graphical controls and displays

The Dashboard block library contains controls and displays that enable you to tune block parameters or monitor signals. The controls can be active during simulation so you can tune parameters and optimize your model while it is running or paused. The blocks are added directly to your Simulink model canvas and connected to signals or blocks. The Dashboard block library includes:

- Knobs
- Switches
- Gauges
- Dashboard scope
- Lamp

Algebraic Loop Highlighting: Find and remove algebraic loops in the model to boost simulation speed

You can now identify and remove all algebraic loops in your model using the new highlight feature. This option traverses through the hierarchy of a model to highlight real and artificial algebraic loops. A report lists the loops with color codes for easy isolation. This technique allows you to view all the loops in a model simultaneously, so you can remove them quickly and speed up your simulation.

For more information, see [Highlight Algebraic Loops in the Model](#).

Faster Simulations with Accelerated Referenced Models: Run faster consecutive simulations and step back and forth through simulations

Simulating referenced models in Accelerator mode now supports these actions that involve using `SimState`:

- Do a fast restart
- Save complete `SimState` in final state logging
- Restart from saved state
- Step through a simulation

If the referenced model simulated in Accelerator mode contains any of these blocks, then you cannot perform the actions listed above:

- Level 2 MATLAB S-Function
- MATLAB System
- n-D Lookup Table
- S-Function (with custom `SimState` or `PWork` vectors)
- To File
- Simscape blocks

Use SimulationMetadata to retrieve simulation metadata information

You can store and retrieve metadata about a simulation using `SimulationMetadata`. The `SimulationOutput` object of the simulation contains the metadata object. Use `SimulationMetadata` to analyze archived simulations or compare results from multiple simulations. See `Simulink.SimulationMetadata` for more details.

Simplified conversion of logged data to Dataset format for a common logged data format

You can use the `Simulink.SimulationData.Dataset` constructor to convert data that was logged in one of the following formats to `Dataset` format:

- Array
- Structure
- Structure with time
- MATLAB timeseries
- `ModelDataLogs`

Converting data from other Simulink logging formats to `Dataset` format simplifies writing scripts to post-process data logged using different:

- Logging techniques (for example, for models with multiple To Workspace blocks using different data formats or for models that use To Workspace blocks and log signals using `ModelDataLogs` format)
- Simulation modes (for example, Normal and Accelerator)

The conversion to `Dataset` format also makes it easier to take advantage of features that require `Dataset` format. Now you can easily convert data logged in earlier releases that used a format other than `Dataset` to work well with `Dataset` data in a more recent release.

You can use the `Simulink.SimulationData.Dataset.concat` method to combine `Dataset` objects into one concatenated `Dataset` object.

For more information, see `Simulink.SimulationData.Dataset`.

Default setting for Automatic solver parameter selection

The default setting for **Automatic solver parameter selection** in the **Diagnostics** pane of the Model Configuration Parameters dialog box is now set to none. Previously, the default setting for this option was warning.

Improved heuristic for step size calculation

Simulink now uses an improved heuristic to calculate the **Max step size** and **Fixed step size** when you set these parameters to auto in the **Solver** pane of the Model Configuration Parameters dialog box. The heuristic uses the dynamics of a model in a better way to calculate these parameters. For more information, see `Max step size`.

Step size details in solver information tooltip

When you set **Max step size** and **Fixed step size** to auto in the **Solver** pane of the Model Configuration Parameters dialog box, you can see the numerical values of these settings in the solver information tooltip. Previously, Simulink displayed this information as a warning message. For more information, see Solver Overview.

Solver information in model after simulation

Starting in R2015a, after a model compiles, the right hand corner of the status bar displays the solver used to compile a model. Previously, Simulink reverted to the configuration setting after a simulation completed. For more information, see Solver Overview.

Component-Based Modeling

Consistent Data Support for Testing Components: Load input and log data of a component from buses and all data types

Logging states and root outports now supports full logging capabilities in Normal and Accelerator (including Model Reference Accelerator) simulation modes. State and root outport logging in R2015a:

- Supports all data types, including:
 - Bus data (virtual and nonvirtual buses and arrays of buses)
 - Enumerated data
 - Fixed-point data
- Logs states and outports based on their rates

Simulink supports `Dataset` objects for root inport data import, which eliminates the need to specify multiple inputs using a comma-separated list.

Model Reference Conversion Advisor enhancements

In R2015a, the Model Reference Conversion Advisor make the conversion of a subsystem to a referenced model easier than in previous releases.

- You can have the advisor compare the results of simulating the top model for the referenced model to the results of simulating the baseline model that has the subsystem. In the **Check conversion input parameters** check, select **Check simulation results after conversion**.

You can specify the following for the comparison of the simulations:

- Stop time for the simulation
- Absolute signal tolerance
- Relative signal tolerance
- In the **Check conversion input parameters** check, you can specify the simulation mode for the Model block that references the referenced model.
- The conversion process stores all of the data it creates (bus objects, signal objects, and tunable parameters) in one file. To specify the file for the advisor to use, in the **Check conversion input parameters** check, use the **Conversion data file name** option.
- The reports for the checks include links that highlight the relevant blocks in the model.
- Restoring the model to the pre-conversion state is faster than in previous releases. Also, the report for the **Complete conversion** check now includes a link to restore the model. That option provides an alternative to selecting **File > Load Restore Point** in the advisor.

For details, see [Convert a Subsystem to a Referenced Model](#).

The `Simulink.SubSystem.convertToModelReference` function supports the Model Reference Conversion Advisor capabilities.

Reduced algebraic loops during model reference simulation

Model reference simulation now addresses some cases that previously resulted in algebraic loops.

Multi-instance support for nonreusable functions in referenced models

Simulation of nonreusable functions in referenced models no longer forces you to set the **Configuration Parameters > Model Referencing > Total number of instances allowed per top model** parameter to One. You can also set the parameter to Multiple.

Model referencing checks in Model Advisor to reduce warning messages

The Model Advisor includes two new checks to reduce the number of warning messages during simulation and code generation. These checks replace the **Configuration Parameters > Diagnostics > Model Referencing > Model configuration mismatch** diagnostic, which generated many warning messages that you could safely ignore.

- The **Check diagnostic settings ignored during accelerated model reference simulation** check lists models referenced in Accelerator mode for which you set certain runtime diagnostics to a value other than none or Disable all.

Simulink ignores the diagnostics that the check lists. For more information, see [Certain Diagnostic Configuration Parameters Ignored for Models Referenced in Accelerator Mode](#).

- The **Check code generation identifier formats used for model reference** check lists referenced models with certain parameters whose settings do not contain a \$R token (which represents the name of the reference model). Code generation prepends the \$R token (if not present) to the identifier format. For more information, see [Configuration Parameters Changed During Code Generation](#).

Version History

The **Model configuration mismatch** diagnostic no longer appears in the **Configuration Parameters > Diagnostics > Model Referencing** pane. Simulink ignores the setting of the corresponding `ModelReferenceCSMismatchMessage` parameter.

- When loading a configuration set from a MAT-file or a model, if the parameter setting is either error or warning, the setting is not honored and remains none.
- When you save a configuration set to a MAT-file or save a model to disk, the parameter is not saved.
- When you export a model to a previous version, the parameter is written to disk with a setting of none.
- If you use `get_param` with this parameter on a configuration set or on a model, the returned value is always the default value (none).
- If you use a `set_param` command for this parameter on a configuration set or on a model with a setting other than none, Simulink ignores the setting.

Model configuration parameter changes

The following configuration parameters are no longer supported and are not saved in the model configuration:

- CodeGenDirectory (Simulink Coder)
- ConfigAtBuild (Simulink Coder)
- ConfigurationMode (Simulink Coder)
- ConfigurationScript (Simulink Coder)
- CustomRebuildMode (Simulink Coder)
- DataInitializer (Simulink Coder)
- Echo
- EnableOverflowDetection
- FoldNonRolledExpr
- GenerateClassInterface(Simulink Coder)
- GenerateCodeInfo (Simulink Coder)
- IncludeERTFirstTime (Simulink Coder)
- InitialValueSource (Embedded Coder)
- MisraCompliance (Embedded Coder)
- ModuleName (Embedded Coder)
- ModuleNamingRule (Embedded Coder)
- ProcessScript (Simulink Coder)
- ProcessScriptMode (Simulink Coder)
- SimBlas
- SimDataInitializer
- SimExtrinsic
- TargetTypeEmulationWarnSuppressLevel
- UseTempVars

In R2014b, these configuration parameters were command-line only parameters and were not available in the Configuration Parameters dialog box.

Version History

- When you save a configuration set to a MAT-file or save a model to disk, these parameters are NOT saved.
- If you use a `set_param` command with one of these parameters on a configuration set or on a model, the value of the parameter is only changed while the model is loaded in the current MATLAB session.
- If you use a `get_param` command with one of these parameters on a configuration set or on a model, the behavior is unchanged.

Property name change in Simulink.ConfigSetRef

The `WSVarName` property of the `Simulink.ConfigSetRef` object has been renamed to `SourceName`. The `SourceName` property specifies the name of the variable in the workspace or the data dictionary that contains the referenced configuration set. The `WSVarName` will be removed in a future release.

Version History

To avoid future incompatibility, change instances of this property name to the new name.

Flexible structure assignment of buses

When a non-tunable structure is assigned to a bus signal (such as a block which uses a structure for its initial condition parameter), the data types of the fields of the structure no longer need to match the data types of the bus elements. The software now performs an automatic casting of the data type of the structure field so that it matches the data type of the bus signal.

Support for empty subsystems as variant choices

Previously, if you added an empty subsystem with no inputs or outputs as a variant choice inside a Variant Subsystem block, Simulink discarded the empty subsystem. This is because empty variant choices were considered invalid. Moreover, if the Variant Subsystem block contained a valid variant choice and an empty variant choice, no preprocessor conditionals were generated for the valid choice when you built the model.

In R2015a, Simulink considers an empty subsystem as a valid variant choice. If you add an empty variant choice inside a Variant Subsystem block, specify a variant condition for this choice in one of the following ways.

- Specify a variant activation condition for the empty choice. During simulation, if the empty variant choice is active, Simulink ignores the empty choice.
- Comment out the variant activation condition by placing a `%` symbol before the condition.

Moreover, if the Variant Subsystem block contains a valid variant choice and an empty variant choice, preprocessor conditionals are now generated for the valid choice when you build the model.

For an example, see [Define, Configure, and Activate Variant Choices](#).

Conversion of MATLAB variables used in variant control expressions into Simulink.Parameter objects

MATLAB variables allow you to rapidly prototype variant control expressions when you are building your model. Previously, if you wanted to generate preprocessor conditionals for code generation, you had to manually convert these variables into `Simulink.Parameter` objects.

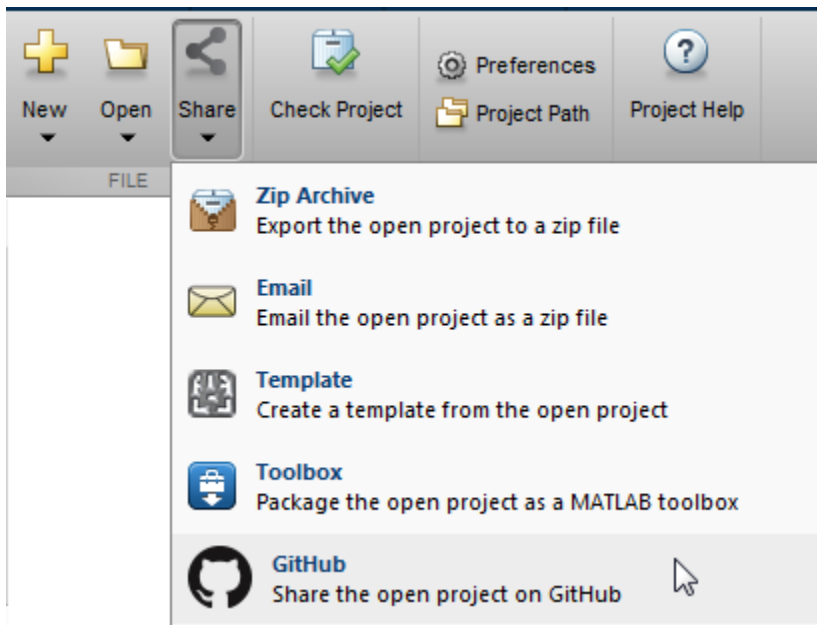
In R2015a, use the function `Simulink.VariantManager.findVariantControlVars` to find and convert MATLAB variables used in variant control expressions into `Simulink.Parameter` objects. For an example, see [Convert Variant Control Variables into Simulink.Parameter Objects](#).

Project and File Management

Simulink Project Sharing: Share a project using GitHub, email, or a MATLAB toolbox

Release 2015a provides new options for sharing Simulink projects:

- Make your project publicly available on GitHub®.
- Share your project via email.
- Package your project as a MATLAB toolbox.



For details, see [Sharing Simulink Projects](#).

Interactively manage the MATLAB search path for your project

You can interactively add or remove folders from the project path. With Simulink Project, opening your project adds the project path to the MATLAB search path. Closing your project removes the project path from the MATLAB search path. For more information, see [Specify Project Path](#).

Easy viewing and editing of project labels

R2015a provides easier management of label data through the improved file details view.

Name	Status	SVN	Revision	Type	Classification	Engineers	Reviewer
batch_jobs	✓	●	2	Folder	None		
data	✓	●	2	Folder	None		
models	✓	●	2	Folder	None		
AnalogControl.mdl	✓	●	2	Simulink Model	Design		
DigitalControl.slx	✓	●	2	Simulink Model	Design		
f14_airframe.slx	✓	●	2	Simulink Model	Artifact	Bob: Please assess	Pam: Perform level 1
LinearActuator.slx	✓	●	2	Simulink Model	Design		
NonLinearActuator.mdl	✓	●	2	Simulink Model	Design	Tom	
slproject_f14.slx	✓	●	2	Simulink Model	Design		
vertical_channel.slx	✓	●	2	Simulink Model	Design		
wind_gust_lib.slx	✓	●	2	Simulink Library	Design		
reports	✓	●	2	Folder	None		
src	✓	●	2	Folder	None		
tests	✓	●	2	Folder	None		
utilities	✓	●	2	Folder	None		
work	✓	○	2	Folder	None		

Model version: 1.25
 Saved in Simulink version: R2015a
 Last modified by: The MathWorks Inc.
 (no description available)

Preview:

Engineers: Bob, Please assess, Tom, Check feasibility. Please report findings at next meeting.

Reviewer: Pam, Perform level 1

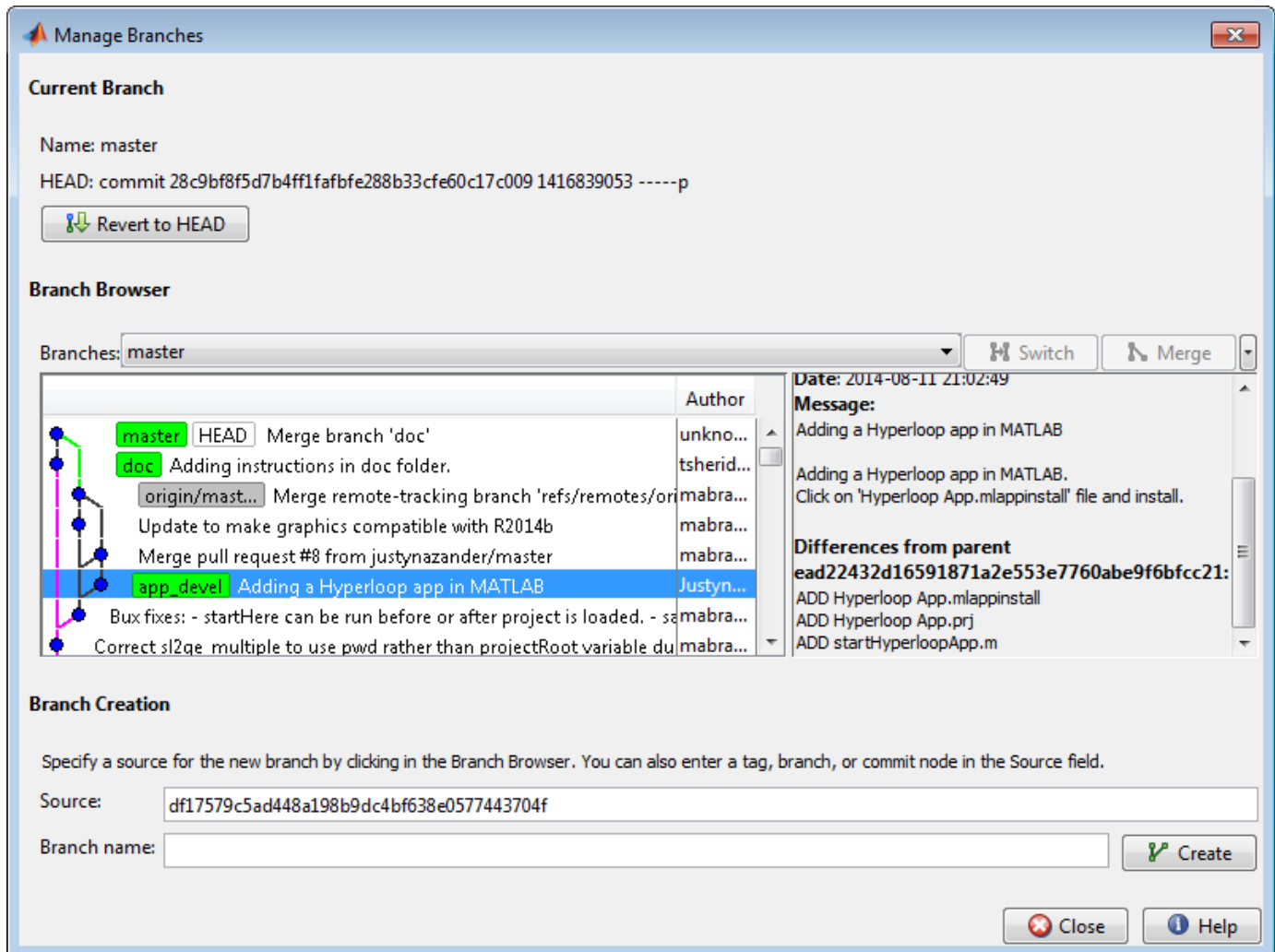
Classification: Artifact

For more information, see:

- Create Labels
- Add Labels to Files
- View and Edit Label Data

Changed file lists and branch deletion in Git Manage Branches dialog box

In R2015a, when using Git source control, you can view branch details and delete branches in the Manage Branches dialog box. You can see which files changed for a particular commit in the branch viewer, and view the author, date, and commit message.



For details, see Branch and Merge Files with Git.

New preferences to control loading and saving models

Simulink has a new preference, **Do not load models that are shadowed on the MATLAB path**. Use the preference to specify whether to load a model that is shadowed by another file of the same name higher on the MATLAB path. If you turn the preference on by selecting the check box, then Simulink displays an error when you try to load a shadowed model. For details, see Do not load models that are shadowed on the MATLAB path.

Another preference, **Do not load models created with a newer version of Simulink**, is now on by default. These preferences help you avoid accidentally loading or editing the wrong model or library.

You can use a new preference, **Save a thumbnail image inside SLX files** to control whether to save a small screen shot image of the model. You can view the screenshot for a selected model in the Current Folder browser preview pane. If your model is very large and you want to reduce the time taken to save the model, then you can turn this preference off to avoid saving thumbnail model images.

Version History

The preference **Do not load models created with a newer version of Simulink** is now on by default. Simulink does not load the model and displays an error message in the Command Window. In previous releases, the preference was off by default, so Simulink loaded models last saved in a newer version and displayed a warning message.

Improved error reporting from `get_param`, `set_param` and `save_system`

The functions `get_param`, `set_param` and `save_system` have improved error reporting to help you correct problems specifying the correct model or block. The new messages tell you if the model is not loaded, the model name is not valid, or the block is not found in the specified model.

Model dependency analysis option to find enumeration definition files

In R2015a, model dependency analysis provides an option to search for enumeration definition files. Turn on this option to detect enumerated data types used as part of a bus object definition. You can then include the files when exporting the model and required files into a zip file.

The option is not on by default because it requires a model update. You cannot perform this search with project impact analysis, because the search requires model updates. Instead, select **Analysis > Model Dependencies > Generate Manifest**, and turn on the **Find enumeration definition files (performs a Model Update)** option.

For details, see [Generate Manifests](#).

Export to previous version supports seven years

In R2015a, in the Export to Previous Version dialog box, the **Save as type** list includes seven years of previous releases. In future releases, this list will continue to provide seven years of previous releases.

In previous releases, the list included versions back to R14 (2004). If you want to export to older versions than seven years, you can use the `save_system` function instead. For details, see [save_system](#).

Data Management

Data Dictionary API: Automate the creation and editing of data dictionaries with MATLAB scripts

You can now manage data dictionaries and interact with dictionary content at the MATLAB command prompt in addition to the Model Explorer. For example, you can:

- Migrate models to use data dictionaries
- Import data from and export data to external files and the MATLAB base workspace
- Create, delete, and reassign entries
- Save and discard changes to entire dictionaries or discard changes to individual entries
- Search for specific entries
- Compose reference dictionary hierarchies

See *Store Data in Dictionary Programmatically* for examples and a list of relevant functions and classes.

Rename All: Change the name of a parameter and all its references

You can now use Model Explorer to rename a variable everywhere it is used by blocks in a Simulink model.

See *Rename Variables* for an example.

MATLAB Editor features for editing model workspace code

When you view the dialog box for a model workspace, for example using the property dialog pane of Model Explorer, and set **Data source** to **MATLAB Code**, the code editing area now behaves like the MATLAB Editor. For example, the editing area applies syntax highlighting to your code.

Management of variables from block dialog box fields

You can now navigate to and edit variables by right-clicking expressions that you specify in block dialog boxes. For each variable in an expression, you can navigate to the workspace that defines the variable or open a separate dialog box to edit the variable.

You can also create variables by right-clicking an expression that contains the names of potential variables. For each name you specify in the expression, you can create a variable in a workspace that is appropriate for the model.

See *Manage Variables from Block Parameter* for more information.

Other Data section added to data dictionary

To store reference data that are not used by a model for simulation, but that are still relevant to the model, use the Other Data section of a data dictionary. For example, if a model simulates the behavior of mechanical equipment, you can store the manufacturer specifications in the Other Data section.

You can also store objects of any MATLAB or Simulink class, including custom classes, in the Other Data section of a dictionary. By contrast, the Design Data section stores only objects relevant to simulation of a model.

Data dictionaries now have a Design Data section, a Configurations section, and an Other Data section. Prior to R2015a, the Design Data section was named Global Design Data.

Model-wide renaming of data stores

You can now rename a data store everywhere it is used by Data Store Read and Data Store Write blocks in a Simulink model. Previously, you manually updated all Data Store Read and Data Store Write blocks with the new name of the data store.

See [Rename Data Stores](#) for more information.

Reporting of enumerated types used by model

You can now use the existing function `Simulink.findVars` to discover the enumerated data types that are needed by a model. `Simulink.findVars` can report the names of enumerated types that are used to define model variables.

You can enable the reporting of enumerated types using the name-value pair `IncludeEnumTypes`. With this name-value pair enabled, `Simulink.findVars` returns a `Simulink.VariableUsage` object for each enumerated type in addition to the `Simulink.VariableUsage` objects returned for model variables.

Root Inport Mapping tool updates

The Root Inport Mapping tool now supports:

- Models configured for Fast Restart.
- Importing and mapping of Simulink Design Verifier test vectors. For more information, see [Import Test Vectors from Simulink Design Verifier Environment](#).

Connection to Educational Hardware

Simulink Support Package for Apple iOS Devices: Create an App that runs Simulink models and algorithms on your Apple iOS device

You can run Simulink models on Apple iOS devices. You can also tune parameter values in the model, and receive data from the model, while it is running on these devices.

Use the **Simulink Support Package for Apple iOS Devices** block library to access the Apple iOS hardware:

- Accelerometer
- Audio Capture
- Audio Playback
- Camera
- Display
- FromApp
- Gyroscope
- Location Sensor
- ToApp
- UDP Receive
- UDP Send

To install or update this support package, perform the steps described in [Install Support for Apple iOS Devices](#).

MathWorks response to the Shellshock vulnerability

The support package for BeagleBoard (v14.2) contains a vulnerable version of the Bash shell. The default configuration of this support package does not expose this vulnerability to a network connection. We recommend running this device inside of a trusted network or behind a firewall. We will update this package as updated software is available.

For more information, see <https://mathworks.com/matlabcentral/answers/158586-what-is-mathworks-response-to-the-shellshock-vulnerability>.

Removed support for Gumstix Overo and PandaBoard hardware

The following support packages have been removed and are no longer available:

- Simulink Support Package for Gumstix® Overo® Hardware
- Simulink Support Package for PandaBoard Hardware

Signal Management

Array of buses with Unit Delay block

You can use an array of buses as an input signal to a Unit Delay block.

Block Enhancements

Resettable Subsystem block to reset the subsystem states

The Resettable Subsystem block is a new block in the Ports & Subsystems library. Use this subsystem to reset the states of all blocks inside the subsystem on triggering. For more information, see Resettable Subsystem.

Conditional display of the Sample Time parameter

The Sample Time parameter in the dialog box of certain Simulink blocks is now hidden by default. If you set the sample time to a value other the default (at the command line or if it is set that way in an existing model), then the parameter is visible. For more information, see Blocks for Which Sample Time Is Not Recommended.

Inheritance of frame-based input returns error

Note This release note applies only if you have installed DSP System Toolbox.

As part of general product-wide changes pertaining to frame-based processing, certain block options that use the frame attribute of the input signal now cause an error in Simulink. For more information on changes in the DSP System Toolbox, see Frame-based processing.

The following sections provide more detailed information about the specific R2015a Simulink software changes for frame-based processing:

- “Input Processing Parameter Set to Inherited” on page 16-20
- “Inherited Setting on Save 2-D Signals” on page 16-21
- “Frame-based Inputs Removed for Bias Block” on page 16-22
- “Frame-based Inputs Removed for Tapped Delay Block” on page 16-22
- “Frame-based Input Removed for Transfer Fcn First Order Block” on page 16-22
- “Frame-based Input Removed for Transfer Fcn Lead or Lag Block” on page 16-22
- “Sampling Mode Set to Frame-based” on page 16-23

Input Processing Parameter Set to Inherited

Setting Input processing parameter to Inherited now errors for these blocks:

- Relay
- Backlash
- Difference
- Delay
- Unit Delay
- Variable Integer Delay
- Discrete Derivative

- Tapped Delay
- Transfer Fcn Real Zero
- Detect Increase
- Detect Decrease
- Detect Change
- Detect Rise Positive
- Detect Rise Nonnegative
- Detect Fall Negative
- Detect Fall Nonpositive

Version History

To ensure consistent results for models created in previous releases, set **Input processing** to:

- Columns as channels (frame based), for frame-based input signals (double-line)
- Elements as channels (sample based), for sample-based signal input signals (single-line)

If you are not sure of which option to choose, select and run the Simulink Upgrade Advisor checks:

- Check model for block upgrade issues requiring compile time information
- Check model for custom library blocks that rely on frame status of the signal, for blocks in a custom library

Inherited Setting on Save 2-D Signals

In the To Workspace block, setting the **Save Format** parameter to Structure or Array and the **Save 2-D signals as** parameter to Inherit from input now causes an error.

Version History

To ensure consistent results for models created in previous releases, set **Save 2-D signals as** to

- **3-D array (concatenate along third dimension)**, for sample-based input signals
- **2-D array (concatenate along first dimension)**, for frame-based input signals

For models created in R2015a:

- For frame-based processing, set **Save 2-D signals as** to **2-D array (concatenate along first dimension)**.
- For sample-based processing, set **Save 2-D signals as** to **3-D array (concatenate along third dimension)**.

If you are not sure of which option to choose, run these Simulink Upgrade Advisor checks:

- Check model for block upgrade issues requiring compile time information
- Check model for custom library blocks that rely on frame status of the signal, for blocks in a custom library

Frame-based Inputs Removed for Bias Block

Frame-based input support is removed from the Bias block.

Version History

To ensure consistent results for models created in older releases,

- 1 Change the block input to sample based by inserting a Frame Conversion block with **Sampling mode of output signal** set to `Sample-based`.
- 2 Insert a Frame Conversion block at the output of the block with **Sampling mode** set to `Frame based`.
- 3 Set the bias parameter of the block to `repmat(b, N, 1)`, where `b` is the value of the bias, and `N` is the input frame length.

Frame-based Inputs Removed for Tapped Delay Block

Frame-based input support is removed from the Tapped Delay block.

Version History

To ensure consistent results for models created in older releases, replace the Tapped Delay block by a Unit Delay block with **Input Processing** set to `Columns as channels (frame based)`.

Frame-based Input Removed for Transfer Fcn First Order Block

Frame-based signal support is removed from the Transfer Fcn First Order block.

Version History

To ensure consistent results for models created in older releases:

- 1 Replace the Transfer Fcn First Order block with a Discrete Transfer Fcn block.
- 2 In the Discrete Transfer Fcn block, set **Input Processing** to `Columns as channels (frame based)`.
- 3 Set **Numerator**, **Denominator** and **Initial Condition** of the new block to `[1-p 0]`, `[1 -p]` and `ic/(1-p)`, respectively, where `p` is the value of the pole and `ic` is the value of the initial condition on the Transfer Fcn First Order block.

Frame-based Input Removed for Transfer Fcn Lead or Lag Block

Frame-based input support is removed from the Transfer Fcn Lead or Lag block.

Version History

To ensure consistent results for models created in older releases:

- 1 Replace the Transfer Fcn Lead or Lag block by a Discrete Filter block.
- 2 In the Discrete Filter block, set **Input Processing** to `Columns as channels (frame based)` and **Filter Structure** to `Direct form I`.

- 3 Set **Numerator** and **Denominator** of the new block to $[1 - z]$ and $[1 - p]$, respectively, where p is the value of the pole and z is the value of the zero on the Transfer Fcn Lead or Lag block.

Sampling Mode Set to Frame-based

Setting **Sampling Mode** parameter to Frame based now errors for these blocks:

- Inport
- Signal Specification
- Outport


Version History

To ensure consistent results for models created in older releases, set **Sampling Mode** to Sample based or Auto instead.

If you are not sure of which option to choose, select and run the Simulink Upgrade Advisor checks:

- Check model for block upgrade issues requiring compile time information
- Check model for custom library blocks that rely on frame status of the signal, for blocks in a custom library

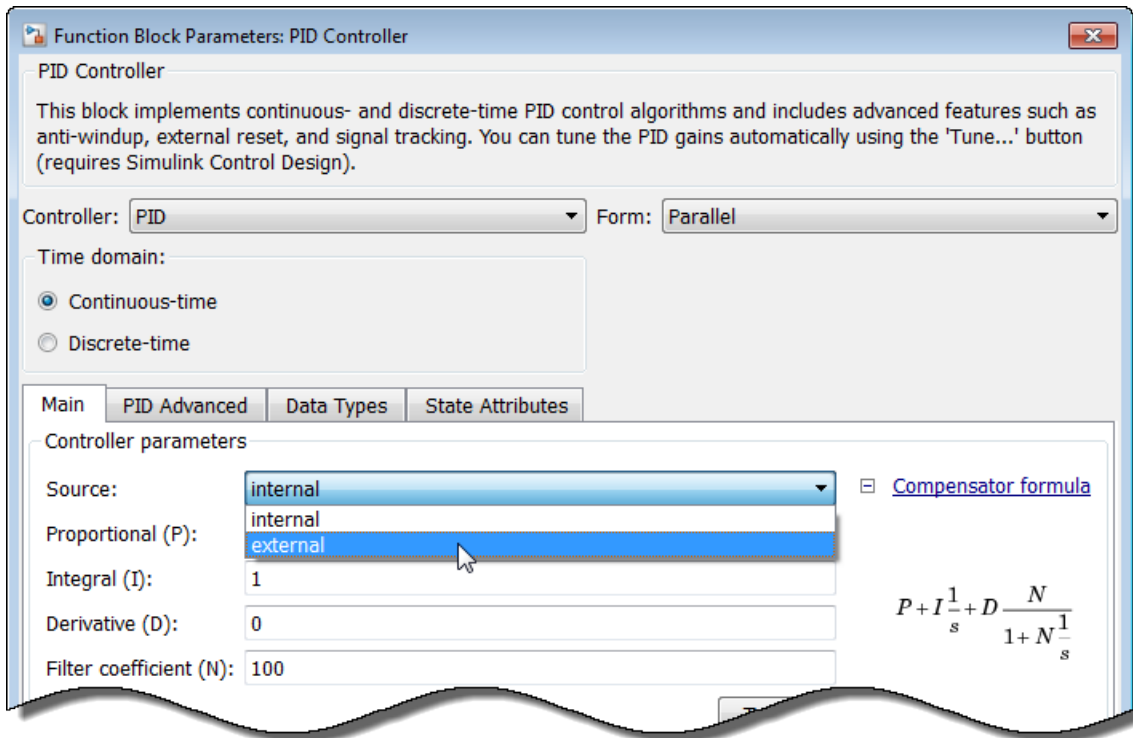
Scope block to Time Scope Block conversion

You can convert a Scope block to a Time Scope block to try the new block. On the Scope toolbar, click the Try Time Scope button . The Scope block converts to a Time Scope block. For Floating Scopes, this button is disabled (grayed out).

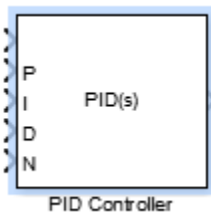
The Time Scope block includes simulation controls (run, forward, backward), additional support for signals (sample-based, frame-based), and debugging tools such as Cursors and Triggers. See Migrate Scope To Time Scope.

Option to provide PID gains as external inputs to PID Controller and PID controller (2DOF) blocks

A new option in the PID Controller and PID Controller (2DOF) blocks adds signal inputs for the PID gains and filter coefficients. Previously the PID parameters had to be entered in the block dialog box as numerical values or MATLAB expressions. Enabling external inputs for the parameters allows you to compute PID gains and filter coefficients externally to the block and provide them to the block as signal inputs. External gain input is useful, for example, to implement gain-scheduled PID control, in which controller gains are determined by logic or other calculation in the Simulink model and passed to the block. To enable external inputs for the PID coefficients, in the block dialog box, in the Controller parameters section, in the **Source** menu, select **external**.



When you click **OK** or **Apply**, the new inputs appear on the block in the Simulink model.



For more information about using the PID controller blocks, see the PID Controller and PID Controller (2 DOF) block reference pages.

Improvements for creating System objects

The following improvements have been made to creating your own System objects:

- Number of allowable code generation inputs increased to 32
- `isInputSizeLockedImpl` method for specifying whether the input port dimensions are locked
- `matlab.system.display.Action` class, used in the `getPropertyGroupsImpl` method, to define a MATLAB System block button that can call a System object method
- `getSimulateUsingImpl` and `showSimulateUsingImpl` methods to set the value of the `SimulateUsing` parameter and specify whether to show the `SimulateUsing` parameter in the MATLAB System block

MATLAB System block support for model coverage analysis

The Simulink Verification and Validation software now supports model coverage analysis for the MATLAB System block with the **Simulate using** parameter set to Code generation. For more information, see MATLAB System Block Limitations.

Enable port on the Delay block

The Delay block now provides an enable port to control its execution at every time step. You can show this port using the **Show enable port** parameter in the block dialog box. For more information, see the Delay block reference page.

MATLAB Function Blocks

More efficient generated code for logical indexing

Code generated for logical array indexing is faster and uses less memory than in previous releases. For example, the generated code for the following function is more efficient than in previous releases.

```
function x = foo(x,N)
assert(all(size(x) == [1 100]))
x(x>N) = N;
```

In R2015a, you do not have to replace `x(x>N) = N` with a for-loop to improve performance.

Faster compile time for large functions and models due to decreased constant folding limit

When possible, the code generation software replaces an expression with the result of the expression evaluation. This practice is called constant folding. In R2015a, the maximum number of instructions that the code generation software constant folds is lower than in previous releases. The lower constant folding limit can reduce compile times for large functions and models.

Version History

For large functions or models, it is possible that some expressions that were constant-folded in previous releases are not constant-folded in R2015a. In these cases, the generated code contains the expressions rather than the results of the evaluated expressions.

JIT compilation technology to reduce model update time

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

When a MATLAB Function block uses JIT mode, debugging is disabled. To debug, set a breakpoint in the MATLAB Function block before simulation. Simulink enables debugging, and does not use JIT mode.

Version History

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

In previous releases, if you set the command-line parameter `SFSIMEnableDebug`, the software enabled debugging for the model. Now, setting this parameter prevents the block from using JIT mode. Do not set this parameter if you want to improve model update performance using JIT mode.

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

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

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

Code generation for casts to and from types of variables declared using `coder.opaque`

For code generation, you can use the MATLAB `cast` function to cast a variable to or from a variable that is declared using `coder.opaque`. Use `cast` with `coder.opaque` only for numeric types.

To cast a variable declared by `coder.opaque` to a MATLAB type, you can use the `B = cast(A, type)` syntax. For example:

```
x = coder.opaque('size_t', '0');
x1 = cast(x, 'int32');
```

You can also use the `B = cast(A, 'like', p)` syntax. For example:

```
x = coder.opaque('size_t', '0');
x1 = cast(x, 'like', int32(0));
```

To cast a MATLAB variable to the type of a variable declared by `coder.opaque`, you must use the `B = cast(A, 'like', p)` syntax. For example:

```
x = int32(12);
x1 = coder.opaque('size_t', '0');
x2 = cast(x, 'like', x1);
```

Use `cast` with `coder.opaque` to generate the correct data types for:

- Inputs to C/C++ functions that you call using `coder.ceval`.
- Variables that you assign to outputs from C/C++ functions that you call using `coder.ceval`.

Without this casting, it is possible to receive compiler warnings during code generation.

Consider this MATLAB code:

```
yt = coder.opaque('size_t', '42');
yt = coder.ceval('foo');
y = cast(yt, 'int32');
```

- `coder.opaque` declares that `yt` has C type `size_t`.
- `y = cast(yt, 'int32')` converts `yt` to `int32` and assigns the result to `y`.

Because `y` is a MATLAB numeric type, you can use `y` as you would normally use a variable in your MATLAB code.

The generated code looks like:

```
size_t yt= 42;
int32_T y;
y = (int32_T)yt;
```

It is possible that the explicit cast in the generated code prevents a compiler warning.

Improved recognition of compile-time constants

In previous releases, the code generation software recognized that structure fields or array elements were constant only when all fields or elements were constant. In R2015a, in some cases, the software can recognize constant fields or constant elements even when some structure fields or array elements are not constant.

For example, consider the following code. Field `s.a` is constant and field `s.b` is not constant:

```
function y = create_array(x)
s.a = 10;
s.b = x;
y = zeros(1, s.a);
```

In previous releases, the software did not recognize that field `s.a` was constant. In the generated code, if variable-sizing was enabled, `y` was a variable-size array. If variable-sizing was disabled, the code generation software reported an error. In R2015a, the software recognizes that `s.a` is a constant. `y` is a static row vector with 10 elements.

As a result of this improvement, you can use individual assignments to assign constant values to structure fields. For example:

```
function y = mystruct(x)
s.a = 3;
s.b = 4;
y = zeros(s.a,s.b);
```

In previous releases, the software recognized the constants only if you defined the complete structure using the `struct` function: For example:

```
function y = mystruct(x)
s = struct('a', 3, 'b', 4);
y = zeros(s.a,s.b);
```

In some cases, the code generation software cannot recognize constant structure fields or array elements. See [Code Generation for Constants in Structures and Arrays](#).

Version History

The improved recognition of constant fields and elements can cause the following differences between code generated in R2015a and code generated in previous releases:

- A function output can be more specific in R2015a than it was in previous releases. An output that was complex in previous releases can be real in R2015a. An array output that was variable-size in previous releases can be fixed-size in R2015a.
- Some branches of code that are present in code generated using previous releases are eliminated from the generated code in R2015a.

Code generation for additional Image Processing Toolbox and Computer Vision System Toolbox functions

Image Processing Toolbox

- `bweuler`
- `bwlabel`
- `bwperim`
- `regionprops`
- `watershed`

See Image Processing Toolbox.

Computer Vision System Toolbox

- `opticalFlow`
- `vision.DeployableVideoPlayer` on Mac platform.

In previous releases, `vision.DeployableVideoPlayer` supported code generation on Linux and Windows platforms. In R2015a, `vision.DeployableVideoPlayer` also supports code generation on a Mac platform.

See Computer Vision System Toolbox.

Code generation for additional Communications System Toolbox, DSP System Toolbox, and Phased Array System Toolbox System objects

Communications System Toolbox

- `comm.CarrierSynchronizer`
- `comm.FMBroadcastDemodulator`
- `comm.FMBroadcastModulator`
- `comm.FMDemodulator`
- `comm.FMModulator`
- `comm.SymbolSynchronizer`

See Communications System Toolbox.

DSP System Toolbox

- `iirparameq`
- `dsp.LowpassFilter`
- `dsp.HighpassFilter`

See DSP System Toolbox.

Phased Array System Toolbox

- `pilotcalib`

- `phased.UCA`
- `phased.MFSKWaveform`

See Phased Array System Toolbox.

Code generation for additional Statistics and Machine Learning Toolbox functions

- `betafit`
- `betalike`
- `pca`
- `pearsrnd`

See Statistics and Machine Learning Toolbox.

Code generation for additional MATLAB functions

Linear Algebra

- `bandwidth`
- `isbanded`
- `isdiag`
- `istril`
- `istriu`
- `lsqnonneg`

See Linear Algebra in MATLAB.

Statistics in MATLAB

- `cummin`
- `cummax`

See Statistics in MATLAB.

Code generation for additional MATLAB function options

- `dimension` option for `cumsum` and `cumprod`

See Functions and Objects Supported for C and C++ Code Generation — Alphabetical List.

Model Advisor

Multiple instances of advisors

Model Advisor infrastructure changes allow you to save analysis time. To prepare for advisor analysis, you can open and configure:

- Different advisors on the same model
- The same advisor on different models

However, you can run the analysis for only one advisor at time.

Advisors	More information
Model Advisor	Consulting the Model Advisor
Performance Advisor	How Performance Advisor Improves Simulation Performance
Upgrade Advisor	Consult the Upgrade Advisor
Model Reference Conversion Advisor	Convert a Subsystem to a Referenced Model
Code Generation Advisor - Available with Simulink Coder	High-Level Code Generation Objectives
Fixed-Point Advisor - Available with Fixed-Point Designer	Fixed-Point Advisor
HDL Workflow Advisor - Available with HDL Coder	HDL Workflow Advisor
Simulink Code Inspector™ Compatibility Checker - Available with Simulink Code Inspector	Model Compatibility

Improved advisor startup performance

Model Advisor infrastructure changes have improved secondary startup performance of these advisors.

Advisors	More information
Model Advisor	Consulting the Model Advisor
Performance Advisor	How Performance Advisor Improves Simulation Performance
Upgrade Advisor	Consult the Upgrade Advisor
Model Reference Conversion Advisor	Convert a Subsystem to a Referenced Model
Code Generation Advisor - Available with Simulink Coder	High-Level Code Generation Objectives
Fixed-Point Advisor - Available with Fixed-Point Designer	Fixed-Point Advisor
HDL Workflow Advisor - Available with HDL Coder	HDL Workflow Advisor

Advisors	More information
Simulink Code Inspector Compatibility Checker - Available with Simulink Code Inspector	Model Compatibility

Model Advisor check input parameters retained for each instance of check

For Model Advisor checks with input parameters, instances of the check retain the input parameters that you enter. When you run the check, the results reflect the input parameter for that instance of the check. Previously, when you entered an input parameter for a check, all instances of the check were updated with the value you entered.

Model referencing checks in Model Advisor to reduce warning messages

The Model Advisor includes two new checks to reduce the number of warning messages during simulation and code generation. These checks replace the **Configuration Parameters > Diagnostics > Model Referencing > Model configuration mismatch** diagnostic, which generated many warning messages that you could safely ignore. For details, see “Model referencing checks in Model Advisor to reduce warning messages” on page 16-9 in the “Component-Based Modeling” section of these release notes.

Version History

The **Model configuration mismatch** diagnostic no longer appears in the **Configuration Parameters > Diagnostics > Model Referencing** pane. Simulink ignores the setting of the corresponding `ModelReferenceCSMismatchMessage` parameter. For details, see “Model referencing checks in Model Advisor to reduce warning messages” on page 16-9 in the “Component-Based Modeling” section of these release notes.

R2014b

Version: 8.4

New Features

Bug Fixes

Version History

Simulink Editor

Smart Editing Cues: Accelerate model building with just-in-time contextual prompts

The Simulink Editor provides several new model editing options that:

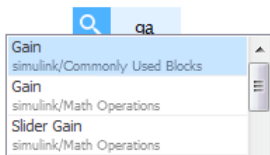
- Allow you to edit a model from within the diagram, without opening separate dialog boxes from a menu
- Provide editing options based on the context of your most recent editing operations

Add and Configure a Block without Leaving a Diagram

You can add a block to a model by typing a block name in a new Simulink Editor quick insert interface. You can perform a quick insert in several ways. For example, to add a Gain block and set the **Gain** parameter to 3, here is one approach:

- 1 Left-click in an empty space in the diagram, near where you want to add the Gain block.
- 2 Type “ga”.

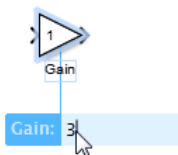
A list of blocks beginning with “ga” appears.



- 3 The Gain block appears first in the list, so just press **Enter**.

A Gain block appears in the diagram.

- 4 In the hot parameter edit box, enter a gain of 3.



The hot parameter edit box allows you to enter a value for one parameter per block that you add using the quick insert feature.

For more information, see [Add Blocks Using Quick Insert](#).

Insert a Complementary Block

You can insert a complementary block for the following blocks from within the canvas.

- GoTo and From
- Data Store Read and Data Store Write

For example, to add a From block associated with GoTo block in a model, hover over the GoTo block

- 1 Hover over the GoTo block.
- 2 Click and drag the blue tear-off guide.

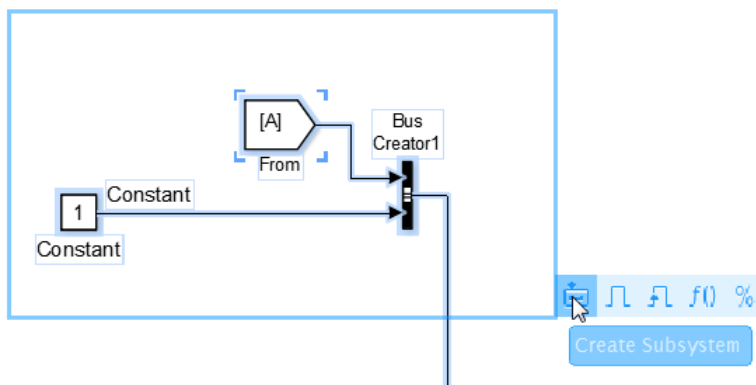


Perform Actions on a Marquee Selection

You can perform the following actions on a marquee selection (multiple selected objects in an area of a model), without leaving the canvas.

- Create a subsystem
- Create a triggered, enabled, or function-call subsystem
- Comment or uncomment the selected blocks

To select an action, hover over the blue action button. For example, to create a subsystem from the selected blocks, click the **Create Subsystem** button.



Connect Aligned Blocks Using a Guide

When you align the ports of two blocks, blue guide lines appear. Release the mouse button. Some blue guide lines remain. You can click a remaining blue guide line to connect blocks.

Viewmarks: Save graphical views of a model for quick access to areas of interest

You can create and manage viewmarks, which are bookmarks to parts of a model. Use viewmarks to capture graphical views of a model or parts of a model. You can capture viewmarks for specific levels in a model hierarchy. You can also pan and zoom in a displayed system, to capture the specific portion of interest.

Some examples of ways you can use viewmarks include:

- Navigate to different levels of complex models without opening multiple Simulink Editor tabs or windows.
- Review model designs

- Visually compare different versions of a model.

You can manage your viewmarks for all models in a single gallery of viewmarks, organized by model.

For more information, see [Use Viewmarks to Save Views of Models](#).

Annotation Connectors: Associate annotations with blocks in models

You can add blue connector lines between an annotation and a block. The connector is similar to a callout, identifying the block that an annotation applies to. As you move the annotation or block to which the connector attaches, Simulink redraws the connector. For more information, see [Add Lines to Connect Annotations to Blocks](#).

When you create an annotation, by default the annotation appears in the model. You can now configure an annotation so that you can choose to hide that annotation. This allows you to include annotations that provide additional information about a model without cluttering the model.

To configure an annotation so that you can hide it:

- 1 Right-click the annotation.
- 2 In the context menu, select **Convert to Markup**.

By default, all annotations appear in the model. To hide annotations that are converted to markup, select **Display > Hide Markup**.

R2014b includes two new `set_param` and `get_param` parameters relating to showing and hiding annotations:

- `MarkupType` — specifies whether an annotation can be hidden ('markup') or always appears ('model')
- `ShowMarkupType` — Specifies whether to display markup annotations ('on') or hide them ('off').

For more information, see [Show or Hide Annotations](#).

Edit bar for quick annotation formatting

When you click in an annotation, the Simulink Editor now displays an annotation edit bar. Click an edit bar button to format an annotation without having to open a menu. Types of formatting you can do include:

- Make text bold or italic
- Enlarge or shrink the font size
- Center text

Annotation table column and row resizing

In an annotation, you can interactively resize table columns or rows by dragging a table column or row border.

Reuse of annotation text formatting

You can copy the formatting from one piece of annotation text to other text in the same annotation, using **Format Painter** button in the annotation edit bar. For details, see Copy Formatting.

Annotation layering

You can specify whether an annotation appears in front of or behind other annotations. (Annotations always appear behind blocks.)

To display an annotation in front of another annotation, use the following steps.

- 1 Do a marquee selection (bounding box) that includes the annotation that you want to appear in front.
- 2 Select the **Diagram > Arrange > Front** menu option.

Access Diagnostic Viewer from status bar

When you update, simulate, or build a model, the status bar at the bottom of the Simulink Editor displays the total number of diagnostics generated. If these diagnostics are in the form of warnings or information, Simulink does not bring the Diagnostic Viewer into focus so that you can continue developing your model. When you are ready to diagnose warnings or view information, click the link displayed in the status bar to bring the Diagnostic Viewer into focus.

Printing to file

When you use the Print Model dialog box and select **Print To File**, the default file format is now PDF instead of the previous default of Postscript. Exporting to PDF generally makes it easier to share models with other people.

The default file name is now the name of the system you exported, which makes it easier to tell which system is in a file. Previously, Simulink generated a file name that did not reflect the system name.

Simulation Analysis and Performance

Fast Restart: Run consecutive simulations more quickly

Fast Restart is a new simulation workflow that eliminates the need for compiling a model repeatedly in iterative simulations. In this mode, the model compiles only in the first iteration. Simulink uses this compile information for successive runs, so there is no need to recompile. You can tune parameters or root-level inputs between runs as long as there are no structural changes to the model. This saves you time spent on recompiling and improves overall simulation efficiency.

Use Fast Restart to tune parameters in a model iteratively, calibrate a system for a desired response, or run multiple simulations in which the compile time is comparable to simulation time. For more information, see [How Fast Restart Improves Iterative Simulations](#).

New Simulation Data Inspector: View live signal data and access visualization options such as data cursors

You can now view a signal in the Simulation Data Inspector during model simulation. This is especially helpful for workflows that involve debugging and optimizing a model. For more information, see [Stream Data to the Simulation Data Inspector](#).

The Simulation Data Inspector now supports Simscape simulation output and includes new visualization options, such as data cursors, which help you inspect signal values in the Simulation Data Inspector plot.

Fixed Point Support for Conditional Breakpoints

Signals of fixed point data type now support conditional breakpoints based on the converted double value.

Quick Scan simulation in Performance Advisor for faster diagnosis

The Quick Scan feature in Performance Advisor helps you analyze a model quickly to deliver an approximate analysis of suboptimal conditions or settings in the model. Quick Scan does not require any preset conditions, and you can run it anytime. Using this feature, you can get a preview of potential performance improvement changes in a model without performing baseline measurement, multiple compilations, or simulations. See [Perform a Quick Scan Diagnosis](#) for more information.

Removal of warning when variable-step solver is selected for discrete models

When you simulate a discrete model with a variable-step solver, Simulink automatically switches the solver selection from `Variable-step continuous` (default) to `Variable-step discrete`. When doing so, Simulink no longer displays a warning about the switch, even if you have set **Diagnostics > Automatic solver parameter selection** to warning in the **Model Configuration Parameters** dialog box.

Block callbacks not evaluated in Rapid Accelerator mode with up-to-date check off

Previously, when you simulated a model in Rapid Accelerator mode with the `RapidAcceleratorUpToDateCheck` parameter set to `off`, Simulink evaluated the start and stop callbacks of the model and the blocks. Starting in R2014b, when the `RapidAcceleratorUpToDateCheck` parameter is set to `off`, Simulink evaluates only the model start and stop callbacks.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>slupdate</code>	Warns	Use the Upgrade Advisor instead.	<code>slupdate</code> will be removed in a future release. The <code>slupdate</code> command can only upgrade some parts of your model. Use the Upgrade Advisor instead. See Model Upgrades.

Improvements to Scope blocks and Scope viewers

Enhancements to viewing signals.

Scope blocks:

- Allow scrolling of signals to the left
- Supports fast restart

Floating Scope blocks:

- New **Log/Unlog Viewed Signals to Workspace** button on the **Scope Parameters > History** pane to set the **Log signal data** check boxes for attached signals
- Supports simulation stepping
- Supports fast restart
- Support for legends
- Ability to change line properties, axes, and figure colors.

Scope Viewers:

- Includes all the enhancements for Floating Scope blocks
- Supports the Dataset format for the **Signal Logging format** parameter in the **Configuration Parameters > Data Import/Export** pane.
- The functionality for the removed Floating Scope viewer is now included with Scope viewers.

See Scope and Floating Scope, Scope Viewer Tasks, Signal Selector.

Component-Based Modeling

Model Templates: Build models using design patterns that serve as starting points to solve common problems

Model templates enable reuse of settings and sharing of knowledge. Create models from templates to encourage best practices and take advantage of previous solutions to common problems. Instead of the blank canvas of a new model, select a template to help you get started.

Use built-in templates or create templates from models that you already configured for your environment or application.

For details, see [Create a New Model](#).

Simulink Functions: Create and call functions across Simulink and Stateflow

The Simulink Function block serves as a starting point for implementing functions using Simulink blocks. Simulink Function responds to the Function Caller block or to a call from Stateflow using a provided function prototype.

The Function Caller block, using a provided function prototype, invokes a function implementation, such as the Simulink Function block, or a Stateflow exported chart function.

Interface Display: View and trace the input and output signals of a model or subsystem

For Simulink models, the Interface feature helps you to understand and manage your model interfaces. You can start with the input and output connections at the edge of the model diagram and examine signal paths going in and coming out of a component. You can trace individual signals, buses, and elements of a bus to learn their uses in the model. The feature also displays characteristics of the inputs and outputs, such as dimensions, port data types, and sample times. Select **Display > Interface** in your Simulink model to turn on this display.

Model reference conversion enhancements

In R2014b, the Model Reference Conversion Advisor and the `Simulink.SubSystem.convertToModelReference` function provide new automatic fix options for converting a subsystem in a model that meets either of these conditions.

- The model contains a Data Store Memory block that Data Store Read or Data Store Write blocks access across the subsystem boundaries.
- The top model uses tunable parameters.

The `Simulink.SubSystem.convertToModelReference` function has a new `PropagateSignalStorageClass` argument. Set that argument to `true` to have the conversion propagate the signal storage class. For example:

```
open_system('sldemo_mdref_conversion');
Simulink.SubSystem.convertToModelReference(gcf, 'test_model', ...
'PropagateSignalStorageClass', true)
```

Version History

The automatic fix for Data Store Memory blocks creates `Simulink.Signal` objects and redirects Data Store Read and Data Store Write blocks to access those signal objects. The automatic fix for tunable parameters creates `Simulink.Parameter` objects for tunable parameters.

The conversion process creates data. Conversion data that the top model and the new referenced model share is saved in the data dictionary, if the top model uses one. Otherwise, the data is saved in a MAT-file.

- If the top model does not use a data dictionary, the conversion stores the data in a MAT-file whose name uses this format: `<model_name>_conversion_data.mat`.
 - If the model has callbacks or if you have scripts that rely on the previous variable names and locations, load the `<model_name>_conversion_data.mat` file before running the callbacks or scripts.
- If the top model uses a data dictionary, the conversion action depends on the state of the data dictionary.
 - If the data dictionary does not need to be saved, the conversion stores the data in the data dictionary.
 - If the data dictionary needs to be saved, the conversion does not save the data. You need to save the data dictionary when the conversion is complete.

Include Simulink Models as Variant Choices

Previously, you could only include subsystems as variant choices inside a Variant Subsystem block. If you wanted to include a model as a variant choice, you had to first wrap the model inside a Subsystem block and then include the subsystem block as the variant choice.

In R2014b, you can include a Simulink model as a variant choice inside a Variant Subsystem block without wrapping the model inside a Subsystem block.

You cannot include a Model block that contains variants inside a Variant Subsystem block. When you attempt this inclusion, Simulink suggests converting the Model block into a Subsystem block.

For example, consider a model block called **IntelligentController** that has two variants: `FuzzyLogicController` and `KalmanFilterController`. If you add this block inside a Variant Subsystem block, Simulink converts the model into a subsystem containing two model blocks: one representing `FuzzyLogicController` and the other, `KalmanFilterController`.

For more information, see [Define, Configure, and Activate Variant Choices](#).

Arithmetic and Bit-Wise Operators in Variant Condition Expressions

In R2014b, you can use arithmetic and bit-wise operators in variant condition expressions, provided the expressions evaluate to a Boolean value.

For a list of supported operators, see [Operators and Operands in Variant Condition Expressions](#).

Export of chart-level functions in export-function models

In R2014b, you can export graphical functions in Stateflow charts residing in export-function models. For more information, see [Export Stateflow Functions for Reuse](#).

Project and File Management

Block Dependencies in Impact Graph: Highlight the blocks affected by changes made to project files

Perform fine-grained dependency analysis using the Impact graph in Simulink Project. The Impact graph now displays which blocks have dependencies. For example, to find the impact of modifying a library, you can find design and test files dependencies. You can then expand the dependent files to see which subsystems have dependencies. You can view dependent blocks, models and libraries, and double-click to highlight the blocks in the models.

Other improvements in dependency analysis tools:

- Impact graph items now have popups when you mouse over a file, so that you can read the text and expand files without needing to change the zoom.
- Line routing in the graph is improved to reduce line crossings and make it easier to interpret large projects.
- Dependency and Impact analysis views have new toolstrips with reorganized tools to fit common workflows. The toolstrips enable discovery of tools previously hidden in context menus.
- You can now exclude external toolboxes from dependency analysis, which can avoid time-consuming analysis.

For details, see [Perform Impact Analysis](#).

Identify modified or conflicted folder contents using source control summary status

In Simulink Project, folders now display rolled-up source control status. This makes it easier to locate changes in files, particularly conflicted files. You can hover over the source control status for a folder to view a tooltip displaying how many files inside are modified, conflicted, added or deleted.

For details, see [View Modified Files and Resolve Conflicts](#).

Simplified file views in Simulink Project

In Simulink Project, the file views are simplified to combine the All files and Project Files nodes. You can find all files in the Files view, and use a filter if you only want to display files in the project. You can use drag and drop to add, move, and remove files from the project.

The source control and project information nodes are also combined to simplify the project tree.

For details of all these views, see [Try Simulink Project Tools with the Airframe Project](#).

Simplified browsing and sharing of project templates

Simulink Project now discovers templates using the MATLAB path. This enables simpler browsing and sharing of templates. You no longer need to import templates or manage a separate template path. If a colleague emails you a template, you can use it by placing it in the current folder or elsewhere on the MATLAB path. The template manager has been removed. You can still access all your existing templates using the Add Template dialog when creating a project.

For details, see [Create a New Project Using Templates](#).

SVN and Git example Simulink Projects

Try out source control features using new SVN and Git example Simulink Projects:

```
sldemo_slproject_airframe_svn  
sldemo_slproject_airframe_git
```

For details, see [Try Simulink Project Tools with the Airframe Project](#).

Data Management

Root Import Mapping tool

The Root Import Mapping tool has been updated. In addition to a new user interface, the tool lets you:

- Import signals from base workspace, MAT-files, and Microsoft Excel (Windows systems only) files
- Export signals
- Visualize signals
- Save and import scenarios
- Create new scenarios

For more information, see [Import and Map Root-Level Import Data](#).

Version History

The Root Import Mapping tool no longer accepts data in the time expression format.

Minimize and maximize buttons for the Configuration Parameters dialog box

You can minimize and maximize the Configuration Parameters dialog box using buttons on the title bar.

Overflow diagnostics to distinguish between wrap and saturation

You can now separately control the diagnostics for overflows that wrap and overflows that saturate by setting each diagnostic to **error**, **warning**, or **none**. These controls simplify debugging models in which only one type overflow is of interest. For example, if you need to detect only overflows that wrap, in the **Data Validity** pane of the Configuration Parameters dialog box you can set **Wrap on overflow** to **error** or **warning**, and set **Saturate on overflow** to **none**.

Change in behavior of `isequaln`

Previously, when you used function `isequaln` to compare two Simulink data objects, the function compared only the handles of the two objects. This behavior was incorrect and did not conform to the intended behavior of `isequaln` in MATLAB. Consider the following example:

```
a = Simulink.Parameter;  
b = Simulink.Parameter;  
isequaln(a,b);  
ans = false
```

Now, the behavior of `isequaln` has changed to conform to the behavior of `isequaln` in MATLAB. Now, `isequaln` compares two Simulink data objects by comparing their individual property values. Based on the above example, provided objects `a` and `b` have the same property values, the new result will be as follows:

```
a = Simulink.Parameter;
b = Simulink.Parameter;
isequaln(a,b);
ans = true
```

Version History

If you are using the `isequaln` function in your MATLAB code to compare Simulink data objects, check your code to ensure that it still works correctly.

Change in Simulink check for types derived from Simulink.IntEnumType

Previously, Simulink generated an error if any of the underlying values of a type derived from `Simulink.IntEnumType` did not fit on the emulation target. This check is defined by the configuration parameter `TargetBitPerInt`.

Now, Simulink generates an error if any of the underlying values of a type derived from `Simulink.IntEnumType` do not fit on the production target. This check is defined by the configuration parameter `ProdBitPerInt`.

Version History

Simulation errors can occur if any of the underlying values of types derived from `Simulink.IntEnumType` do not fit on your production target, even if those values previously fit on your emulation target.

Methods no longer inherited by Simulink enumerations

The base class `Simulink.IntEnumType` no longer defines these methods:

- `getDescription`
- `getHeaderFile`
- `getDataScope`
- `addClassNameToEnumNames`

You can define these methods in your enumeration classes, but classes derived from `Simulink.IntEnumType` no longer inherit these methods.

Previously, you called these inherited methods at the command prompt to query attributes of an enumerated type derived from `Simulink.IntEnumType`. For example, you called the method `getDefaultValue` to query the default enumeration member. Now, to query attributes of an enumerated type, you use the function `Simulink.data.getEnumTypeInfo`.

The table shows how to use `Simulink.data.getEnumTypeInfo`, instead of the methods, to query the attributes of an enumerated type `BasicColors`.

Previous method call	New function call
<code>BasicColors.getDescription</code>	<code>Simulink.data.getEnumTypeInfo(... 'BasicColors', 'Description')</code>

Previous method call	New function call
<code>BasicColors.getHeaderFile</code>	<code>Simulink.data.getEnumTypeInfo(... 'BasicColors','HeaderFile')</code>
<code>BasicColors.getDataScope</code>	<code>Simulink.data.getEnumTypeInfo(... 'BasicColors','DataScope')</code>
<code>BasicColors.addClassNameToEnumNames</code>	<code>Simulink.data.getEnumTypeInfo(... 'BasicColors','AddClassNameToEnumNames')</code>

Version History

If you run MATLAB code that uses these previously inherited methods without overriding them, MATLAB returns errors.

However, code that overrides these methods, as described in [Customize Simulink Enumeration](#), is unaffected.

Connection to Educational Hardware

More Arduino Support: Run your model on Arduino Leonardo, Mega ADK, Mini, Fio, Pro, Micro and Esplora boards

Updates to Simulink Support Package for Arduino Hardware: You can run your model on Arduino Leonardo, Mega ADK, Mini, Fio, Pro, Micro, and Esplora boards.

Documentation installation with hardware support package

Starting in R2014b, each hardware support package installs with its own documentation. See Simulink Supported Hardware for a list of support packages available for Simulink, with links to documentation.

Signal Management

Signal name inheritance from bus object elements

For a Bus Creator block that specifies a bus object, you can now have bus signal names inherit signal names from the corresponding element names in the bus object. To inherit signal names from bus element names, clear the new **Override bus signal names from inputs** check box. This approach:

- Enforces strong data typing.
- Avoids your having to enter a signal name multiple times. Without this option, you need to enter the signal names in the bus object and in the model, which can lead to accidentally creating signal name mismatches.
- Supports the array of buses requirement to have consistent signal names across array elements.

The Bus Creator block parameters dialog box has been reorganized to group related parameters. For details, see the Bus Creator block reference page.

Version History

In R2014b, if you open a model created before R2014b that uses a bus object for the output data type, clearing the **Override bus signal names from inputs** parameter might require you to change the model because:

- A signal name in a downstream Bus Selector or Bus Assignment block might no longer be the same.

Change any selected signal names in the Bus Selector or Bus Assignment block dialog boxes that do not match the corresponding bus object element names.

- Signal names in signal logging data might change.

Update scripts to reflect signal logging signal names that match the corresponding bus object names.

Faster and more flexible `Simulink.Bus.createMATLABStruct` function

If you use the `Simulink.Bus.createMATLABStruct` function repeatedly for the same model (for example, in a loop in a script), you can now improve performance by avoiding multiple model compilations. For improved speed, put the model in compile before using the function multiple times.

Also, you can now use a cell array of bus object names as an input argument for the function.

Block Enhancements

Nearest interpolation method available for n-D Lookup Table Block

The 1-D Lookup Table, 2-D Lookup Table, and n-D Lookup Table blocks have the option to select Nearest for interpolation methods.

MATLAB System block updates

The MATLAB System block:

- Can now be contained in a For Each Subsystem block. The new `supportsMultipleInstanceImpl` method enables using System objects in Simulink For Each subsystems. Include this method in your System object class definition file when you define a new kind of Simulink. For more information, see System Objects in For Each Subsystems.
- The **New > Block Extension** option has been renamed to **New > Simulink Extension**.

Level-1 MATLAB S-Functions

Support for Level-1 MATLAB S-Functions will be removed in a future release.

Version History

For information on alternatives to creating custom blocks, see Comparison of Custom Block Functionality.

Unfiltered-derivative option in discrete-time PID Controller blocks

You can now specify an unfiltered derivative term in the discrete-time PID Controller and PID Controller (2DOF) blocks. Previously, these blocks required a finite derivative filter constant on the derivative term.

To specify an unfiltered derivative, in the **Main** pane of the block dialog box, uncheck **Use filtered derivative**. Unchecking this option replaces the derivative filter with a Discrete Derivative block. The option is checked by default for compatibility with previous versions.

MATLAB Function Blocks

Code generation for additional Image Processing Toolbox and Computer Vision System Toolbox functions

Image Processing Toolbox

<code>bwdist</code>	<code>imadjust</code>	<code>intlut</code>	<code>ordfilt2</code>
<code>bwtraceboundary</code>	<code>imclearborder</code>	<code>iptcheckmap</code>	<code>rgb2ycbcr</code>
<code>fitgeotrans</code>	<code>imlincomb</code>	<code>medfilt2</code>	<code>stretchlim</code>
<code>histeq</code>	<code>imquantize</code>	<code>multithresh</code>	<code>ycbcr2rgb</code>

For the list of Image Processing Toolbox functions supported for code generation, see Image Processing Toolbox.

Computer Vision System Toolbox

`vision.DeployableVideoPlayer` on Linux.

For the list of Computer Vision System Toolbox functions supported for code generation, see Computer Vision System Toolbox.

Code generation for additional Communications System Toolbox and DSP System Toolbox functions and System objects

Communications System Toolbox

- `iqcoef2imbal`
- `iqimbal2coef`
- `comm.IQImbalanceCompensator`

For the list of Communications System Toolbox™ functions supported for code generation, see Communications System Toolbox.

DSP System Toolbox

- `dsp.CICCompensationDecimator`
- `dsp.CICCompensationInterpolator`
- `dsp.FarrowRateConverter`
- `dsp.FilterCascade`

You cannot generate code directly from this System object. You can use the `generateFilteringCode` method to generate a MATLAB function. You can generate C/C++ code from this MATLAB function.

- `dsp.FIRDecimator` for transposed structure
- `dsp.FIRHalfbandDecimator`
- `dsp.FIRHalfbandInterpolator`

- `dsp.PeakToPeak`
- `dsp.PeakToRMS`
- `dsp.PhaseExtractor`
- `dsp.SampleRateConverter`
- `dsp.StateLevels`

For the list of DSP System Toolbox functions and System objects supported for code generation, see DSP System Toolbox.

Code generation for ode23 and ode45 ordinary differential equation solvers in MATLAB

- `ode23`
- `ode45`
- `odeget`
- `odeset`

See Numerical Integration and Differentiation in MATLAB.

Code generation for additional MATLAB functions

Data and File Management in MATLAB

- `feof`
- `frewind`

See Data and File Management in MATLAB.

Linear Algebra in MATLAB

- `ishermitian`
- `issymmetric`

See Linear Algebra in MATLAB.

String Functions in MATLAB

`str2double`

See String Functions in MATLAB.

Code generation for additional MATLAB function options

- 'vector' and 'matrix' eigenvalue options for `eig`
- All output class options for `sum` and `prod`
- All output class options for `mean` except 'native' for integer types
- Multidimensional array support for `flipud`, `fliplr`, and `rot90`
- Dimension to operate along option for `circshift`

See [Functions and Objects Supported for C and C++ Code Generation — Alphabetical List](#).

Code generation for enumerated types based on built-in MATLAB integer types

In previous releases, enumerated types used in MATLAB Function blocks were based on the `Simulink.IntEnumType` class. In R2014b, you can also base an enumerated type on one of the following built-in MATLAB integer data types:

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`

You can use the base type to control the size of the enumerated type in generated C and C++ code. You can choose a base type to:

- Represent an enumerated type as a fixed-size integer that is portable to different targets.
- Reduce memory usage.
- Interface to legacy code.
- Match company standards.

The base type determines the representation of the enumerated types in generated C and C++ code. For the base type `Simulink.IntEnumType`, the code generation software generates a C enumeration type. For example:

```
typedef enum {  
    GREEN = 1,  
    RED  
} LEDcolor;
```

For the built-in integer base types, the code generation software generates a `typedef` statement for the enumerated type and `#define` statements for the enumerated values. For example:

```
typedef int16_T LEDcolor;  
  
#define GREEN ((LEDcolor)1)  
#define RED ((LEDcolor)2)
```

See [Enumerated Types Supported in MATLAB Function Blocks](#).

Code generation for function handles in structures

You can now generate code for structures containing fields that are function handles. See [Function Handle Definition for Code Generation](#).

Collapsed list for inherited properties in code generation report

The code generation report displays inherited object properties on the **Variables** tab. In R2014b, the list of inherited properties is collapsed by default.

Model Advisor

New check for Unit Delay and Zero-Order Hold blocks that perform rate transition

A new check in Model Advisor and Upgrade Advisor identifies Unit Delay and Zero-Order Hold blocks that are used for rate transition between input and output signals. The check prompts you to replace these blocks with actual Rate Transition blocks. The replacement provides accurate information about block transfer rates and enables traceability. For more information, see [Check Unit Delay and Zero-Order Hold blocks for rate transition](#).

Highlighted configuration parameters from Model Advisor reports

When you click a link to a configuration parameter from a Model Advisor report, the parameter is highlighted in the Configuration Parameters dialog box.

R2014a

Version: 8.3

New Features

Bug Fixes

Version History

Simulink Editor

Annotations with rich text, graphics, and hyperlinks

In addition to plain text and text formatted with TeX, annotations can now include:

- Rich text, which gives you the ability to format text and to add tables and lists, as you would using Microsoft Word
- Images, either by copying and pasting or by importing a graphics file
- Hyperlinks to Web pages or other documents

For details, see [Create an Annotation with a Link, Lists, and an Image](#).

Diagnostic Viewer to collect information, warnings, and error messages

In addition to displaying errors and warnings generated during simulation, the Diagnostic Viewer now displays information at the time of update diagram and build. The messages are displayed in a hierarchical structure within tabs for each model. For details, see [Manage Errors and Warnings](#)

Version History

The `diary` function does not intercept messages, errors, warning, and information transmitted to the Diagnostic Viewer.

Therefore, if you use the `diary` function to log messages, errors, warnings, and information generated during model build and simulation, replace instances of `diary` with `sldiagviewer.diary` in one of these ways.

- `sldiagviewer.diary('filename','encoding')`, where both *filename* and *encoding* are optional arguments.
 - The default value for *filename* is `diary.txt` in the current folder.
 - A valid value for encoding is UTF-8. If you do not specify a value, the default encoding value is set.
 - The command toggles the logging state of the specified file.
 - You can keep multiple log files active simultaneously.
- `sldiagviewer.diary('on')` and `sldiagviewer.diary('off')` toggle the logging state of the file specified in the last executed `sldiagviewer.diary` command.

If no file was specified in the last command, the logging state of the default file is toggled.

Option to bring contents of a hierarchical subsystem into the parent subsystem with one click

You can now expand subsystem contents to flatten the model hierarchy. Expanding a subsystem is useful when refactoring a model. Flattening a model hierarchy can be the end result, or just one step in refactoring. For example, you could pull a set of blocks up to the parent system by expanding the

subsystem, deselect the blocks that you want to leave in the parent, and then create a subsystem from the remaining selected blocks.

For details, see [Expand Subsystem Contents](#).

Support for native OS touch gestures, such as pinch-to-zoom and panning

MathWorks supports the use of multitouch gestures for panning and zooming on the Microsoft Windows with a Windows 7 certified or Windows 8 certified touch display.

- Zoom by spreading two fingers.
- Zoom in by pinching two fingers together.
- Pan by dragging two fingers.

Other supported Simulink platforms that also support multitouch gestures might also support pan and zoom gestures, but MathWorks has not fully tested those platforms.

Operating system print options for models

The Print Model dialog box includes a **Print using system dialog** button that opens the print dialog box for your operating system. The operating system print dialog box provides printing options for models in addition to those that the Print Model dialog box provides. For example, you can use the operating system print dialog box for double-sided printing, color printing (if your print driver supports color printing), and nonstandard paper sizes. For details, see [Specify the Page Layout and Print Job](#).

Preference for line crossing style

By default, straight signal lines that cross each other but are not connected display a slight gap before and after the vertical line where it intersects the horizontal line. You can change the line crossing style to line hops or solid lines. Use **Simulink Preferences > Editor Defaults > Line crossing style**. For details, see [Line crossing style](#).

Scalable graphics output to clipboard for Macintosh

On Macintosh platforms, when you copy a model to the clipboard, Simulink now saves the model in a scalable format, in addition to a bitmap format. When you paste from the clipboard to an application, that application selects the format that best meets its requirements. For details, see [Export Models to Third-Party Applications](#).

Sliders, dials, and spinboxes available as parameter controls in masks

In this release, Simulink provides the capability to control mask parameters through three additional widgets: sliders, dials, and spinboxes. For details, see [Parameters & Dialog Pane](#).

Component-Based Modeling

Option to choose default variants

In R2014a, you can specify a default variant choice. If no other variant is active, Simulink selects and uses the default choice.

For more information, see [Set Default Variant](#).

Option to choose variants that differ in number of input and output ports

In this release, variant subsystems can have different numbers of inports and outports, provided that they satisfy the following conditions:

- The inport names are a subset of the parent variant subsystem's inport names.
- The outport names are a subset of the parent variant subsystem's outport names.

During simulation, Simulink disables the inactive ports in a variant subsystem block.

Advisor-based workflow for converting subsystems to Model blocks

The new Model Reference Conversion Advisor simplifies the process of converting a subsystem to a referenced model. The advisor includes **Fix** buttons to automatically update the model for a successful conversion. After converting the subsystem, by default the advisor automatically updates the model, removing the Subsystem block and adding a Model block that references the newly created referenced model. For details, see [Convert a Subsystem to a Referenced Model](#).

The `Simulink.SubSystem.convertToModelReference` command now has a `UseConversionAdvisor` argument, which opens the Model Reference Conversion Advisor, and an `AutoFix` argument, which automatically fixes several kinds of conversion issues.

Single-model workflow for algorithm partitioning and targeting of multicore processors and FPGAs

You can use the concurrent execution dialog box to configure a model for concurrent execution on heterogeneous targets.

A new example has been added. Navigate to **Simulink > Examples > Modeling Features > Modeling Concurrency > Modeling Concurrent Execution on Multicore Targets**.

For a list of supported heterogeneous targets, see [Supported Heterogeneous Targets](#).

Easier MATLAB System block creation via autocompletion and browsing for System object names

The MATLAB System block dialog box has new browse and autocompletion capabilities to specify the System object name. It also allows creation of new System objects from templates.

Improved algebraic loop handling and reduced data copies with the Bus Selector block

The Bus Selector block processing now reduces:

- Artificial algebraic loops involving nonvirtual buses
- Data copies during bus element selection
- The number of lines of generated code

Version History

The Bus Selector block performance enhancements result in these compatibility considerations.

- Attaching a non-auto storage class at output of Bus Selector causes an error.

Workaround: Insert a Signal Conversion block after the Bus Selector block, and specify the storage class on the output of the Signal Conversion block.

- In model reference Accelerator mode, Bus Selector output signal logging using the ModelDataLogs format causes an error.

To work around this issue, change the signal logging format to Dataset or insert a Signal Conversion block after the Bus Selector block and log the Signal Conversion block output (instead of the Bus Selector block output).

- The priority specified on a Bus Selector block is ignored.
- A Bus Selector block connected to root Outputport block in referenced model honors the **Invalid root Inport/Outputport block connection** diagnostic. This is an issue only if the diagnostic is set to Error or Warning. There is no impact on generated code.

Workaround: Insert a Signal Conversion block after the Bus Selector block.

Faster response time when opening bus routing block dialog boxes and propagating signal labels

Opening the Bus Creator, Bus Selector, or Bus Assignment block dialog box is faster. Propagation of signal labels is also faster.

Usability enhancements to configure a model for concurrent execution on a target

Modeling for concurrent execution has the following enhancements:

- You can use MATLAB System blocks at the top level of a model to model parallel computations. For more information, see Model Parallel Computations.
- You can clear the **Solver > Allow tasks to execute concurrently on target** check box in the Configuration Parameters dialog for a referenced model. In this case, Simulink will not report a parameter mismatch for the model hierarchy. When this option is cleared, any Rate Transition blocks in the referenced model may have the `Ensure deterministic data transfer (maximum delay)` option selected.

- Simulink now handles the data transfer for the rate transitions occurring at the top level of a model. For more information, see [Configure Your Model](#).
- There are new function to work with models for concurrent execution. Use these functions instead of the command-line interface from previous releases. For more information, see [Command-Line Interface for Concurrent Execution](#).

Version History

Use the new functions instead of the following:

- `Simulink.SoftwareTarget.concurrentExecution`
- `Simulink.SoftwareTarget.AperiodicTrigger`
- `Simulink.SoftwareTarget.PeriodicTrigger`
- `Simulink.SoftwareTarget.PosixSignalHandler`
- `Simulink.SoftwareTarget.Task`
- `Simulink.SoftwareTarget.TaskConfiguration`
- `Simulink.SoftwareTarget.WindowsEventHandler`
- `Simulink.SoftwareTarget.Trigger`

Default setting of Underspecified initialization detection diagnostic is Simplified

Simplified initialization mode is the new default mode of initialization for all models created in R2014a. Simplified initialization mode has improved output consistency over classic initialization mode. For more information, see [Conditional Subsystem Output Initialization](#).

However, any new configuration sets created using `Simulink.ConfigSet` still use classic initialization mode.

Discrete-Time Integrator block has dialog box changes for initialization

The **Use Initial value as initial and reset value for** parameter of the Discrete-Time Integrator block has been replaced by the **Initial Condition Setting** parameter. The new parameter provides options to carry out either output or state initialization. An additional option of `Compatibility` does exist for this parameter, but use it only for compatibility purposes.

System objects Propagates mixin methods

Four new methods have been added to the Propagates mixin class. You use this mixin when creating a new kind of System object for use in the MATLAB System block in Simulink. You use these methods to query the input and specify the output of a System object.

- `propagatedInputComplexity`
- `propagatedInputDataType`
- `propagatedInputFixedSize`

- propagatedInputSize

Simulation Analysis and Performance

Reduced setup and build time for Model blocks when using Rapid Accelerator mode

Building a model that uses model referencing in Rapid Accelerator mode is faster than in previous releases, because Simulink reuses the simulation target.

If you have Parallel Computing Toolbox, you can enhance Rapid Accelerator build speed further by using parallel builds.

In R2014a, Simulink stores Rapid Accelerator build artifacts in the simulation cache folder instead of the code generation folder. This change avoids cluttering the code generation folder with simulation artifacts.

Version History

- If you have a script that relies on Rapid Accelerator build artifacts being stored in the code generation folder, the R2014a change to the artifact storage location requires you to update that script only if you specify two different folders for the Simulink preferences **Simulation cache folder** and **Code generation folder**.
- In R2014a, the Upgrade Advisor checks that S-functions work properly in top model Rapid Accelerator simulation. The advisor identifies and assists you with updating S-functions that meet all of the following conditions:
 - The S-function was created in R2013b or earlier, using either the S-Function Builder block or the Legacy Code Tool.
 - The S-function uses a bus signal as an input or output.
 - Simulink has added padding to that bus signal.

Before you simulate such S-functions in a top model in Rapid Accelerator mode, regenerate the S-functions with the tool used for creating them. The Model Advisor automatically regenerates as many of these S-functions as it can and identifies any other S-functions that you must regenerate.

Performance Advisor checks that validate overall performance improvement for all suggested changes and set code generation option for MATLAB System block

- Performance Advisor validates the overall performance improvement to your model using a final validation check. If performance is worse than baseline, Performance Advisor discards all changes and loads the original model. For more information, see Final Validation.
- Performance Advisor uses the **Check MATLAB System block simulation mode** check to identify which MATLAB System blocks can generate code and changes the **Simulate using** parameter value to Code generation where possible. For more information, see Check MATLAB System block simulation mode.

Improved navigation of the Performance Advisor report

For improved navigation and readability, in the Performance Advisor HTML report, you can:

- Filter the report to display results for checks that pass, warn, or fail.
- Display check results based on a keyword search.
- Quickly navigate to sections of the report using a navigation pane of the contents.
- Expand and collapse content in the check results.

For more information, see [Use Performance Advisor Reports](#).

Block behavior for asynchronous initiator with constant sample time

In the simulation target workflow, Simulink displays a warning for an asynchronous initiator with constant sample time. To avoid the warning, set the sample time parameter of the asynchronous initiator to `inherited`. This change in sample time parameter does not affect the code generation workflow.

Global setting for validation of checks in Performance Advisor

You can enable validation for all selected checks in Performance Advisor using a global setting. Previously, you could only enable validation for checks in Performance Advisor individually. For more information, see [Select Validation Actions for the Advice](#).

Guided setup in Performance Advisor

The user interface in Performance Advisor has been enhanced with a guided setup and workflow. The interface helps you to follow the workflow in Performance Advisor and also select settings required for performance optimization runs. For more information, see [Prepare a Model for Performance Advisor](#).

Project and File Management

Branching support through Git source control

Git integration with Simulink Project provides distributed source control with support for creating and merging branches and working offline. You can manage your models and source code using Git within Simulink Project.

For details, see [Set Up Git Source Control](#).

Tip If you want to add version control to your project files without sharing with another user, you can create a local Git repository in your sandbox with four clicks. For details, see [Add a Project to Git Source Control](#).

Comparison of project dependency analysis results

You can compare a Simulink Project impact analysis graph with a previously saved result of dependency analysis. This creates an interactive report you can use to investigate how the structure of project dependencies has changed.

For details, see [Save, Reload, and Compare Dependency Analysis Results](#).

Impact graph layout algorithm improved for easier identification of top models and their dependencies

Improved hierarchical layout algorithm for the Simulink Project Impact graph makes it easier to identify top models, now always on the left, and their dependencies, on the right. Graph layout is repeatable so the top model is always in the same place if you run dependency analysis multiple times. Graph performance is also faster. Dependencies are layered to vertically line up all files referenced by the same file and minimize layer crossings. These layers make it easier to identify dependencies at the same level. This makes it easier to see connections between a top model and its dependencies.

For details, see [Perform Impact Analysis](#).

Impact analysis example for finding and running impacted tests

The `sldemo_slproject_impact` example shows how to perform impact analysis in Simulink Project to find and run impacted tests to validate a change. You can search for dependencies of modified files to identify the tests you need to run. You can run a batch processing function on the files found by impact analysis and examine the results in Simulink Project.

For details, see [Perform Impact Analysis with a Simulink® Project](#).

Performance improvements for common scripting operations such as adding and removing files and labels

Common scripting operations for programmatically adding and removing files and file labels in a Simulink Project are now faster. For example, adding a label to 100 files is up to 40 times faster. Performance improvement depends on the project size.

For details, see Automate Project Management Tasks.

Conflict resolution tools to extract conflict markers

Source control tools can insert conflict markers in files. Simulink Project can identify conflict markers and offer to extract them and compare the files causing the conflict. You can then decide how to resolve the conflict.

For details, see Resolve Conflicts.

Updated Power Window Example

The Power Window Control Project example has been updated to take advantage of design concepts such as:

- Simulink Projects
- Referenced models
- Variant subsystems

For more information, see Power Window Case Study.

Data Management

Data dictionary for defining and managing design data associated with models

In R2014a, Simulink provides the ability to store, edit, and access design data using a data dictionary, which functions as a persistent repository of design data that your model uses.

For more information, see the following.

- What Is a Data Dictionary?
- Considerations before Migrating to Data Dictionary
- Migrate Single Model to Use Dictionary
- View and Revert Changes to Dictionary Entries

Rapid Accelerator mode signal logging enhanced to avoid rebuilds and to support buses and referenced models

In Rapid Accelerator mode, you can now log:

- Bus signals (including virtual, nonvirtual, and array of buses signals)
- Signals in referenced models

Also, in Rapid Accelerator mode, no rebuild occurs when you change:

- The **Configuration Parameters > Data Import/Export > Signal logging** parameter
- Any settings using the Signal Logging Selector

Simplified tuning of all parameters in referenced models

This release simplifies the way Simulink considers the `InlineParameters` option when it is set to `Off`. You can perform the following operations:

- Tune all block parameters in your model during simulation, either through the parameters themselves or through the tunable variables that they reference.
- Preserve the mapping between a block parameter and a variable in generated code even when the block parameter does not reference any tunable variables.
- Retain the mapping between tunable workspace variables and variables in generated code, irrespective of the `InlineParameters` setting.
- Set the value of `InlineParameters` to `Off` for model references.

These behaviors are consistent across models containing reusable subsystems and reference models.

The simplified behavior enhances the generated code and provides improved mapping between a block parameter and a variable in generated code.

Block parameter expression	Code generated previously	Code generated in R2014a
Expressions referencing global variables (e.g., K+1)	Variable name is not preserved. Block parameter name is preserved. <pre>struct Parameters_model_ { real_T Gain_Gain; // Expression: K+1 } y = model_P.Gain_Gain*u;</pre>	Expression is considered tunable. Variable name is preserved in code and is tunable. <pre>real_T K = 2.0; y = (K+1)*u;</pre>
Expressions referencing mask parameters for nonreusable subsystems (e.g., MP*3), the value of MP being a nontunable expression.	Variable name is not preserved. Block parameter name is preserved. <pre>struct Parameters_model_ { real_T Gain_Gain; // Expression: MP*3 } y = model_P.Gain_Gain*u;</pre>	Expression is considered tunable. Variable name is substituted by parameter value. <pre>struct Parameters_model_ { real_T Subsystem_MP; } y = (model_P.Subsystem_MP * 3) * u;</pre>
Expressions referencing model arguments (resp. mask parameters) for referenced models (resp. reusable subsystems) (e.g., Arg+1)	Variable name is not preserved. Block parameter name is preserved. <pre>struct Subsystem { Gain_Gain; // Expression: Arg+1 } y = model_P.Subsystem1.Gain_Gain*u;</pre>	Variable name is preserved as an argument name. <pre>subsystem(y, u, rtp_Arg) { y = (rtp_Arg+1)*u; }</pre>

Simulink.findVars supported in referenced models

In this release, Simulink provides the ability to search model reference hierarchies for variables that are used or not used.

See `Simulink.findVars` for information on how to run these searches from the command-line.

Saving workspace variables and their values to a MATLAB script

In a future release, Simulink will not support `Simulink.saveVars`, which provides the ability to save workspace variables to a MATLAB script.

Instead, you can use the MATLAB function `matlab.io.saveVariablesToScript` to perform this operation from the command line.

Version History

If your scripts contain references to the function `Simulink.saveVars`, replace these references with `matlab.io.saveVariablesToScript`.

Frame-based signals in the To Workspace block

In the To Workspace block parameters dialog box, if you set **Save format** to either **Array** or **Structure**, you can set the new **Save 2-D signals as** parameter. By default, this new parameter is set for sample-based signals. To have the To Workspace block treat the input signal as a frame-based signal, change the setting to 2-D array (concatenate along first dimension).

Use this new To Workspace block parameter to treat an input signal to the block as a frame-based signal. This method of having Simulink treat an input signal as frame-based configures the model to work in future releases, when Simulink will no longer support using a signal property to specify that a signal is frame-based.

For information about frame-based signals, in the DSP System Toolbox documentation, see [Frame-Based Processing](#).

Version History

Before R2014a, if you added a To Workspace block to a model, and you set **Save format** to either Array or Structure, you did not need to change any block parameter settings to handle frame-based signals.

In R2014a, in addition to setting **Save format** to either Array or Structure, you need to set the new **Save 2-D signals as** parameter to 2-D array (concatenate along first dimension).

If you open a model created before R2014a that inputs a frame-based signal to a To Workspace block, Simulink sets the **Save 2-D signals as** parameter to Inherit from input (this choice will be removed - see release notes). When you simulate the model, Simulink displays a warning, with a link to the Upgrade Advisor, which you can use to update your model to use the new frame-based signal handling approach (automatically setting **Save 2-D signals as** to 2-D array (concatenate along first dimension)).

Simulation mode consistency for Data Import/Export pane output options parameter

The behavior for the **Configuration Parameters > Data Import/Export > Output options** parameter value of Produce specified output only is now consistent for Rapid Accelerator, Accelerator, and Normal mode. In all three simulation modes, this setting results in automatically generating output signal data for variable-step solvers for the start and stop times, as well as for the times that the user specifies.

Version History

Prior to R2014a, in Rapid Accelerator mode the Produce specified output only setting only generated data for the specified times, and did not automatically generate data for the start and stop times.

You need to update scripts that rely on:

- Indexing that assumes that the first generated value is the first data value that you specify using the Produce specified output only setting
- The number of the data points

Dimension mismatch handling for root Inport blocks improved

For the input and output of a root Inport block, Simulink no longer distinguishes between a dimension of 1 and a dimension that is a vector of ones.

For example, Simulink no longer reports an error when both of these conditions apply:

- Input data for a root Inport block includes an element whose dimension is [1x1].
- The root Inport block has as its data type for output a bus object whose corresponding element has a dimension of [1].

Version History

Before R2014a, Simulink reported an error for this kind of dimension mismatches. If you have tests that rely on Simulink reporting an error, you need to modify the tests to identify such mismatches explicitly.

Simulink.Bus.createObject support for structures of timeseries objects

You can use the `Simulink.Bus.createObject` function to create bus objects from a structure of MATLAB timeseries objects. This facilitates importing logged signal data.

Signal logging override for model reference variants

You can override programmatically a subset of signals for model reference variant systems, including:

- Model reference variants
- Model blocks that contain a Subsystem Variant block or model reference variant

To log a subset of signals for these model reference variant systems, set the `SignalLoggingSaveFormat` parameter to `Dataset`. For details, see [Override Signal Logging Settings from MATLAB](#).

Improved To Workspace block default for fixed-point data

The block parameter **Log fixed-point data as fi object** is now enabled by default for the To Workspace block in the Simulink library. This causes Simulink to log data to the To Workspace block in a data format designed for fixed-point data.

Version History

You need to update any scripts that:

- Insert from the Simulink library a To Workspace block that logs fixed-point data
- Rely on the **Log fixed-point data as fi object** parameter not being enabled (which results in logging fixed-point data as doubles)

Legacy Code Tool support for 2-D row-major matrix

When you use the `legacy_code` function, you can now specify the automatic conversion of a matrix between a 2-D column-major format and a row-major format. Use the `convert2DMatrixToRowMajor` S-function option. The 2-D column-major format is used by MATLAB, Simulink, and the generated code. The row-major format is used by C. By default, the value is `false` (0).

Model Explorer property name filtering refined

In the Model Explorer **Filter Contents** edit box, when you enter a property name without specifying a value (for example, `BlockType:`), the **Contents** pane displays only those objects that have that property.

For details, see Filter Objects in the Model Explorer.

Connection to Educational Hardware

Support for Arduino Due hardware

You can use the Arduino Due support package independently of the Arduino support package for Arduino Uno, Arduino Mega 2560, and Arduino Nano hardware. The Arduino Due support package shares the common block library with the Arduino Support package including the Ethernet and the WiFi blocks. The target hardware for this support package is a 32-bit ARM architecture-based microcontroller.

To use this support package, install Arduino Due support package as follows:

- 1 In the Command Window, enter `supportPackageInstaller`.
- 2 Using `supportPackageInstaller`, install the Arduino Due support package.

Support for Arduino WiFi Shield hardware

You can use the Arduino WiFi Shield with the Simulink Support Package for Arduino Hardware to connect to wireless networks. The block library for the Arduino WiFi shield hardware includes WiFi TCP/IP and WiFi UDP blocks that enable you to design wireless communication in embedded systems. The WiFi shield supports wireless external mode simulation via TCP/IP.

The block library for the Arduino WiFi Shield hardware includes:

- Arduino WiFi TCP/IP Send and Arduino WiFi TCP/IP Receive blocks that enable TCP/IP based wireless communication.
- Arduino WiFi UDP Send and Arduino WiFi UDP Receive blocks that enable UDP based wireless communication.

To install or update this support package, perform the steps described in [Install Support for Arduino Hardware](#).

For more information, see [Arduino Hardware](#)

Support for LEGO MINDSTORMS EV3 hardware

You can run Simulink models on LEGO MINDSTORMS EV3 hardware. You can also tune parameter values in the model, and receive data from the model, while it is running on LEGO MINDSTORMS EV3 hardware.

Use the **Simulink Support Package for LEGO MINDSTORMS EV3 Hardware** block library to access LEGO MINDSTORMS EV3 peripherals:

- Button
- Color Sensor
- Encoder
- Gyro Sensor
- Infrared Sensor
- Display

- Motor
- Speaker
- Touch Sensor
- Ultrasonic Sensor

To install or update this support package, perform the steps described in [Install Support for LEGO MINDSTORMS EV3 Hardware](#).

For more information, see [LEGO MINDSTORMS EV3 Hardware](#).

Updates to support for LEGO MINDSTORMS NXT hardware

You can use the dGPS Sensor from Dexter Industries with the Simulink Support Package for LEGO MINDSTORMS NXT Hardware. Use the GPS Sensor block to measure the latitude and longitude of your current position, or get the distance and angle from your current position to a destination latitude and longitude.

To install or update this support package, perform the steps described in [Install Support for LEGO MINDSTORMS NXT Hardware](#).

For more information, see [LEGO MINDSTORMS NXT Hardware](#).

Support for Samsung GALAXY Android devices

You can run Simulink models on the Samsung GALAXY S4 and Tab 2 10.1 devices. You can also tune parameter values in the model, and receive data from the model, while it is running on Samsung GALAXY S4 and Tab 2 10.1 devices.

Use the **Simulink Support Package for Samsung GALAXY Android Devices** block library to access the Samsung GALAXY Android hardware:

- Accelerometer
- Ambient Temperature Sensor
- Audio Capture
- Audio Playback
- Camera
- Display
- FromApp
- Gyroscope
- Humidity Sensor
- Light Sensor
- Location Sensor
- Pressure Sensor
- ToApp
- UDP Receive
- UDP Send

To install or update this support package, perform the steps described in Install Support for Samsung GALAXY Android Devices.

Block Enhancements

Enumerated data types in the Direct Lookup Table (n-D) block

The Direct Lookup Table (n-D) block now supports enumerated data types for its index and table data values.

Improved performance and code readability in linear search algorithm for Prelookup and n-D Lookup Table blocks

The Prelookup and n-D Lookup Table blocks have an improved linear search algorithm which improves performance and code readability.

System object file templates

The MATLAB System block has new templates to help create System objects.

Relay block output of fixed-in-minor-step continuous signal for continuous input

The Relay block now delivers fixed-in-minor-step continuous signals for continuous input. Previously, the Relay block output signal was a continuous output for a continuous input. The new sample time represents the behavior of the block more accurately.

MATLAB Function Blocks

Generating Simulation Target typedefs for imported bus and enumerated data types

You can configure Simulink to generate typedefs for imported bus and enumerated data types or to use typedefs you provide in a header file. In the **Simulation Target** pane of the **Configuration Parameters** dialog box, use the **Generate typedefs for imported bus and enumeration types** check box. This setting applies to model simulation for MATLAB Function blocks. For more information see [Simulation Target Pane: General](#).

Complex data types in data stores

You can now access complex data types in Data Store Memory blocks and Simulink.Signal objects using MATLAB Function blocks.

Unicode character support for MATLAB Function block names

You can use international characters for MATLAB Function block names in the Simulink Editor.

Support for int64 and uint64 data types in MATLAB Function blocks

You can now use int64 and uint64 data types in MATLAB code inside MATLAB Function blocks. You cannot use int64 or uint64 types for input or output signals to MATLAB Function blocks.

Streamlined MEX compiler setup and improved troubleshooting

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

Code generation for additional Image Processing Toolbox functions

Image Processing Toolbox

<code>affine2d</code>	<code>im2uint16</code>	<code>imhist</code>
<code>bwpack</code>	<code>im2uint8</code>	<code>imopen</code>
<code>bwselect</code>	<code>imbothat</code>	<code>imref2d</code>
<code>bwunpack</code>	<code>imclose</code>	<code>imref3d</code>
<code>edge</code>	<code>imdilate</code>	<code>imtophat</code>
<code>getrangefromclass</code>	<code>imerode</code>	<code>imwarp</code>
<code>im2double</code>	<code>imextendedmax</code>	<code>mean2</code>
<code>im2int16</code>	<code>imextendedmin</code>	<code>projective2d</code>
<code>im2single</code>	<code>imfilter</code>	<code>strel</code>

See [Image Processing Toolbox](#).

Code generation for additional Signal Processing Toolbox, Communications System Toolbox, and DSP System Toolbox functions and System objects

Signal Processing Toolbox

- `findpeaks`
- `db2pow`
- `pow2db`

See Signal Processing Toolbox.

Communications System Toolbox

- `comm.OFDMModulator`
- `comm.OFDMDemodulator`

See Communications System Toolbox.

DSP System Toolbox

<code>ca2tf</code>	<code>firhalfband</code>	<code>ifir</code>	<code>iirnotch</code>
<code>cl2tf</code>	<code>firlpnorm</code>	<code>iircomb</code>	<code>iirpeak</code>
<code>firceqrip</code>	<code>firminphase</code>	<code>iirgrpdelay</code>	<code>tf2ca</code>
<code>fireqint</code>	<code>firnyquist</code>	<code>iirlpnorm</code>	<code>tf2cl</code>
<code>firgr</code>	<code>firpr2chfb</code>	<code>iirlpnormc</code>	<code>dsp.DCBlocker</code>

See DSP System Toolbox.

Code generation for MATLAB `fminsearch` optimization function, additional interpolation functions, and additional `interp1` and `interp2` interpolation methods

You can generate code for the `interp1` function 'spline' and 'v5cubic' interpolation methods.

You can generate code for the `interp2` function 'spline' and 'cubic' methods.

You can generate code for these interpolation functions:

- `interp3`
- `mkpp`
- `pchip`
- `ppval`
- `spline`
- `unmkpp`

See Interpolation and Computational Geometry in MATLAB.

You can generate code for these optimization functions:

- `fminsearch`
- `optimget`
- `optimset`

See Optimization Functions in MATLAB.

Code generation for `fread` function

You can now generate code for the `fread` function.

Enhanced code generation for switch statements

Code generation now supports:

- Switch expressions and case expressions that are noninteger numbers, nonconstant strings, variable-size strings, or empty matrices
- Case expressions with mixed types and sizes

If all case expressions are scalar integer values, the code generation software generates a C `switch` statement. At run time, if the switch value is not an integer, the code generation software generates an error.

When the case expressions contain noninteger or nonscalar values, the code generation software generates C `if` statements in place of a C `switch` statement.

Code generation for value classes with `set.prop` methods

You can now generate code for value classes that have `set.prop` methods.

Code generation error for properties that use `AbortSet` attribute

Previously, when the current and new property values were equal, the generated code set the property value and called the `set` property method regardless of the value of the `AbortSet` attribute. When the `AbortSet` attribute was true, the generated-code behavior differed from the MATLAB behavior.

In R2014a, the code generation software generates an error if your code has properties that use the `AbortSet` attribute.

Toolbox functions for code generation

See [Functions and Objects Supported for C and C++ Code Generation — Alphabetical List and Functions and Objects Supported for C and C++ Code Generation — Categorical List](#).

Communications System Toolbox

- `comm.OFDMModulator`
- `comm.OFDMDemodulator`

Data and File Management Functions

fread

DSP System Toolbox Classes and Functions

ca2tf	firhalfband	ifir	iirnotch
cl2tf	firlpnorm	iircomb	iirpeak
firceqrip	firminphase	iirgrpdelay	tf2ca
fireqint	firnyquist	iirlpnorm	tf2cl
firgr	firpr2chfb	iirlpnormc	dsp.DCBlocker

Image Processing Toolbox

affine2d	im2uint16	imhist
bwpack	im2uint8	imopen
bwselect	imbothat	imref2d
bwunpack	imclose	imref3d
edge	imdilate	imtophat
getrangefromclass	imerode	imwarp
im2double	imextendedmax	mean2
im2int16	imextendedmin	projective2d
im2single	imfilter	strel

Interpolation and Computational Geometry Functions

- interp2
- interp3
- mkpp
- pchip
- ppval
- polyarea
- rectint
- spline
- unmkpp

Matrix and Array Functions

flip

Optimization Functions

- fminsearch
- optimget
- optimset

Polynomial Functions

- `polyder`
- `polyint`
- `polyvalm`

Signal Processing Toolbox

- `findpeaks`
- `db2pow`
- `pow2db`

Modeling Guidelines

Modeling guidelines for high-integrity systems

The following are new modeling guidelines to develop models and generate code for high-integrity systems:

- hisl_0029: Usage of Assignment blocks
- himl_0004: MATLAB Code Analyzer recommendations for code generation
- himl_0005: Usage of global variables in MATLAB functions
- himl_0006: MATLAB code if / elseif / else patterns
- himl_0007: MATLAB code switch / case / otherwise patterns
- himl_0008: MATLAB code relational operator data types
- himl_0009: MATLAB code with equal / not equal relational operators
- himl_0010: MATLAB code with logical operators and functions

Model Advisor

Improved navigation of the Model Advisor report, including a navigation pane, collapsible content, and filters based on check status

For improved navigation and readability, the Model Advisor HTML report allows you to:

- Filter the report to display results for checks that pass, warn, or fail.
- Display check results based on a keyword search.
- Quickly navigate to sections of the report using a table-of-contents navigation pane.
- Expand and collapse content in the check results.

For more information, see [View and Save Model Advisor Reports](#).

Option to run Model Advisor checks in the background

If you have a Parallel Computing Toolbox license, you can run the Model Advisor in the background, allowing you to continue working on your model during Model Advisor analysis. When you start a Model Advisor analysis run in the background, Model Advisor takes a snapshot of your model. The Model Advisor analysis does not reflect changes that you make to your model while Model Advisor is running. For more information, see [Run Checks in the Background](#).

Upgrade Advisor check for `get_param` calls for block `CompiledSampleTime`

Upgrade Advisor has a new check to scan MATLAB files in your working environment for `get_param` function calls that return the block `CompiledSampleTime` parameter. For multirate blocks, Simulink returns this parameter as a cell array of pairs of doubles. Run this check to identify MATLAB code that accepts the block `CompiledSampleTime` parameter as a pair of doubles. You can edit these instances of code to accept a cell array of pairs of doubles.

For more information, see [Check `get_param` calls for block `CompiledSampleTime`](#).

Upgrade Advisor check for signal logging in Rapid Accelerator mode

When simulating your model in Rapid Accelerator mode, you can run a check in Upgrade Advisor to see if signals logged in your model are globally disabled. Using this check, you can enable signal logging globally.

For more information, see [Check Rapid Accelerator signal logging](#).

R2013b

Version: 8.2

New Features

Bug Fixes

Version History

New Simulink Editor

Ability to add rich controls, links, and images to customized block interfaces using the Mask Editor

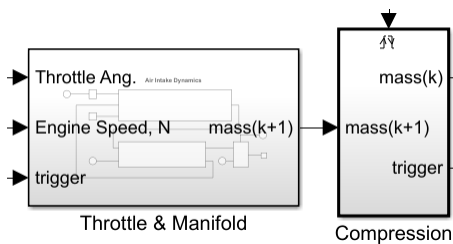
The mask editor is enhanced to support designing mask dialogs using controls such as images, links, and buttons. For details, see Masking.

Content preview for subsystems and Stateflow charts

Without opening the blocks, in the Simulink Editor you can preview the content of the following hierarchical items:

- Subsystems
- Model blocks
- Stateflow charts and subcharts

For example, the Throttle & Manifold subsystem uses content preview, and the Compression subsystem does not.



Content preview can help to make a model self-documenting.

For details, see Preview Content of Hierarchical Items.

Comment-through capability to temporarily delete blocks and connect input signals to output signals

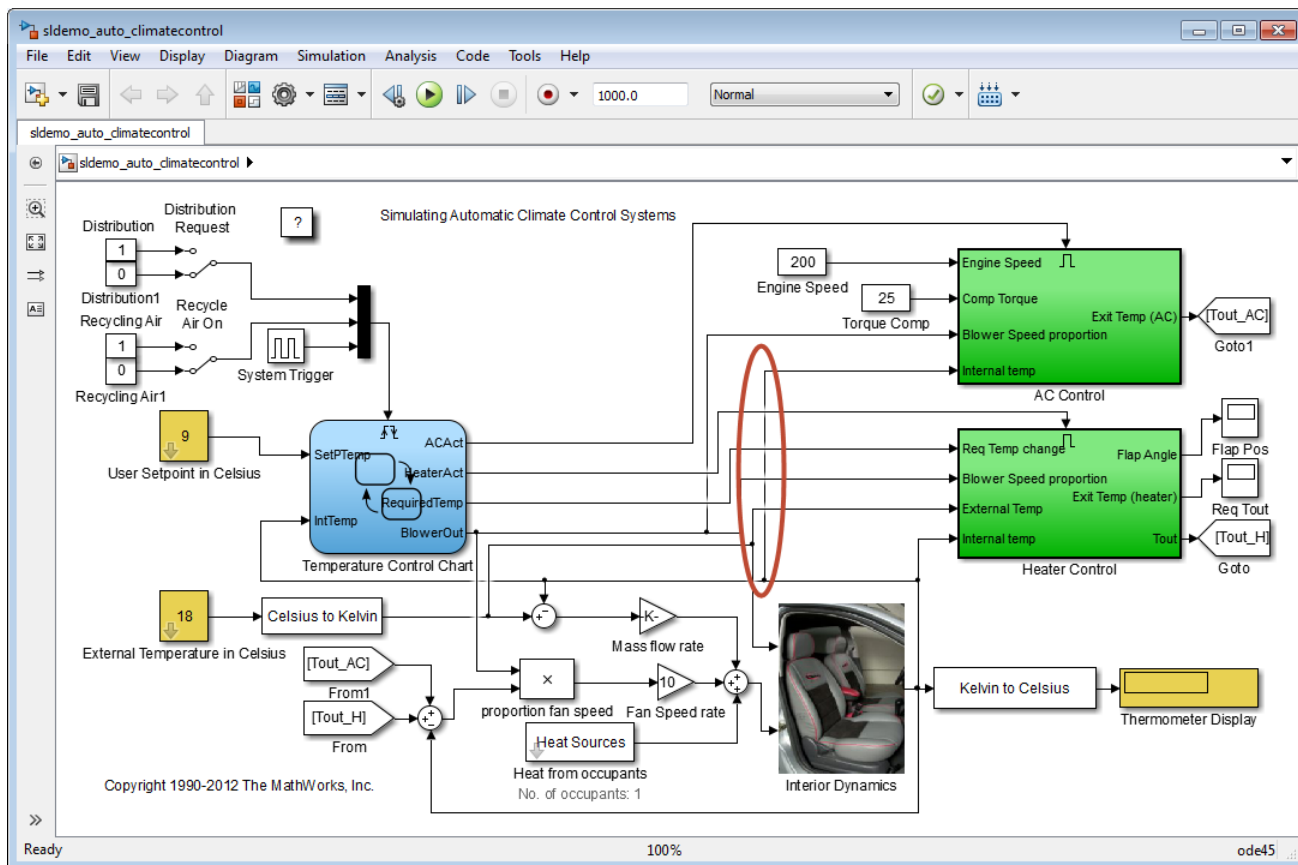
You can comment through blocks such that they appear transparent during simulation. For more information, see Comment Blocks.

New options added to find_system command

You can exclude or include blocks from search using the IncludeCommented option of the find_system command. For more information, see Modeling Basics.

Visual cues for signal lines that cross

Unconnected signal lines that cross have a small break on either side of the intersection, to show that the signals are not connected. For example:



For details, see Signal Line Crossings.

UTF-16 character support for block names, signal labels, and annotations in local languages

You can use international characters when editing text in the Simulink Editor for the following:

- Block names (the text below the block)
- Block annotations (text in the Block Properties dialog box **Annotations** tab, for block property tokens such as %<BlockDescription>)
- Signal names (in the signal label text box below a signal)
- Model annotations

You can use international text for block names and signal names when using MATLAB commands such as `find_system`.

Unified Print Model dialog box for printing

The reorganized Print Model dialog box provides the same printing interface on Microsoft Windows, Macintosh, and Linux platforms.

To consolidate model printing options, the Print Model dialog now includes:

- Paper type
- Page orientation

For details, see Printing Options.

Version History

The reorganized Print Model dialog box no longer has these options:

- Selection for **Print Range**

You can use tiled printing and options such as **Current system and below** to specify the part of a model to print.

- **Properties**

Instead, use the **File > Print > Printer Setup > Properties** dialog box.

Block Parameters dialog box access from Block Properties dialog box

In the Block Properties dialog box, you can open the Block Parameters dialog box by using the **Open Block** link. For details, see Block Properties Dialog Box.

Notification bar icon indicator for multiple notifications

After you simulate a model, if there are multiple notifications, the Simulink Editor notification bar icon becomes an arrow. For details, see Record Simulation Data.

Component-Based Modeling

MATLAB System block for including System objects in Simulink models

The MATLAB System block is a new block in the Simulink User-Defined Functions library. Use this block to create a Simulink block that includes a System object in your model. This capability is useful for including algorithms. For more information, see [System Object Integration](#).

Variant Manager that manages all the variants in a model in one place

Variant Manager is a graphical tool that allows you to define and manage multiple variant configurations for Simulink models. For details, see [Variant Management](#).

Improved componentization capabilities for modeling scheduling diagrams with root-level function-call inports

Models can now export functions by using root-level function-call Inport blocks. Such models can simulate in Normal, Accelerator, Rapid Accelerator, SIL, and PIL simulation modes.

Array of buses signal logging in model reference accelerator mode

You can log an array of buses signal in models using model reference accelerator mode or Normal mode. In R2013a, you could only log array of buses signals in Normal mode.

Ability to add, delete, and move input signals within Bus Creator block

The Bus Creator block includes **Up**, **Down**, **Add**, and **Remove** buttons to simplify organizing the input signals. When you add or remove signals, Simulink automatically updates the **Number of inputs** parameter of the Bus Creator block and maintains port connectivity in the model.

For details, see [Reorder, Add, or Remove Signals in the Bus Creator block reference page](#).

Streamlined approach to migrating from Classic to Simplified initialization mode

Simplified initialization mode has increased flexibility, such as accepting an empty matrix (`[]`) for **Initial output**. These abilities make it easier to migrate custom libraries and models from classic to simplified initialization mode. For more information, see [Conditional Subsystem Output Initialization](#).

Simplified display of sorted execution order

The display of sorted execution order is enhanced for subsystems and blocks such as function-call blocks, making it easier to understand the execution order in a model.

Enhanced model reference rebuild algorithm for MATLAB Function blocks

In conjunction with the **Model Configuration > Model Referencing > Rebuild** parameter, Simulink examines a set of known target dependencies when determining whether they have

changed. R2013b adds another known target dependency: MATLAB files used by MATLAB function blocks.

For details, see [Rebuild](#).

Simulation Analysis and Performance

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

The Windows 64-bit platform now includes LCC-win64 as the default compiler for running simulations. You no longer have to install a separate compiler for simulation in Stateflow and Simulink. You can run simulations in Accelerator and Rapid Accelerator modes using this compiler. You can also build model reference simulation targets and accelerate MATLAB System block simulations.

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

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

Signal logging in Rapid Accelerator mode

You can log signals in Rapid Accelerator mode. Signal logging in Rapid Accelerator mode is similar to signal logging in Normal and Accelerator mode. During model development and testing, using Rapid Accelerator mode usually speeds up simulation with signal logging. For details, see Signal Logging in Rapid Accelerator Mode.

Performance Advisor checks for Rapid Accelerator mode and data store memory diagnostics

Performance Advisor checks have been enhanced to improve simulation and performance optimization.

A single check compares all four simulation modes before recommending the optimal choice:

- Normal
- Accelerator
- Rapid Accelerator
- Rapid Accelerator with up-to-date check off

There is another check that compares compiler optimizations if the selected simulation mode is Accelerator or Rapid Accelerator.

Also, you can now disable runtime diagnostics for data store memory to reduce runtime overhead and improve simulation speed. For more information, see Diagnostics.

Long long integers in simulation targets for faster simulation on Win64 machines

If your target hardware and your compiler support the C99 long long integer data type, you can select this data type for code generation and simulation. Using long long results in more efficient

generated code that contains fewer cumbersome operations and multiword helper functions. This data type also provides more accurate simulation results for fixed-point and integer simulations. If you are using Microsoft Windows (64-bit), using long long improves performance for many workflows including using Accelerator mode and working with Stateflow software.

For more information, see the **Enable long long** and **Number of bits: long long** configuration parameters on the Hardware Implementation Pane.

At the command line, you can use the following new model parameters:

- **ProdLongLongMode**: Specifies that your C compiler supports the long long data type. Set this parameter to 'on' to enable **ProdBitPerLongLong**.
- **ProdBitPerLongLong**: Describes the length in bits of the C long long data type supported by the production hardware.
- **TargetLongLongMode**: Specifies whether your C compiler supports the long long data type. Set this parameter to 'on' to enable **TargetBitPerLongLong**.
- **TargetBitPerLongLong**: Describes the length in bits of the C long long data type supported by the hardware used to test generated code.

For more information, see Model Parameters.

Auto-insertion of rate transition block

Simulink can now automatically handle rate transitions for periodic tasks even if the Greatest Common Divisor (GCD) rate is not in the model. Previously, Simulink required that a block with the GCD sample rate be present in the model to allow automatic insertion of the rate transition block. For more information, see Automatically handle rate transition for data transfer.

Compiled sample time for multi-rate blocks returns cell array of all sample times

For multi-rate blocks (including subsystems), Simulink now returns the compiled sample time for the block as a cell array of all the sample rates present in the block. Therefore, to query the **CompiledSampleTime** and determine if a subsystem is multi-rate, you no longer need to loop over all the blocks inside a subsystem or build up a list of sample times for the subsystem. The subsystem block **CompiledSampleTime** parameter now contains that information.

Previously, Simulink returned only the greatest common divisor (GCD) of all sample rates present in the block. This might cause a problem if the GCD rate did not exist in the model, thereby causing Simulink Coder to generate empty functions.

To obtain the complete list of sample times in a block, use the following command:

```
theBlockSampleTimes = get_param(SubsystemBlock, 'CompiledSampleTime');
```

This is more efficient than querying the sample time for each block and sorting the list.

For more information, see Sample Times in Subsystems.

Version History

Because Simulink now returns the `CompiledSampleTime` parameter as a cell array of pairs of doubles (instead of a pair of doubles), some compatibility issues can occur.

Consider a variable `blkTs`, which has been assigned the compiled sample time of a multi-rate block.

```
blkTs = get_param(block, 'CompiledSampleTime');
```

Here are some examples in which the original code works only if `blkTs` is a pair of doubles and the block is a single-rate block:

- Example 1

```
if isinf(blkTs(1))
    disp('found constant sample time')
end
```

Since `blkTs` is now a cell array, Simulink gives the following error message:

Undefined function 'isinf' for input arguments of type 'cell'

Instead, use this code, for which `blkTs` can be a cell array or a pair of doubles.

```
if isequal(blkTs, [inf,0])
    disp('found constant sample time')
end
```

- Example 2

```
if all(blkTs == [-1,-1])
    disp('found triggered sample time')
end
```

For the above example, since `blkTs` is now a cell array, Simulink gives the following error:

Undefined function 'eq' for input arguments of type 'cell'

Instead, use this code, for which `blkTs` can be a cell array or a pair of doubles.

```
if isequal(blkTs, [-1,-1])
    disp('found triggered sample time')
end
```

- Example 3

```
if (blkTs(1) == -1)
    disp('found a triggered context')
end
```

Again, since `blkTs` is now a cell array, Simulink gives the following error:

Undefined function 'eq' for input arguments of type 'cell'

Instead, use this code.

```
if ~iscell(blkTs)
    blkTs = {blkTs};
end
for idx = 1:length(blkTs)
```

```
    thisTs = blkTs{idx};  
    if (thisTs(1) == -1)  
        disp('found a triggered context')  
    end  
end
```

The above code checks for a triggered type sample time (triggered or async). In cases in which a block has constant sample time (`[inf,0]`) in addition to triggered or async or when a block has multiple async rates, this alternative detects sample times in all such cases.

Improvement to model reference parallel build check in Performance Advisor

The model reference parallel build check has been improved to take into account parallel build overhead time when estimating overall overhead time. For more information, see [Check model reference parallel build](#).

Improved readability in Performance Advisor reports

Tables in the HTML report generated by Performance Advisor now appear in a new format for improved clarity and readability.

Simulation Data Inspector launch using `simplot` command

The `simplot` command now launches the Simulation Data Inspector. The `simplot` command is no longer supported and redirects to the Simulation Data Inspector.

Use the Simulation Data Inspector to inspect and compare signal data from simulations. For more information on using the Simulation Data Inspector, see [Validate System Behavior](#).

Version History

In R2013b, the `simplot` command launches the Simulation Data Inspector, and the return arguments to the function are empty handles. In previous releases, the `simplot` command returned handles to the graphics figure.

Project and File Management

Impact analysis by exploring modified or selected files to find dependencies

In Simulink Project, use impact analysis to find the impact of changing particular files. You can investigate dependencies visually to explore the structure of your project. You can analyze selected or modified files to find their required and impacted files. Visualize changes and dependencies in the Impact graph.

Impact analysis can show you how a change will impact other files before making the change. For example:

- Investigate the potential impact of a change in requirements by finding the design files linked to the requirements document.
- Investigate change set impact by finding upstream and downstream dependencies of modified files before committing the changes. This can help you identify design and test files that might need modifications and find the tests you need to run.

You can label, open, or export the files found by impact analysis. You can use the impact analysis results to create reports or artifacts describing the impact of a change.

For details, see Perform Impact Analysis.

Option to export impact analysis results to the workspace, batch processing, or image files

In Simulink Project, after performing impact analysis, you can export the results to a workspace variable, to batch processing, or to image files. This enables further processing or archiving of impact analysis results.

For details, see Export Impact Results.

Identification of requirements documents during project dependency analysis

In Simulink Project, Dependency Analysis searches for requirements documents linked using the Requirements Management Interface. You can view linked requirements documents in Simulink Project and navigate to and from the linked documents. You can only create or edit Requirements Management links if you have Simulink Verification and Validation.

Previously, you could find requirements documents only for a single model by generating a manifest. Now you can find requirements documents linked anywhere in your project.

For details, see Choose Files and Run Dependency Analysis.

Simplified label creation by dragging a label onto files in any view

In Simulink Project, add labels to files by dragging a label onto files. Create labels and categories of labels from any view. These features simplify label creation and application without any need to

switch view. You can view project labels at the same time as project files or dependency analysis results.

Previously you could create new labels only at the Labels node, and you could apply a label to a single file only by opening a dialog box.

For details, see [Create Labels](#) and [Add Labels to Files](#).

Shortcut renaming, grouping, and execution from any view using the Toolstrip

Simulink Project tools for managing shortcuts enable you to:

- Create shortcut groups to organize shortcuts by type, for example, to separate shortcuts for loading data, opening models, generating code, and running tests.
- Annotate shortcuts to make their purpose visible, without changing the file name or location of the script or model the shortcut points to, for example, to change a cryptic file name to a useful string for the shortcut name.
- Execute project shortcuts from any view in Simulink Project using the toolstrip.

These features simplify shortcut management, allowing you to use shortcuts while viewing your project files or dependency analysis results.


Previously you could view and execute shortcuts only at the Shortcuts node. Now you can find and execute shortcuts whenever they are needed in the projects workflow without switching the view.

For details, see [Create Shortcuts to Frequent Tasks](#), [Annotate Shortcuts to Use Meaningful Names](#), and [Use Shortcuts to Find and Run Frequent Tasks](#).

Data Management

Streamlined selection of one or more signals for signal logging

In the Simulink Editor, you no longer need to open the Signal Properties dialog box to enable signal logging for a signal. Instead:

- 1 Select one or more signals.
- 2 Click the **Record** button arrow  and click **Log/Unlog Selected Signals**.

Simplified modeling of single-precision designs

In R2013b, you can model single-precision designs more easily.

- There is now a model-wide setting that you can use to specify that Simulink use singles as the default type during data type propagation. See “New option to set default for underspecified data types” on page 19-13.
- Simulink avoids the use of double data types to achieve strict single design for operations between singles and integers. In previous releases, Fixed-Point Designer used double data types in intermediate calculations for higher precision. You might see a difference in numerical behavior of an operation between earlier releases and R2013b. See “Operations between singles and integer or fixed-point data types avoid use of doubles” on page 19-13.
- There is a new Model Advisor check that detects the presence of double data types in a model. See Identify questionable operations for strict single-precision design.

New option to set default for underspecified data types

There is now a model-wide setting to specify the data type to use if Simulink cannot infer the type of a signal during data type propagation. You can now choose to set the default value for underspecified data types to double or single for simulation and code generation. For embedded designs that target single-precision processors, set the **Default for underspecified data type** configuration parameter to **single** to avoid introducing double data types. For more information, see Default for underspecified data type.

Operations between singles and integer or fixed-point data types avoid use of doubles

Simulink now supports strict single-precision algorithms for mixed single and integer data types for cast and math operations. Operations, such as cast, multiplication and division, use single-precision math instead of introducing higher precision doubles for intermediate calculations in simulation and code generation. You no longer have to explicitly cast integers or fixed-point inputs of these operations to single precision. To detect the presence of double data types in your model to help you refine your mixed single and integer design, use the Model Advisor **Identify questionable operations for strict single-precision design** check.

Version History

In R2013b, for both simulation and code generation, Simulink avoids the use of double data types to achieve strict single design for operations between singles and integers. In previous releases, Simulink used double data types in intermediate calculations for higher precision. You might see a difference in the numerical behavior of an operation between earlier releases and R2013b.

For example, when the cast is from a fixed-point or integer data type to single or vice versa, the type for intermediate calculations can have a big impact on the numerical results. Consider:

- Input type: `ufix128_En127`
- Input value: 1.99999999254942 — Stored integer value is $(2^{128} - 2^{100})$
- Output type: `single`

Release	Calculation performed by Fixed-Point Designer	Output Result	Design Goal
R2013b	<code>Y = single(2^-127) * single(2^128-2^100)</code> <code>= single(2^-127) * Inf</code>	Inf	Strict singles
Previous releases	<code>Y = single(double(2^-127) * double(2^128 - 2^100))</code> <code>= single(2^-127 * 3.402823656532e+38)</code>	2	Higher precision intermediate calculation

There is also a difference in the generated code. Previously, Fixed-Point Designer allowed the use of doubles in the generated code for a mixed multiplication that used single and integer types.

```
m_Y.Out1 = (real32_T)((real_T)m_U.In1*(real_T)m_U.In2);
```

In R2013b, it uses strict singles.

```
m_Y.Out1=(real32_T)m_U.In1*m_U.In2;
```

To revert to the numerical behavior of previous releases, insert explicit casting from integers to doubles for the inputs of these operations.

Connection status visualization and connection method customization for root inport mapping

The Root Inport Mapping dialog box has the following updates:

- The Root Inport Mapping dialog box now has connection status visualization. If the Root Inport Mapping status area lists warnings or failures, the software highlights the Inport block associated with the data. Warnings display as yellow Inport blocks outlined in orange, failures display as yellow Inport blocks outlined with bold red, and successes display as normal Inport blocks outlined with blue. Selecting the line item highlights the associated Inport block. For more information, see [Understanding Mapping Results](#).
- The root inport mapping capability has a new function, `getSlRootInportMap`. This function provides a new connection method for custom mappings. Use this function when you have a mapping method that is similar to, but not exactly the same as, one of the existing Simulink root inport mapping methods.

Conversion of numeric variables into Simulink.Parameter objects

You can now convert a numeric variable into a `Simulink.Parameter` object using a single step.


```
myVar = 5; % Define numerical variable in base workspace
% Create data object and assign variable value to data object value
myObject = Simulink.Parameter(myVar);
```

Previously, you did this conversion using two steps.

```
myVar = 5; % Define numerical variable in base workspace
myObject = Simulink.Parameter; % Create data object
myObject.Value = myVar; % Assign variable value to data object value
```

Model Explorer search options summary hidden by default

To provide more space for displaying search results, the Model Explorer **Search Results** pane hides the summary of the search options (such as search criteria) that you used.

To view the search options summary, at the top of the **Search Results** pane, click the **Expand Search Results** button .

Simulink.DualScaledParameter class

The new `Simulink.DualScaledParameter` class extends the capabilities of the `Simulink.Parameter` class. You can define a parameter object that stores two scaled values of the same physical value. Suppose you want to store temperature measurements as Fahrenheit or Celsius values. You can define a parameter that stores the temperature in either measurement scale with a computational method to convert between the dual-scaled values.

You can use `Simulink.DualScaledParameter` objects in your model for both simulation and code generation. The parameter computes the internal value before simulation or code generation via the computational method, which can be a first-order rational function. This offline computation results in leaner generated code.

For more information, see `Simulink.DualScaledParameter`.

Legacy data type specification functions return numeric objects

In previous releases, the following functions returned a MATLAB structure describing a fixed-point data type:

- `float`
- `sfix`
- `sfrac`
- `sint`
- `ufix`
- `ufrac`
- `uint`

Effective R2013b, they return a `Simulink.NumericType` object. If you have existing models that use these functions as parameters to dialog boxes, the models continue to run as before, and there is no need to change any model settings.

These functions do not offer full Data Type Assistant support. To benefit from this support, use `fixdt` instead.

Function	Return Value in Previous Releases – MATLAB structure	Return Value Effective R2013b – NumericType
float('double')	Class: 'DOUBLE'	DataTypeMode: 'Double'
float('single')	Class: 'SINGLE'	DataTypeMode: 'Single'
sfix(16)	Class: 'FIX' IsSigned: 1 MantBits: 16	DataTypeMode: 'Fixed-point: unspecified scaling' Signedness: 'Signed' WordLength: 16
ufix(7)	Class: 'FIX' IsSigned: 0 MantBits: 7	DataTypeMode: 'Fixed-point: unspecified scaling' Signedness: 'Unsigned' WordLength: 7
sfrac(33,5)	Class: 'FRAC' IsSigned: 1 MantBits: 33 GuardBits: 5	DataTypeMode: 'Fixed-point: binary point scaling' Signedness: 'Signed' WordLength: 33 FractionLength: 27
ufrac(44)	Class: 'FRAC' IsSigned: 0 MantBits: 44 GuardBits: 0	DataTypeMode: 'Fixed-point: binary point scaling' Signedness: 'Unsigned' WordLength: 44 FractionLength: 44
sint(55)	Class: 'INT' IsSigned: 1 MantBits: 55	DataTypeMode: 'Fixed-point: binary point scaling' Signedness: 'Signed' WordLength: 55 FractionLength: 0
uint(77)	Class: 'INT' IsSigned: 0 MantBits: 77	DataTypeMode: 'Fixed-point: binary point scaling' Signedness: 'Unsigned' WordLength: 77 FractionLength: 0

Version History

MATLAB Code

MATLAB code that depends on the return arguments of these functions being a structure with fields named `Class`, `MantBits` or `GuardBits`, no longer works correctly. Change the code to access the appropriate properties of a `NumericType` object, for example, `DataTypeMode`, `Signedness`, `WordLength`, `FractionLength`, `Slope`, and `Bias`.

C Code

Update C code that expects the data type of parameters to be a legacy structure to handle `NumericType` objects instead. For example, if you have S-functions that take legacy structures as parameters, update these S-functions to accept `NumericType` objects.

MAT-files

Effective in R2013b, if you open a Simulink model that uses a MAT-file that contains a data type specification created using the legacy functions, the model uses the same data types and behaves in the same way as in previous releases but Simulink generates a warning. To eliminate the warning, recreate the data type specifications using `NumericType` objects, and save the MAT-file.

You can use the `fixdtupdate` function to update a data type specified using the legacy structure to use a `NumericType`. For example, suppose you saved a data type specification in a MAT-file in a previous release as follows:

```
oldDataType = sfrac(16);  
save myDataTypeSpecification oldDataType
```

Use `fixdtUpdate` to recreate the data type specification to use `NumericType`:

```
load DataTypeSpecification  
fixdtUpdate(oldDataType)
```

```
ans =
```

```
    NumericType with properties:
```

```
    DataTypeMode: 'Fixed-point: binary point scaling'  
    Signedness: 'Signed'  
    WordLength: 16  
    FractionLength: 15  
    IsAlias: 0  
    DataScope: 'Auto'  
    HeaderFile: ''  
    Description: ''
```

For more information, at the MATLAB command line, enter:

```
fixdtUpdate
```

Root Inport Mapping Error Messages

The error message handling has been improved for root import mapping at the consistency check phase. The Simulation Diagnostics Viewer now displays these error messages.

Root inport mapping example

The `Using Simulink Mapping Modes in Custom Mapping Functions` example shows how to use the `getSlRootInportMap` function to create a mapping object. This example uses a mapping mode similar to the Simulink block name mapping mode.

Connection to Educational Hardware

Ability to run models on target hardware from the Simulink toolbar

You can use the **Deploy to Hardware** or **Run** button on the Simulink toolbar to run models on target hardware.

To enable these buttons, first configure your model to use for Run on Target Hardware.

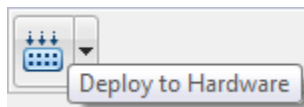
- 1 If you have not done so already, use `supportPackageInstaller` and install a Simulink support package.
- 2 In a model, select **Tools > Run on Target Hardware > Prepare to Run**.

The Configuration Parameters dialog box opens and displays the Run on Target Hardware pane.

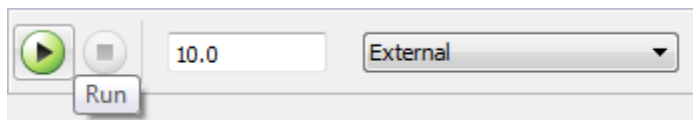
- 3 Set the **Target hardware** parameter to match your target hardware.

After configuring the model, you can use either button:

- To deploy the model, click **Deploy to Hardware**. The model runs as a standalone application on the target hardware.



- To use External mode, set **Simulation mode** to `External`, and then click **Run**. In External mode, you can tune parameters and monitor a model running on your target hardware.



Note This feature works only with Simulink support packages. The names of these support packages begin with “Simulink Support Package for...”

For more information, see [What is “Run on Target Hardware”?](#)

Note Some target hardware does not support External mode. For more information, consult the documentation for the specific target hardware.

Support for Arduino hardware available on Mac OS X

You can use Simulink Support Package for Arduino Hardware on Apple Mac OS X platform. This includes support for Arduino Mega 2560, Arduino Uno, Arduino Nano, and Arduino Ethernet Shield hardware.

To use this feature, install the Simulink Support Package for Arduino Hardware, as described in [Install Support for Arduino Hardware](#).

Support for Arduino Ethernet Shield and Arduino Nano 3.0 hardware

You can use Simulink Support Package for Arduino Hardware with the Arduino Ethernet Shield hardware and Arduino Nano 3.0 hardware. The block library for Arduino Ethernet Shield hardware includes TCP/IP and UDP blocks that enable you to design network-enabled embedded systems.

To use this feature, install the Simulink Support Package for Arduino Hardware, as described in [Install Support for Arduino Hardware](#).

The block library for the Arduino Ethernet Shield hardware includes the following blocks:

- [Arduino TCP/IP Send](#) and [Arduino TCP/IP Receive](#) enable TCP/IP communications with networked devices using an Ethernet port.
- [Arduino UDP Send](#) and [Arduino UDP Receive](#) enable UDP communications with networked devices using an Ethernet port.

For more information about this feature, see the [Arduino Hardware](#) topic.

Signal Management

Port number display to help resolve error messages

You can display the port numbers in a block that an error message highlights by hovering over the block input or output ports. For details, see [Display Port Numbers When Addressing Errors](#).

Enforced bus diagnostic behavior

For models that use bus primitives, set the **Configuration Parameters > Diagnostics > Connectivity > Mux blocks used to create bus signals** diagnostic to `error`. Bus primitives include the Bus Creator, Bus Selector, and Bus Assignment blocks, as well as bus objects.

R2013b enforces setting the diagnostic to `error`. Benefits of setting this diagnostic to `error` include:

- Prevents introducing Mux/bus mixtures into your model. For information about the problems with that mixture, see [Prevent Bus and Mux Mixtures](#).
- Improves handling of feedback loops.
- Supports important signal features, including:
 - Nonzero initialization of bus signals
 - Bus support for blocks such as Constant, Data Store Memory, From File, From Workspace, To File, and To Workspace
 - Signal label propagation enhancements
 - Arrays of buses

Version History

When you compile (update or simulate) a model in R2013b, Simulink analyzes the model to determine the extent to which the model uses Mux blocks to create bus signals. The table describes how Simulink handles different kinds of models at compile time, in relationship to the diagnostic.

Model Configuration	Simulink Actions at Compile Time	Your Required Actions
No bus primitives or muxes that involve different data types	Displays no warning, error, or upgrade message related to this diagnostic Sets diagnostic to <code>error</code> when you save the model	None
Bus primitives and diagnostic is set to <code>error</code>	None	None
No bus primitives, but one or more mux with different data types	Displays an error message prompting you to launch the Upgrade Advisor	Run the Upgrade Advisor and compile your model.

Model Configuration	Simulink Actions at Compile Time	Your Required Actions
Bus primitives and diagnostic set to Warning or None	Displays an error message prompting you to launch the Upgrade Advisor	Run the Upgrade Advisor and set the diagnostic to error.

Block Enhancements

Improved performance of LUT block intermediate calculations

Blocks in the Lookup Tables library have a new internal rule for fixed-point data types to enable faster hardware instructions for intermediate calculations (with the exception of the Direct Lookup Table (n-D), Prelookup and Lookup Table Dynamic blocks). To use this new rule, select **Speed** for the **Internal Rule Priority** parameter in the dialog box. Select **Precision** to use the internal rule in R2013a.

Name changes that unify storage class parameters

The following parameters have been renamed to unify the names of storage class parameters.

Old Name	New Name
RTWStateStorageClass	StateStorageClass
CodeGenStateStorageClass	StateStorageClass

The blocks affected by this name change are Delay, Unit Delay, Memory, Discrete State-Space, Discrete Zero-Pole, Discrete Filter, Discrete Transfer Function, Data Store Memory, and Discrete-Time Integrator.

Version History

Simulink will not support the old parameter names in a future release. If you use the old parameter names in your code to programmatically set parameter values, replace them with the new names.

Warnings when using old parameter names with spaces

Parameter names that include spaces (including block type names) from Simulink Version 1.3 (1994) and earlier now warn. Do not use parameter names with spaces.

Version History

Old parameter names that include spaces (including block type names) from Simulink Version 1.3 (1994) and earlier now cause a warning if you use them with `get_param`, `set_param`, `find_system`, or `add_block`. Messages direct you to the new parameter name if it exists.

Strictly monotonically increasing time values on Repeating Sequence block

The **Time values** parameter in the Repeating Sequence block can take only strictly monotonically increasing values. In R2013a, this parameter could take duplicate time values. Run the Upgrade Advisor check “Check model for known block upgrade issues” to identify this issue in your model.

pow function in Math function block that supports Code Replacement Library (CRL)

The Math Function block supports the Code Replacement Library for the pow function, which is useful if you use code generation for models that use this function. For more information, see the Math Function block.

Continuous Linear Block improvements, such as diagnostics, readability, and stricter checks

The Continuous library blocks State-Space, Transfer Fcn, and Zero-Pole have been enhanced as follows:

- Improved error diagnostics.
- More readable generated code for large matrices.
- Stricter checks for changes made to tunable parameters at run time.
- Unused tunable parameters removed from generated code.
- Ability for the State-Space block to act as a source block. This change enables the modeling of autonomous linear state-space systems.
- The following changes in the Transfer Fcn block:
 - Computation of time-domain realization has been enhanced for better performance and error diagnostics
 - Support of sparse Jacobian computation for reduced memory usage, better implicit solver support, and improved structural analysis

MATLAB Function Blocks

Code generation for Statistics Toolbox and Phased Array System Toolbox

Code generation now supports more than 100 Statistics Toolbox™ functions. For implementation details, see Statistics Toolbox Functions.

Code generation now supports most of the Phased Array System Toolbox™ functions and System objects. For implementation details, see Phased Array System Toolbox Functions and Phased Array System Toolbox System Objects.

Toolbox functions for code generation

For implementation details, see Functions Supported for C/C++ Code Generation — Alphabetical List.

Data Type Functions

- `narginchk`

Programming Utilities

- `mfilename`

Specialized Math

- `psi`

Computer Vision System Toolbox Classes and Functions

- `extractFeatures`
- `detectSURFFeatures`
- `disparity`
- `detectMSERFeatures`
- `detectFASTFeatures`
- `vision.CascadeObjectDetector`
- `vision.PointTracker`
- `vision.PeopleDetector`
- `cornerPoints`
- `MSERRegions`
- `SURFPoints`

External C library integration using `coder.ExternalDependency`

You can define the interface to external code using the new `coder.ExternalDependency` class. Methods of this class update the compile and build information required to integrate the external code with MATLAB code. In your MATLAB code, you can call the external code without needing to update build information. See `coder.ExternalDependency`.

Updating build information using `coder.updateBuildInfo`

You can use the new function `coder.updateBuildInfo` to update build information. For example:

```
coder.updateBuildInfo('addLinkFlags','/STACK:1000000');
```

adds a stack size option to the linker command line. See `coder.updateBuildInfo`.

Conversion of MATLAB expressions into C constants using `coder.const`

You can use the new function `coder.const` to convert expressions and function calls to constants at compile time. See `coder.const`.

Highlighting of constant function arguments in the compilation report

The compilation report now highlights constant function arguments and displays them in a distinct color. You can display the constant argument data type and value by placing the cursor over the highlighted argument. You can export the constant argument value to the base workspace where you can display detailed information about the argument.

For more information, see [Viewing Variables in Your MATLAB Code](#).

`coder.target` syntax change

The new syntax for `coder.target` is:

```
tf = coder.target('target')
```

For example, `coder.target('MATLAB')` returns true when code is running in MATLAB. See `coder.target`.

You can use the old syntax, but consider changing to the new syntax. The old syntax will be removed in a future release.

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

The Windows 64-bit platform now includes LCC-win64 as the default compiler for running simulations. You no longer have to install a separate compiler for simulation of MATLAB Function blocks.

LCC-win64 is used only when another compiler is not configured in MATLAB.

Modeling Guidelines

Modeling guidelines for high-integrity systems

The following are new modeling guidelines to develop models and generate code for high-integrity systems:

- hisl_0024: Inport interface definition
- hisl_0025: Design min/max specification of input interfaces
- hisl_0026: Design min/max specification of output interfaces
- hisl_0027: Usage of Signed Square Root blocks
- hisl_0028: Usage of Reciprocal Square Root blocks

Model Advisor

Collapsible content within Model Advisor reports

The Model Advisor report now collapses the results, making it easier to navigate through the report.

Reorganization of Model Advisor checks

Checks previously provided with Simulink in the Model Advisor Embedded Coder folder are now available with either Simulink Coder or Embedded Coder. For a list of checks available with each product, see:

- Simulink Coder Checks
- Embedded Coder Checks

Check for strict single precision usage

The new Identify questionable operations for strict single-precision design check identifies blocks that introduce double-precision operations. Use this check to help you refine your strict single design.

R2013a

Version: 8.1

New Features

Bug Fixes

Version History

New Simulink Editor

Reordering of tabs in tabbed windows

You can rearrange the order of tabs in a Simulink Editor window. For example, if you have opened several subsystems, to make it easier to access the tab for the last subsystem that you opened, you could select that tab and drag it to make it the first tab on the left.

Scalable vector graphics for mask icons

Images in .svg format can be used with the `image` command for creating block mask images in Simulink.

Simulation Stepper Default Value Change

The default value of the **Interval between stored back steps** stepping option is now 10. In the previous release, this value was 1.

Component-Based Modeling

Direct active variant control via logical expressions

Variant control can accept a variant object or any condition expression. Defining variant objects for use in modeling variants requires saving variant objects external to a model file. Using direct expressions in variant control reduces complexity. For more information, see Variant Systems.

Live update for variant systems and commented-out blocks

You can control active variants by setting variant controls. When you modify active variants, the display refreshes automatically; you do not need to use Update Diagram. You can also comment out blocks in your model to exclude them from simulation. For more information, see Variant Systems.

Masking of linked library blocks

You can now create masks on linked blocks. Masking a linked block (one that already has a mask) creates a stack of masks. Simulink libraries can contain blocks that become library links when copied to a model. Masking such blocks previously involved wrapping them inside a subsystem and creating a mask on that subsystem. Masking linked blocks instead uses less memory and management overhead. For more information, see Masking Linked Blocks.

Target profiling for concurrent execution to visualize task execution times and task-to-core assignment

A new pane, Profile Report on the Concurrent Execution dialog box, enables the visualization of task execution times and task-to-core affinitization. You can profile using Simulink Coder (GRT) and Embedded Coder (ERT) targets. For more information, see Profile and Evaluate.

The Modeling Concurrent Execution on Multicore Targets example has been updated to reflect the use of this capability.

Incremental block-to-task mapping workflow support enabled by automatic block-to-task assignment for multicore execution on embedded targets

This support allows for the partial mapping of blocks to tasks, allowing you to specify task assignments only for the blocks you are interested in. For more information, see Analyze Baseline.

With this change, the **Map blocks to tasks** pane no longer has a **Get Default Configuration** button.

PIL and SIL modes for concurrent execution

The following simulation modes are now supported for concurrent execution:

- Processor-in-the-loop (PIL)
- Software-in-the-loop (SIL)

Parameterized task periods for concurrent execution

You can now parameterize task periods for concurrent execution using variables from the MATLAB base workspace.

Relaxed configuration parameter setting requirements

Configuration requirements for model referencing have been removed. The following parameters no longer need to be the same for top and referenced models:

- **Templates > Target operating system**
- **Solver > Allow tasks to execute concurrently on target**

Connection to Educational Hardware

Support for Gumstix Overo hardware

Run Simulink models on Gumstix Overo hardware. Tune parameter values in the model, and receive data from the model, while it is running on Gumstix Overo hardware.

Use the **Simulink Support Package for Gumstix Overo Hardware** block library to access Gumstix Overo peripherals:

- Overo UDP Receive and Overo UDP Send — Communicate with networked devices using an Ethernet port.
- Overo ALSA Audio Capture — Capture audio from sound card using ALSA
- Overo ALSA Audio Playback — Send audio to sound card for playback using ALSA
- Overo V4L2 Video Capture — Capture video from USB camera using V4L2
- Overo SDL Video Display — Display video using SDL
- Overo GPIO Read and Overo GPIO Write — Communicate with external devices using GPIO pins. The blocks provide diagrams that help you locate specific GPIO pins.
- Overo LED — Illuminate built-in LEDs on your target hardware. The block provides a diagram that helps you locate specific LEDs.
- Overo eSpeak Text to Speech — Convert text to speech for output to the default audio device.

To get these capabilities and the block library, enter `targetinstaller` in a MATLAB Command Window. Then, use Support Package Installer to install the support package for Gumstix Overo hardware. For more information, see the Gumstix Overo topic.

After installing the support package, you can open the block library by entering `overolib` in the MATLAB Command Window. The Simulink Support Package for Gumstix Overo Hardware block library is also available in the Simulink Library Browser.

Support for Raspberry Pi hardware

Run Simulink models on Raspberry Pi hardware. Tune parameter values in the model, and receive data from the model, while it is running on Raspberry Pi hardware.

Use the **Simulink Support Package for Raspberry Pi Hardware** block library to access Raspberry Pi peripherals:

- Raspberry Pi UDP Receive and Raspberry Pi UDP Send — Communicate with networked devices using an Ethernet port.
- Raspberry Pi ALSA Audio Capture — Capture audio from sound card using ALSA
- Raspberry Pi ALSA Audio Playback — Send audio to sound card for playback using ALSA
- Raspberry Pi V4L2 Video Capture — Capture video from USB camera using V4L2
- Raspberry Pi SDL Video Display — Display video using SDL
- Raspberry Pi GPIO Read and Raspberry Pi GPIO Write — Communicate with external devices using GPIO pins. The blocks provide diagrams that help you locate specific GPIO pins.
- Raspberry Pi LED — Illuminate built-in LEDs on your target hardware. The block provides a diagram that helps you locate specific LEDs.

- Raspberry Pi eSpeak Text to Speech — Convert text to speech for output to the default audio device.

To get these capabilities and the block library, enter `targetinstaller` in a MATLAB Command Window. Then, use Support Package Installer to install the support package for Raspberry Pi hardware. For more information, see the Raspberry Pi topic.

After installing the support package, you can open the block library by entering `pilib` in the MATLAB Command Window. The Simulink Support Package for Raspberry Pi Hardware block library is also available in the Simulink Library Browser.

Blocks for GPIO, LED, and eSpeak Text to Speech on BeagleBoard

The block libraries for BeagleBoard hardware include four new blocks:

- BeagleBoard GPIO Read and BeagleBoard GPIO Write — Communicate with external devices using GPIO pins.
- BeagleBoard LED — Illuminate built-in LEDs on your target hardware.
- BeagleBoard eSpeak Text to Speech — Convert text to speech for output to the default audio device.

To get these blocks, enter `targetinstaller` in a MATLAB Command Window. Then, use Support Package Installer to install the support package for BeagleBoard hardware. For more information, see the BeagleBoard topic.

After installing the support package, you can open the updated block library. In a MATLAB Command Window, enter `beagleboardlib`. You can also access these block libraries through the Simulink Library Browser.

Blocks for GPIO, LED, and eSpeak Text to Speech on PandaBoard

The block libraries for PandaBoard hardware include four new blocks:

- PandaBoard GPIO Read and PandaBoard GPIO Write — Communicate with external devices using GPIO pins.
- PandaBoard LED — Illuminate built-in LEDs on your target hardware.
- PandaBoard eSpeak Text to Speech — Convert text to speech for output to the default audio device.

To get these blocks, enter `supportPackageInstaller` in a MATLAB Command Window. Then, use Support Package Installer to install the support package for PandaBoard hardware. For more information, see the PandaBoard topic.

After installing the support package, you can open the updated block library. In a MATLAB Command Window, enter `pandaboardlib`. You can also access these block libraries through the Simulink Library Browser.

Blocks for Compass and IR Receiver sensors on LEGO MINDSTORMS NXT

The block library for LEGO MINDSTORMS NXT hardware includes two new blocks:

- LEGO MINDSTORMS NXT Compass Sensor — Read the magnetic heading of the compass sensor.
- LEGO MINDSTORMS NXT IR Receiver Sensor — Receive IR signals from of a LEGO Power Functions IR Speed Remote Control.

To get these blocks, in a MATLAB Command Window, enter `targetinstaller`. Then, use Support Package Installer to install the support package for LEGO MINDSTORMS NXT hardware. For more information, see the LEGO MINDSTORMS NXT topic.

After installing the support package, you can open the updated block library. In a MATLAB Command Window, enter `legonxtlib`. The block library is also available in the Simulink Library Browser.

Project and File Management

Simplified scripting interface for automating Simulink Project tasks

Simulink Projects provide a new API with shorter commands for automating project tasks with file and label management.

See Simulink Projects for links to project functions.

Version History

The new Simulink Projects API replaces the class `Simulink.ModelManagement.Project.CurrentProject` and its methods. `Simulink.ModelManagement.Project.CurrentProject` will be removed in a future release. Instead, use `simulinkproject` and related functions for project manipulation.

Option to use elements from multiple templates when creating a new project

When you create a new project, you can select multiple templates to apply. You can select templates directly on the New Project menu.

See Use Templates to Create Standard Project Settings.

Saving and reloading of dependency analysis results

The Simulink Project Tool remembers the results of previous dependency analysis and saves the results with your project. You can view your previous results without having to run time-consuming analysis again. You can also save and reload previous results.

See Analyze Project Dependencies.

Robust loading of projects with conflicted metadata project definition files

The Simulink Project Tool now has tolerance for loading projects with conflicts in metadata project definition files. You can load the conflicted project and resolve the conflicts. In previous releases you could not load a project with conflicts in the metadata.

New project preferences to control logging and warnings

Simulink Project Tool has a new Preferences dialog where you can control options for logging and warnings.

Data Management

Fixed-Point Advisor support for model reference

The Fixed-Point Advisor now performs checks on referenced models. It checks the entire model reference hierarchy against fixed-point guidelines. The Advisor also provides guidance about model configuration settings and unsupported blocks to help you prepare your model for conversion to fixed point.

Arrays of buses loading and logging

You can log array of buses signal data using signal logging. For details, see step 5 in Set Up a Model to Use Arrays of Buses.

You can load array of buses data to a root Inport block. For details, see Import Array of Buses Data.

Root Inport Mapping tool changes

The following are changes to the Root Inport Mapping tool:

- The Root Inport Mapping tool can now import the following additional data formats from a MAT-file:
 - Array of buses
 - Asynchronous function-call signals
- A new button, **Map Signals**, has been added to the **Mapping Mode** section of the tool. Use this button to map the data to the root-level ports. Then, use the **Apply** or **OK** buttons to commit the changes to the model.
- The Status area of the tool has been visually updated to better display the mapping status.
- A new function, `getInputString`, enables you to create a comma-separated list of variables to be mapped.

For more information, see Import and Map Data to Root-Level Inports.

New Root Inport Mapping Examples

The following examples show how to use the Root Inport Mapping dialog box:

- Converting Test Harness Model to Harness-Free Model

The `Converting Harness-Driven Models to Use Harness-Free External Inputs` example shows how to convert a harness model using Signal Builder block as an input to a harness-free model with root inports.

- Custom Mapping for External Inputs of a Model

This example, available from **Simulink > Simulink Examples > Modeling Features**, shows how to create a custom mapping function for mapping data to root-level input ports.

Level-1 data classes not supported

Simulink no longer supports level-1 data classes. Extend Simulink data classes using MATLAB class syntax instead.

For more information, see [Define Data Classes](#).

To upgrade your level-1 data classes, see [Upgrade Level-1 Data Classes](#).

Version History

When you upgrade your level-1 data classes, the way that MATLAB code is generated and model files are loaded remains the same. However, you may encounter errors if your code includes the following capabilities specific to level-1 data classes:

- Inexact property names such as `a.datatype` instead of the stricter `a.DataType`.
- Vector matrix containing `Simulink.Parameter` and `Simulink.Signal` data objects. Previously, using level-1 data classes, you could define a vector matrix `v` as follows:

```
a = Simulink.Signal;  
b = Simulink.Parameter;  
v = [a b];
```

Such mixed vector matrices are no longer supported.

In these cases, modify your code to replace these capabilities with those supported by MATLAB class syntax. For more information on how to make these replacements, see [Begin Using Object-Oriented Programming in MATLAB](#) documentation.

Simulink data type classes do not support inexact enumerated property value matching

Previously, Simulink data type classes permitted partial enumerated property value matching and did not enforce case sensitivity. For example, after creating a `Simulink.NumericType` data type object

```
a = Simulink.NumericType;
```

you could set the value of property `DataTypeMode` of the object by using one of the following commands:

- Partial matching of enumerated property value:

```
a.DataTypeMode = 's';
```

Here, 's' is equivalent to 'Single'.

- Case-insensitive matching of enumerated property value:

```
a.DataTypeMode = 'Fixed-Point: binary point scaling';
```

Here, 'Fixed-Point: binary point scaling' is equivalent to 'Fixed-point: binary point scaling'.

Now, Simulink data type classes do not permit partial or case-insensitive matches of enumerated property values.

Version History

You may encounter errors or warnings if your code relies on setting enumerated property values of data type objects using inexact matches. In these case, replace your code so that these property values are set using exact matches. For example, after creating a `Simulink.NumericType` data type object

```
b = Simulink.NumericType;
```

set the value of property `DataTypeMode` using the following command:

```
b.DataTypeMode = 'Single';
```

Tip Tab completion works with enumerated properties. For example, if you enter a property name followed by an equal sign, MATLAB pops up a selection box with a list of values for that property.

Simulation Analysis and Performance

Simulation Performance Advisor report that shows both check results and actions taken

Performance Advisor HTML report now include actions in addition to checks. For more information, see View Performance Advisor Reports.

Improved simulation performance when stepping back is enabled

The performance of stepping back using the Simulation Stepper has been improved. For more information on the Simulation Stepper, see Simulation Stepping.

Simulation Data Inspector run-configuration options for names and placement in run list

When managing many runs in the Simulation Data Inspector, you can specify whether to add new runs at the top or bottom of the Signal Browser table. In addition, you can customize automatic naming of new runs added to the Simulation Data Inspector. For more information, see Run Management Configuration.

Arrays of buses displayed in Simulation Data Inspector

The Simulation Data Inspector records logged arrays of buses. After recording an array of buses, the Simulation Data Inspector can display this data in a hierarchical format.

Simulation Data Inspector overwrite run specification

In the Simulation Data Inspector, you can overwrite a previously recorded run with a new run using the **Overwrite Run** button. Overwriting a run eliminates a large accumulation of runs when you are establishing a baseline run for comparing simulation runs. When you overwrite a run, signal selection and color are retained.

Signal Management

Referenced models sample times

You can use one or more variable sample times in a referenced model. You can include blocks with variable sample times, such as the Pulse Generator block, in a referenced model in Normal or Accelerator mode. The Sample Time Legend reflects this new capability with the following:

- Text label `Variable` in the **Description** column.
- Display of hierarchical structure of the value in the reference model hierarchy in the **Value** column.

For more information, see [Designate Sample Times](#).

Opening a model that contains referenced models with variable sample times generates errors in releases prior to R2013a.

Version History

If you want to use an S-function block that contains one variable sample time in an R2013a referenced model, recompile the S-function code in the R2013a environment first. Otherwise, compiling this model reference hierarchy in the R2013a environment generates errors.

Triggered subsystem sample times

The Sample Time Legend now displays the source of triggered subsystem sample times. The block area of a model now has annotations that show the triggered sample time and the sample time index number. For more information, see [Designate Sample Times](#)

Simulation of variable-size scalar signals

Previously, a model that used a variable-size scalar signal (width equals 1) would cause an error during a model update. You can now simulate a model with a variable-size scalar signal.

Block Enhancements

CORDIC approximation method for atan2 function of Trigonometric Function block

The Trigonometric Function block now supports the CORDIC approximation method for computing the output of the `atan2` function. For more information, see [Trigonometric Function](#).

Product and Gain blocks support Basic Linear Algebra Subprogram (BLAS) library

The Product and Gain blocks now support the Basic Linear Algebra Subprogram (BLAS) library. The BLAS library is a library of external linear algebra routines optimized for fast computation of large matrix operations. Whenever possible, the blocks use BLAS library routines to increase simulation speed.

Performance Advisor check for Delay block circular buffer setting

To improve simulation, the Performance Advisor checks that each Delay block in the model uses the appropriate buffer type. By default, the block uses an array buffer (the **Use circular buffer for state** option is not selected). However, when the delay length is large, a circular buffer can improve execution speed by keeping the number of copy operations constant. For more information, see [Check Delay block circular buffer setting](#).

MATLAB Function Blocks

Masking of MATLAB Function blocks to customize appearance, parameters, and documentation

In R2013a, you can mask a MATLAB Function block directly. In previous releases, you had to place the MATLAB Function Block in a subsystem, and then mask that subsystem.

Version History

In R2013a, MATLAB scripts or functions that rely on the `MaskType` property of MATLAB Function blocks need to be updated. For example, `get_param(handle_to_block, 'MaskType')` or `get_param(handle_to_block, 'MaskDescription')` now returns an empty value. Using `find_system(block_diagram_root, 'SFBlockType', 'MATLAB Function')` returns all MATLAB Function blocks. Using `get_param(handle_to_block, 'SFBlockType')` returns MATLAB Function. Do not create masks with Mask Type Stateflow, because the behavior is unpredictable.

File I/O function support

The following file I/O functions are now supported for code generation:

- `fclose`
- `fopen`
- `fprintf`

To view implementation details, see [Functions Supported for Code Generation — Alphabetical List](#).

Support for nonpersistent handle objects

You can now generate code for local variables that contain references to handle objects or System objects. In previous releases, generating code for these objects was limited to objects assigned to persistent variables.

Include custom C header files from MATLAB code

The `coder.cinclude` function allows you to specify in your MATLAB code which custom C header files to include in the generated C code. Each header file that you specify using `coder.cinclude` is included in every C/C++ file generated from your MATLAB code. You can specify whether the `#include` statement uses double quotes for application header files or angle brackets for system header files in the generated code.

For example, the following code for function `foo` specifies to include the application header file `mystruct.h` in the generated code using double quotes.

```
function y = foo(x1, x2)
%#codegen
coder.cinclude('mystruct.h');
...
```

For more information, see `coder.cinclude`.

Load from MAT-files

MATLAB Coder now supports a subset of the `load` function for loading run-time values from a MAT-file while running a MEX function. It also provides a new function, `coder.load`, for loading compile-time constants when generating MEX or standalone code. This support facilitates code generation from MATLAB code that uses `load` to load constants into a function. You no longer have to manually type in constants that were stored in a MAT-file.

To view implementation details for the `load` function, see [Functions Supported for Code Generation — Alphabetical List](#).

For more information, see `coder.load`.

`coder.opaque` function enhancements

When you use `coder.opaque` to declare a variable in the generated C code, you can now also specify the header file that defines the type of the variable. Specifying the location of the header file helps to avoid compilation errors because the code generation software can find the type definition more easily.

You can now compare `coder.opaque` variables of the same type. This capability helps you verify, for example, whether an `fopen` command succeeded.

```
null = coder.opaque('FILE*', 'NULL', 'HeaderFile', 'stdio.h');
ftmp = null;
ftmp = coder.ceval('fopen', fname, permission);
if ftmp == null
    % Error - file open failed
end
```

For more information, see `coder.opaque`.

Complex trigonometric functions

You can now use complex `acosD`, `acotD`, `acscD`, `asecD`, `asinD`, `atanD`, `cosD`, `cscD`, `cotD`, `secD`, `sind`, and `tand` functions with the MATLAB Function block.

Support for integers in number theory functions

Code generation supports integer inputs for the following number theory functions:

- `cumprod`
- `cumsum`
- `factor`
- `factorial`
- `gcd`
- `isprime`
- `lcm`

- median
- mode
- nchoosek
- nextpow2
- primes
- prod

To view implementation details, see [Functions Supported for Code Generation — Alphabetical List](#).

Enhanced support for class property initial values

If you initialize a class property, you can now assign a different type to the property when you use the class. For example, class `foo` has a property `prop1` of type `double`.

```
classdef foo %#codegen
    properties
        prop1= 0;
    end
    methods
        ...
    end
end
```

Function `bar` assigns a different type to `prop1`.

```
function bar %#codegen
    f=foo;
    f.prop1=single(0);
    ...
end
```

In R2013a, code generation ignores the initial property definition and uses the reassigned type. In previous releases, code generation did not support this reassignment and failed.

Version History

In previous releases, if the reassigned property had the same type as the initial value but a different size, the property became variable-size in the generated code. In R2013a, code generation uses the size of the reassigned property, and the size is fixed. If you have existing MATLAB code that relies on the property being variable-size, you cannot generate code for this code in R2013a. To fix this issue, do not initialize the property in the property definition block.

For example, you can no longer generate code for the following function `bar`.

Class `foo` has a property `prop1` which is a scalar `double`.

```
classdef foo %#codegen
    properties
        prop1= 0;
    end
    methods
        ...
    end
end
```

Function bar changes the size of prop1.

```
function bar %#codegen
    f=foo;
    f.prop1=[1 2 3];
    % Use f
    disp(f.prop1);
    f.prop1=[1 2 3 4 5 6 ];
```

Default use of Basic Linear Algebra Subprograms (BLAS) Libraries

Code generated for MATLAB Function blocks now uses BLAS libraries whenever they are available. There is no longer an option to turn off the use of these libraries.

Version History

If existing configuration settings disable BLAS, code generation now ignores these settings.

New toolbox functions supported for code generation

To view implementation details, see [Functions Supported for Code Generation — Alphabetical List](#).

Bitwise Operation Functions

- flintmax

Computer Vision System Toolbox Classes and Functions

- binaryFeatures
- insertMarker
- insertShape

Data File and Management Functions

- computer
- fclose
- fopen
- fprintf
- load

Image Processing Toolbox Functions

- conndef
- imcomplement
- imfill
- imhmax
- imhmin
- imreconstruct
- imregionalmax

- `imregionalmin`
- `iptcheckconn`
- `padarray`

Interpolation and Computational Geometry

- `interp2`

MATLAB Desktop Environment

- `ismac`
- `ispc`
- `isunix`

String Functions

- `strfind`
- `strep`

Function being removed

The `emlmex` function has been removed.

Version History

The `emlmex` function generates an error in R2013a.

Modeling Guidelines

Modeling Guidelines for High-Integrity Systems

The following are new modeling guidelines to develop models and generate code for high-integrity systems:

- himl_0001: Usage of standardized function headers
- himl_0002: Strong data typing (MATLAB Function block boundary)
- himl_0003: Limitation of MATLAB Function complexity

MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow

The MathWorks Automotive Advisory Board (MAAB) working group created Version 3.0 of the MAAB Guidelines Using MATLAB, Simulink, and Stateflow. See MAAB Control Algorithm Modeling for the MathWorks presentation of the guidelines.

Model Advisor

Model Advisor checks reorganized in a future release

In a future release, the Model Advisor checks will be reorganized.

Version History

MathWorks will review the products and licenses required to run the Model Advisor checks.

Model Advisor navigation between Upgrade Advisor, Performance Advisor, and Code Generation Advisor

In the Model Advisor window, you can select:

- **Code Generation Advisor** to help configure your model to meet code generation objectives.
- **Upgrade Advisor** to help upgrade models.
- **Performance Advisor** to help improve the simulation performance of your model.

Report

Single file HTML

Images in the Model Advisor html report are now embedded within the file, making it easier to export and maintain the report. Previously, the icon images were stored in separate files.

Format

The Model Advisor report format now uses indenting and *italics* to make it easier to assess the results of an analysis.

Preferences dialog box

From the Model Editor, select **Analysis > Model Advisor > Preferences** to open the Model Advisor Preferences dialog box. Alternately, in the Model Advisor window, select **Settings > Preferences**.

- From **Default Mode**, select **Model Advisor** or **Model Advisor Dashboard** to specify the interface that you want to use.
- Select **Show Accordion** for entry points to other Advisors from the Model Advisor window.

To display additional information about the checks in the Model Advisor window, select:

- **Show By Product Folder** for checks available for each product.
- **Show By Task Folder** for checks related to specific tasks.
- **Show Source tab** for check Title, TitleID, and location of the MATLAB source code for the check.
- **Show Exclusion tab** for checks that are excluded from the Model Advisor analysis.

By Product folder not displayed

By default, in the Model Advisor window, the **By Product** folder is not displayed. To display checks in the **By Product** folder, in the Model Advisor window, select **Settings > Preferences**. In the Model Advisor Preferences dialog box, select **Show By Product Folder**.

R2012b

Version: 8.0

New Features

Bug Fixes

Version History

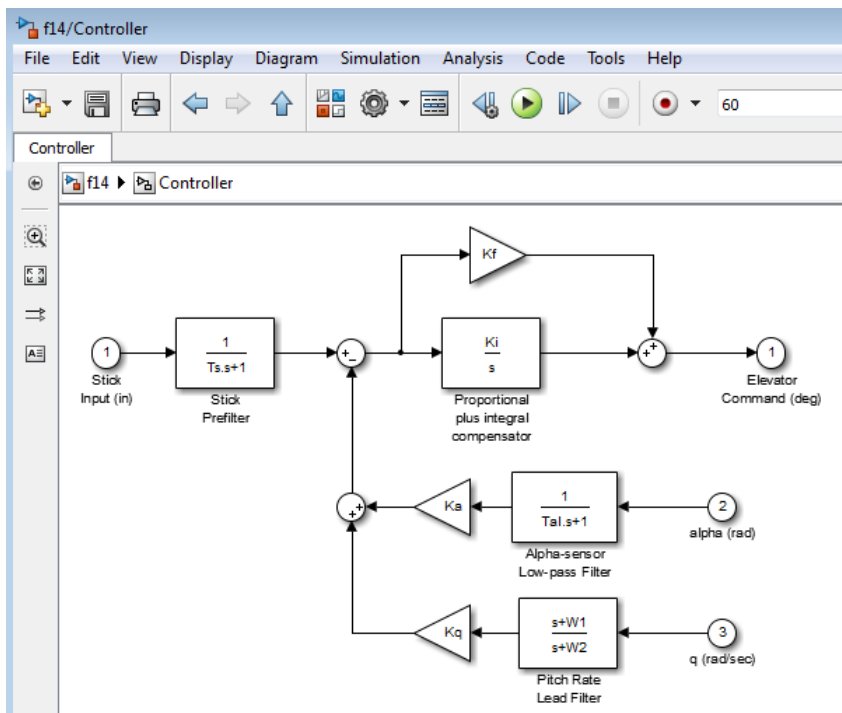
New Simulink Editor

Tabbed windows and automatic window reuse to minimize window clutter

By default, the Simulink Editor uses:

- One window to display a model and its subsystems
- The same tab for each system that you open in a model

For example, if you open the f14 model and then open the Controller subsystem, the Simulink Editor reuses the window and tab.



Window reuse and tabs:

- Conserve desktop space
- Keep the display of systems in a model together
- Provide easy navigation between systems in a model

To override the default window reuse behavior for a specific subsystem, right-click the subsystem and select either **Open in New Tab** or **Open in New Window**.

For more information, see Window Management.

Version History

The window display behavior for models created before R2012b remains the same in R2012b as it was in earlier releases. For example, if opening a model in an earlier release (such as R2011a)

opened three Simulink Editor windows, then when you open that same model in R2012b, three Simulink Editor windows also open.

However, if you open an earlier model that has a *callback* that opens a subsystem, by default the subsystem opens in the same Simulink Editor window that is used for the model. If you want the callback to open a separate window for the subsystem, include an `open_system` call that uses the `new window` argument.

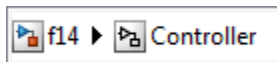
Smart signal routing that determines the simplest signal line path without overlapping blocks

When you draw lines to connect blocks, the Simulink Editor automatically routes the line to avoid other blocks and to minimize diagram clutter. You can manually modify the routing.

For more information, see [Connect Blocks](#).

Explorer bar to help with navigating through a model

The Explorer bar in the Simulink Editor provides a breadcrumb that shows the nested path for the currently open system.



Select a system in the breadcrumb to open that system in the model window. If you click in the Explorer bar whitespace, you can edit the hierarchy. Also, the down arrow at the right side of the Explorer bar provides a model display history.

Simulation stepper to simulate and rewind a model one step at a time

With the new simulation stepper, you can:

- Step forward and back in time during a simulation
- Set time breakpoints
- Set conditional breakpoints on a scalar signal

For more information, see [Simulation Stepping](#).

Ability to comment out blocks

Comment out blocks in your model if you want to exclude them during simulation. To exclude a block, right-click the selected block and select **Comment out**.

Commenting out blocks can be useful for several tasks, including to:

- Incrementally test parts of a model under development
- Debug a model without having to delete and restore blocks between simulation runs
- Test and verify the effects of certain model blocks on simulation results
- Improve simulation performance

Subsystem badges to identify and look under masked subsystems

Identify masked subsystems by the badge that appears in the lower-left corner of the mask (▼). Click this badge to open the mask.

Note The badge does not appear for masks of library links if the library is locked or its `LockLinksToLibrary` property is set to `true`.

Reorganized menu to fit common Model-Based Design workflow

The new Simulink Editor provides a top-level menu structure that reflects the steps of the model-based design process that you perform in the Simulink environment. The process is iterative and involves using different menus in different orders. The following table indicates the main menus associated with each Model-Based Design step.

Model-Based Design Step	Corresponding Menus
Build the Simulink block diagram.	<ul style="list-style-type: none"> • File • Edit • View • Diagram
Run the simulation.	Simulation
Validate the simulation results.	<ul style="list-style-type: none"> • View • Display • Analysis

The **Help** and **Tools** menus provide information and tools that apply throughout the steps of the process.

The **Code** menu provides options relating to code generation.

For more information, see Simulink Editor.

Version History

The names and menu paths for some menu items have changed from what they were in the R2012a Simulink Editor. For a summary of the changes to menus, keyboard and mouse shortcuts, badges, and Simulink preferences, see “Simulink Editor Changes” on page 21-6.

Palette for commonly used actions

You can use a palette of icons (to the left of the canvas area) to perform common actions, such as adding annotations or marquee zooming.

Panning and zooming

Pan through a model by pressing the mouse scroll wheel and dragging the mouse or by pressing the space bar while dragging the mouse. Zoom using the mouse scroll wheel. To change the default action of the mouse scroll wheel, use Simulink preferences.

For more information, see [Zoom and Pan Block Diagrams](#).

Display of overlapping blocks

Control which block appears on top, when there are overlapping blocks. From a block context menu, select **Arrange** and then choose either **Bring To Front** or **Send to Back**. (In the R2012a Simulink Editor, Simulink automatically set the display order, based on alphabetical order of the block name.)

Unification of Simulink and Stateflow Editors

The Simulink and Stateflow Editors now share most menu items. Additional aspects of the editor unification include:

- **Unified canvas:** Edit models and Stateflow charts in the same window.
- **Unified Model Browser tree:** Model Browser tree shows complete hierarchy, including Stateflow.

Also, you can now access the same editor functionality on Windows, UNIX®, and Mac platforms.

Simulink Editor preferences

The Simulink Preferences dialog box now includes preferences to control the look and behavior of the Simulink Editor. For example, you specify how the scroll wheel behaves, the look of the diagram (the diagram theme), and the toolbar configuration at a specific level (for example, whether or not to display the **Simulation** toolbar).

To access the Simulink Editor preferences from the Simulink Editor, select **File > Simulink Preferences > Editor Defaults**. To access those preferences from the Library Browser, select **File > Preferences > Editor Defaults**.

Version History

The following R2012a preferences do not appear in R2012b:

- **Window reuse**
- **Display Defaults for New Models > Browser visible**

In the new Simulink Editor, you can control these editor behaviors directly from within the editor. For details, see “Simulink Preferences Changes” on page 21-15.

Toolbar and status bar control

You can control the hiding or displaying of toolbars and the status bar by using Simulink Editor Default preferences. These preferences persist across editor sessions.

Version History

The model parameters `StatusBar` and `ToolBar` have no effect in the Simulink Editor. These parameters will be removed in a future release.

Visual editing based on model objects

The Simulink Editor provides tools such as smart guides and automated line routing that work based on the relative locations of model objects. This approach replaces pixel-oriented visual editing.

Version History

The model parameters `GridSpacing` and `ShowGrid` have no effect in the Simulink Editor. These parameters will be removed in a future release.

Improved callback error handling

When an interactive operation triggers a callback that causes an error in MATLAB, Simulink:

- Undoes the operation
- Issues an error message

Version History

Before R2012b, for an interactive operation that triggered a callback error in MATLAB, Simulink reported a warning (not an error), even though Simulink stopped the callback operation at the point of failure.

Simulink Editor Changes

- “Mapping from R2012a Simulink Editor to the New Simulink Editor” on page 21-7
- “File Menu” on page 21-7
- “Edit Menu” on page 21-7
- “View Menu” on page 21-8
- “Simulation Menu” on page 21-8
- “Format Menu” on page 21-9
- “Tools Menu” on page 21-10
- “Help Menu” on page 21-11
- “Simulink Editor Context Menu Changes” on page 21-11
- “Simulink Editor Mouse and Keyboard Shortcut Changes” on page 21-13
- “Simulink Editor Badges Changes” on page 21-15
- “Simulink Preferences Changes” on page 21-15
- “Simulink and Stateflow Editor Customization Changes” on page 21-15

Mapping from R2012a Simulink Editor to the New Simulink Editor

The following tables list the new Simulink Editor menu bar items that are different from the R2012a Simulink Editor.

The tables do *not* include menu bar items that:

- Have not changed
- Appear only in the new Simulink Editor

The arrows (>) indicate nested menu paths.

File Menu

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Preferences	File > Simulink Preferences and File > Stateflow Preferences.
Export to Web	File > Export Model to Web.
Print Details	File > Print > Print Details.
Print Setup	File > Print > Printer Setup.
Enable Tiled Printing	File > Print > Enable Tiled Printing.

Edit Menu

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Copy Model to Clipboard	Edit > Copy Current View to Clipboard (copies the open Model Editor window contents to the clipboard).
Create Subsystem	Diagram > Subsystem & Model Reference > Create Subsystem from Selection.
Create Mask	If the block is not already masked, the dynamic menu path is Diagram > Mask > Create Mask . If the block is already masked, the dynamic menu path is Diagram > Mask > Edit Mask .
Look Under Mask	Diagram > Mask > Look Under Mask.
Link Options	Diagram > Library Link.
Links and Model Blocks > Refresh	Diagram > Subsystem & Model Reference > Refresh Selected Model Block or, for library links, Diagram > Refresh Blocks.
Links and Model Blocks > Model Block Normal Mode Visibility	Diagram > Subsystem & Model Reference > Model Block Normal Mode Visibility.
Update Diagram	Simulation > Update Diagram.

View Menu

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Back	Access each option using View > Navigate .
Forward	Go To Parent changed to Up to Parent .
Go To Parent	
Model Browser Options	View > Model Browser .
Block Data Tip Options	Display > Blocks > Tool Tip Options . The submenu item Block Description changed to Description .
Requirements	View > Requirements at This Level .
Signal Hierarchy	Diagram > Signals & Ports > Signal Hierarchy .
Sample Time Legend	Display > Sample Time > Sample Time Legend .
Zoom In	Access each option using View > Zoom .
Zoom Out	Fit System to View changed to Fit to View .
Fit System To View	Normal (100%) changed to Normal View (100%) .
Normal (100%)	
Show Page Boundaries	File > Print > Show Page Boundaries .
Port Values submenus, such as Show When Hovering	Display > Data Display in Simulation .
Highlight/Remove Highlighting	Display > Highlight Signal to Source or Display > Highlight Signal to Destination or Display > Remove Highlighting .

Simulation Menu

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Configuration Parameters	Simulation > Model Configuration Parameters .
Normal	Access each option using Simulation > Mode .
Accelerator	
Rapid Accelerator	
Software-in-the-Loop (SIL)	
Processor-in-the-Loop	
External	
Start	Simulation > Run .

Format Menu

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Font	Diagram > Format > Font Style. On Mac platforms, supported fonts must be in both the X11 and Mac font manager.
Text Alignment	Access each option using Diagram > Format.
Enable TeX Commands	
Show Name	Diagram > Format > Show Block Name.
Show Drop Shadow	Diagram > Format > Block Shadow.
Show Port Labels	Diagram > Format > Port Labels.
Foreground Color	Access each option using Diagram > Format.
Background Color	
Screen Color	Diagram > Format > Canvas Color.
Show Smart Guides	View > Smart Guides.
Align Blocks	Access using block alignment, distribution, and resizing options using Diagram > Arrange.
Distribute Blocks	
Resize Blocks	
Flip Name	Diagram > Rotate & Flip > Flip Block Name.
Flip Block	Diagram > Rotate & Flip > Flip Block.
Rotate Block	Access block rotation options using Diagram > Rotate & Flip.
Port/Signal Displays	Access most port and signal display options using Display > Signals & Ports. Signal Resolution Indicators changed to Signal to Object Resolution Indicator.
Block Display > Sorted Order	Display > Blocks > Sorted Execution Order.
Block Display > Model Block Version	Display > Blocks > Block Version for Referenced Models.
Block Display > Model Block I/O Mismatch	Display > Blocks > Block I/O Mismatch for Referenced Model.
Block Display > Execution Context Indicator	Display > Blocks > Sorted Execution Order.
Library Link Display	Access library link display options using Display > Library Links.
Sample Time Display	Access sample time options using Display > Sample Time. None changed to Off.

Tools Menu

The tools that appear in the **Tools** menu and other menus reflect the products for which you have a license.

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Compare Simulink XML Files	Access these tools using Analysis .
Control Design	The menu text for these items has changed:
Coverage Settings	<ul style="list-style-type: none"> • Coverage Settings changed to Coverage.
Design Verifier	<ul style="list-style-type: none"> • Fixed-Point changed to Fixed Point Tool (i.e., there is no hyphen now)
Fixed-Point Tool	<ul style="list-style-type: none"> • Model Dependencies > View/Edit Manifest Contents changed to Model Dependencies > Edit Manifest Contents.
Model Advisor	<ul style="list-style-type: none"> • Profiler changed to Show Profiler Report.
Model Dependencies	
Parameter Estimation	
Profiler	
Requirements	
Simulink Debugger	Simulation > Debug > Debug Model .
Bus Editor	Access these tools using Edit .
Lookup Table Editor	
Inspect Logged Signals	Simulation > Output > Simulation Data Inspector .
Signal & Scope Manager	Diagram > Signals & Ports > Signal & Scope Manager .
Code Generation	Access these options using Code .
External Mode Control Panel	Code Generation changed to C/C++ Code .
HDL Code Generation	HDL Code Generation changed to HDL Code .
Simulink Code Inspector	
Verification Wizards	
Define Data Classes	Access these options using Code > Data Objects .
Data Object Wizard	Define Data Classes changed to Design Data Classes .
Response Optimization	Analysis > Response Optimization .

Help Menu

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Using Simulink	Help > Simulink > Simulink Help.
Blocks	Help > Simulink > Blocks & Blocksets Reference.
Blocksets	
Block Support Table	Help > Simulink > Block Data Types & Code Generation Support.
Shortcuts	Help > Keyboard Shortcuts.
S-Functions	Help > Simulink > S-Functions.
Demos	Help > Simulink > Examples and Help > Stateflow > Examples.
About Simulink	Help > About > Simulink.

Simulink Editor Context Menu Changes

From the Canvas

R2012a Simulink Editor Context Menu	New Simulink Editor Equivalent
Back	Not available from context menu.
Forward	From the menu bar, select the appropriate menu item from View > Navigate .
Go to Parent	
Configuration Parameters	Model Configuration Parameters .
Format > Wide Nonscalar Lines	Access these options using Display > Signals & Ports .
Format > Signal Dimensions	
Format > Port Data Types	
Link Options	Not available from the canvas context menu. Access either from a block context menu or from the menu bar, using Diagram > Library Link .
Requirements	Requirements at This Level .
Screen Color	Canvas Color .
Signal & Scope Manager	Not available from context menu. From the menu bar, use Diagram > Signals & Ports > Signal & Scope Manager .
Fixed-Point Tool	Fixed Point Tool .

From a Block

R2012a Simulink Editor Context Menu	New Simulink Editor Equivalent
Block Properties	Properties .

R2012a Simulink Editor Context Menu	New Simulink Editor Equivalent
Foreground Color	Access these options using Format .
Background Color	
Convert to Model Block (for Subsystem block context menu)	Subsystem & Model Reference > Convert Subsystem to > Referenced Model.
Format > Flip Block	Access these options using Rotate & Flip .
Format > Flip Name	
Format > Rotate Block	
Format > Font	Format > Font Style.
Format > Show Drop Shadow	Format > Block Shadow.
Format > Show Name	Format > Show Block Name.
Format > Show Port Labels	Format > Port Labels.
HDL Code Generation	HDL Code.
Link Options	Not available from context menu. From the menu bar, use Diagram > Library Link .
Mask Subsystem	If the block is <i>not</i> already masked, the dynamic menu path is Mask > Create Mask . If the block is already masked, the dynamic menu path is Mask > Edit Mask .
Look Under Mask	Mask > Look Under Mask.
Port Signal Properties	Signals & Ports.
Signal & Scope Manager	Not available from context menu. From the menu bar, use Diagram > Signals & Ports > Signal & Scope Manager .
Create Subsystem	Subsystem & Model Reference > Create Subsystem from Selection.
Code Generation > Generate Protected Model (from a Model block)	From a Model block: C/C++ Code > Generate Protected Model Note that you can also access this option from the main menu: Code > C/C++ Code > Generate Protected Model .
Refresh (from a Model block)	Subsystem & Model Reference > Refresh Selected Model Block

From a Signal

R2012a Simulink Editor Context Menu	New Simulink Editor Equivalent
Connect to Existing Viewer	Connect to Viewer.

R2012a Simulink Editor Context Menu	New Simulink Editor Equivalent
Disconnect & Delete Viewer	Use a combination of Disconnect Viewer and Delete Viewer .
Fixed-Point Tool	Not available from context menu. From the menu bar, use Analysis > Fixed Point .
Highlight to Source	Highlight Signal to Source .
Highlight to Destination	Highlight Signal to Destination .
Linearization Points	Linear Analysis Points .

Simulink Editor Mouse and Keyboard Shortcut Changes

Mouse Scroll Wheel

By default, in the new Simulink Editor, rolling the mouse scroll wheel up zooms in on a model, and rolling the wheel down zooms out.

To scroll left and right using the mouse scroll wheel, press **Shift** while rolling the wheel. To scroll up and down, press **Ctrl** while rolling the wheel.

You can change the default scroll wheel behavior. In the Simulink Preferences dialog box, clear **Editor Defaults > Scroll wheel controls zooming**.

Model Viewing Shortcuts

Task	R2012a Simulink Editor Shortcut	New Simulink Editor Equivalent
Zoom in	r	Use the mouse scroll wheel or Ctrl+ + (the plus sign)
Zoom out	v	Use the mouse scroll wheel or Ctrl +- (the minus sign).
Zoom to normal view	1	Alt+1
Pan left	d or Ctrl+Left Arrow	If scroll bars are visible, then with nothing selected, use Shift+Left Arrow or for finer panning, just the Left Arrow .
Pan right	g or Ctrl+Right Arrow	If scroll bars are visible, then with nothing selected, use Shift+Right Arrow or for finer panning, just the Right Arrow .
Pan up	e or Ctrl+Up Arrow	If scroll bars are visible, then with nothing selected, use Shift+Up Arrow or for finer panning, just the UpArrow .
Pan down	c or Ctrl+Down Arrow	If scroll bars are visible, then with nothing selected, use Shift+Down Arrow or for finer panning, just the Down Arrow .

Task	R2012a Simulink Editor Shortcut	New Simulink Editor Equivalent
Fit selection to screen	f	You can also use marquee zoom to fit a selection to the screen.

Block Editing Shortcuts

Task	R2012a Simulink Editor Shortcut	New Simulink Editor Equivalent
Move a block from one model to another model	Shift , press the left mouse button, and drag block to different model. The block is disconnected moved from the source model to the other model.	In the new Simulink Editor, Shift , press the left mouse button, and drag block to different model copies the block, but does not remove it from the source model.

Line Editing Shortcuts

Task	R2012a Simulink Editor Shortcut	New Simulink Editor Equivalent
Create line segments	Move the cursor to the end of line and drag the line.	New Simulink Editor performs autorouting. You can manually control the line segment by clicking the arrow guides. The arrow guides appear when you edit a previously disconnected signal from its endpoint. During new line creation, let go of the mouse button and then move it off the newly created endpoint.
Create diagonal line segments	Click anywhere on a line, Shift , and move the cursor.	For an existing line, click a bend (corner) or solder (joint) point, Shift , and move the cursor. For new lines, use Shift and the arrow guides.

Signal Label Editing Shortcuts













Task	R2012a Simulink Editor Shortcut	New Simulink Editor Equivalent
Delete signal label	Shift and click label. Then press Delete .	Right-click the label and from the context menu, select Delete Label .

Annotation Editing Shortcuts

Task	R2012a Simulink Editor Shortcut	New Simulink Editor Equivalent
Delete annotation	Shift and select annotation. Then press Delete .	Right-click the annotation and from the context menu, select Delete .

Simulink Editor Badges Changes

Badges are the icons that appear in the Simulink Editor to provide information about how you have configured a model.

Badge	R2012a Simulink Editor Badge	New Simulink Editor Badge
Signal viewer		
Library link active		
Library link inactive		
Library link locked		
Library link parameterized		
Model protected		

Simulink Preferences Changes

Two R2012a Simulink preferences no longer appear in R2012b. In the new Simulink Editor, you can control these editor behaviors directly from within the editor.

R2012a Simulink Preference	New Simulink Editor Equivalent
Window reuse	The Simulink Editor opens subsystems in the same window that it uses for the model that contains the subsystem. There is no global setting to change that behavior. However, you can control the window and tab behavior when opening a specific subsystem. For details, see Navigate Model Hierarchy .
Browser visible	To display or hide the Model Browser, select or clear the View > Model Browser > Show Model Browser option.

Also, the MATLAB Preferences dialog box **Figure Copy Template > Copy Options** preferences no longer apply to copying a model from the clipboard to a third-party application.

Simulink and Stateflow Editor Customization Changes

Customizing the new Simulink and Stateflow editors is as it was in R2012a, with the following exceptions:

- The addition of custom menu functions to the ends of top-level menus depends on the active editor:
 - Menus bound to `Simulink:FileMenu` only appear when the Simulink Editor is active.
 - Menus bound to `Stateflow:FileMenu` only appear when the Stateflow Editor is active.
 - To have a menu to appear in both of the editors, call `addCustomMenuFcn` twice, once for each tag. Check that the code works in both editors.

- If a filter is applied to the `Simulink` tag, then a menu item that existed in Simulink and Stateflow editors in R2012a is filtered, regardless of the active editor type. However, if the filter is applied to the `Stateflow` tag, then the menu item is only filtered in the Stateflow Editor.
- If a menu item tag has changed in R2012b, you do not need to change the tag to the R2012b tag.

For more information about customizing menus, see [Add Items to Model Editor Menus and Disable and Hide Model Editor Menu Items](#).

Connection to Educational Hardware

Support for Arduino and PandaBoard hardware

- “Support for Arduino Mega 2560 and Arduino Uno hardware” on page 21-17
- “Support for PandaBoard hardware” on page 21-17

Support for Arduino Mega 2560 and Arduino Uno hardware

Run Simulink models on Arduino Mega 2560 and Arduino Uno hardware. For more information, see the Arduino topic.

To use this capability, first run Target Installer and install support for Arduino hardware. To run Target Installer, enter `targetinstaller` in the MATLAB Command Window.

After installing support, you can use the Simulink “**Target for Use with Arduino Hardware**” block library. To open this block library, enter `arduinolib` in the MATLAB Command Window.

This block library contains the following blocks:

- Arduino Analog Input
- Arduino PWM
- Arduino Digital Input
- Arduino Digital Output
- Arduino Serial Receive
- Arduino Serial Transmit
- Arduino Standard Servo Read
- Arduino Standard Servo Write
- Arduino Continuous Servo Write

After you install support, Target Installer displays the following examples:

- Getting Started with Arduino Hardware
- Communicating with Arduino Hardware (Arduino Mega 2560 only)
- Servo Control
- Drive with PID Control

Support for PandaBoard hardware

Run Simulink models on PandaBoard hardware. For more information, see the PandaBoard topic.

To use this capability, first run Target Installer and install support for PandaBoard hardware. To run Target Installer, enter `targetinstaller` in the MATLAB Command Window.

After installing support, you can use the Simulink “**Target for Use with PandaBoard Hardware**” block library. To open this block library, enter `pandaboardlib` in the MATLAB Command Window.

This block library contains the following blocks:

- PandaBoard UDP Receive
- PandaBoard UDP Send
- PandaBoard ALSA Audio Capture
- PandaBoard ALSA Audio Playback
- PandaBoard V4L2 Video Capture
- PandaBoard SDL Video Display

Bluetooth download to LEGO MINDSTORMS NXT hardware

You can use a Bluetooth connection instead of a USB cable to download a Simulink model from your host computer to the LEGO MINDSTORMS NXT hardware. Previously, a USB cable was the only connection available for downloading models to the NXT hardware. For more information, see:

- Run Model on NXT Brick
- Set Up A Bluetooth Connection

Performance

Simulation Performance Advisor that analyzes your model and provides advice on how to increase simulation performance

Use the Performance Advisor to check models for conditions and configuration settings that can result in inefficient simulation of the system that the model represents. The Performance Advisor produces a report that lists the suboptimal conditions or settings that it finds, suggesting better model configuration settings where appropriate. It also provides mechanisms for automatically fixing warnings and failures or allowing you to fix them manually. For more information, see Consult the Performance Advisor.

Project and File Management

Simulink default file format SLX that uses the OPC standard

In R2012b, Simulink has a new default file format for models, SLX, with the file extension `.slx`. In R2012a, SLX was available as an option.

The SLX file format contains the same information as an MDL file and is a compressed package that conforms to the Open Packaging Conventions (OPC) interoperability standard. SLX stores model information using Unicode UTF-8 in XML and other international formats.

Saving Simulink models in the SLX format:

- Typically reduces file size. The file size reduction between MDL and SLX varies depending on the model.
- Solves some problems in previous releases with loading and saving MDL files containing Korean and Chinese characters.
- Supports new features in future releases not supported with MDL format.

You can still choose to save model files as MDL, and the MDL format will remain available for the foreseeable future.

For more information, see [Saving Models in the SLX File Format](#).

Version History

If you upgrade an MDL file to SLX file format, the file contains the same information as the MDL file, and you always have a backup file. All functionality and APIs that currently exist for working with models, such as the `get_param` and `set_param` commands, are also available when using the SLX file format.

The MDL file format will continue to be supported, but, after R2012b, new features might be available only if you use the SLX file format.

The new file extension `.slx` might cause compatibility issues if your scripts contain hard-coded references to file names with extension `.mdl`. To check for problems, verify that your code works with both the MDL and SLX formats. If you find any places in your scripts that need to be updated, use functions like `which` and `what` instead of strings with `.mdl`.

Caution If you use third-party source control tools, be sure to register the model file extension `.slx` as a binary file format. If you do not, these third-party tools might corrupt SLX files when you submit them.

Operations with Possible Compatibility Considerations	What Happens	Action
Hard-coded references to file names with extension <code>.mdl</code> .	Scripts cannot find or process models saved with new file extension <code>.slx</code> .	Make your code work with both the <code>.mdl</code> and <code>.slx</code> extension. Use functions like <code>which</code> and <code>what</code> instead of strings with <code>.mdl</code> .
Third-party source control tools that assume a text format by default.	Binary format of SLX files can cause third-party tools to corrupt the files when you submit them.	Register <code>.slx</code> as a binary file format with third-party source control tools.

The format of content within MDL and SLX files is subject to change. Use documented APIs (such as `get_param`, `find_system` and `Simulink.MDLInfo`) to operate on model data.

Simulink Upgrade Advisor to help migrate files to the current release

Use the Upgrade Advisor for help with using the current release to upgrade and improve models.

The Upgrade Advisor identifies cases where you can benefit by changing your model to use new features and settings in Simulink. The Advisor provides advice for transitioning to new technologies and upgrading a model hierarchy.

The Upgrade Advisor also identifies cases when a model will not work because changes and improvements in Simulink require changes to a model.

The Upgrade Advisor offers options to perform recommended actions automatically or instructions for manual fixes.

See Consult the Upgrade Advisor.

Built-in SVN adapter for Simulink Projects that provides connectivity to SVN and support for server-based repositories

Simulink Projects now provide built-in Subversion source control integration, reducing setup and providing faster performance and improved support for connecting to servers. You can now connect to servers that require login.

Previously, you had to install an additional command-line SVN client to use Projects with SVN. Now you can use SVN for project source control with no additional installation steps.

See Subversion Integration with Projects.

Simulink Project Tool dependency graph that provides highlights by file type, dependency type, and label

After you run dependency analysis on your project, you can use the Graph view to examine dependencies for impact analysis. You can now highlight dependencies by file type, dependency type, and file labels. For example, you might want to highlight model files and see which have the label To Review. Previously, you could highlight only upstream and downstream dependencies of the selected file. Now you can also highlight circular dependencies.

You can now also perform file operations in the Graph view, such as **Open**, **Add to Project**, **Add Label**, and **Remove from Project**. File operations can be useful when using the graphical dependency view to identify required changes to your project, such as identifying files that need removing or files that share a common label.

The Dependencies results list and Graph view are now separate tree node views for efficient workflow, instead of tabs within the Dependency Analysis view. These views also now display a time stamp to identify when the analysis was performed.

See Analyze Project Dependencies.

Redesigned graphical tool for efficient Simulink Projects workflow

The Simulink Project Tool is redesigned for more efficient workflow and access to tools.

- Simulink Projects are integrated with the MATLAB Toolstrip when you dock the tool, with new options to create and open recent projects from MATLAB.
- The toolstrip contains components that were previously available in menus and toolbars.
- All project views have new tools for improved browsing, searching, and filtering.
- There are new tree nodes for accessing the batch processing, dependency results, and dependency graph views.
- The new source control pane provides access to configuration management tasks and more space in other views.
- New archive options include the capability to create a new project from a zip archive.
- New project integrity checks assist with MDL to SLX upgrades, check for `slprj` folders added to projects, and provide **Fix** buttons for tasks that can be automated.

See Simulink Projects.

Batch operation support for files in a Simulink Project

The Simulink Project Tool has a new Batch Job view to help you create and run functions on selected project files. Batch tools provide guidance for creating your own batch functions. An example batch job function identifies and saves any model files that contain unsaved changes.

See Run Batch Functions on Project Files.

Create and open recent Simulink Projects from MATLAB

The MATLAB Toolstrip has new options to create Simulink Projects and open recent projects direct from the MATLAB Desktop.

Simulink Projects are integrated with the MATLAB Toolstrip when you dock the tool.

See:

- Create a New Simulink Project
- Open Recent Projects
- Create a New Project from an Archived Project

- Retrieve a Working Copy of a Project from Source Control

.

Block Enhancements

Menu item to convert configurable and normal subsystems to variant subsystems

Previously, to convert configurable or normal subsystems to variant subsystems, you had to create a new variant subsystem in your model and manually modify it to match the subsystem you were converting.

That method was error-prone. Manually matching a variant subsystem and its ports to the converted subsystem was also a time-consuming process. Moreover, configurable subsystems will not be supported in a future release.

In this release, you can do this conversion by right-clicking a subsystem and selecting **Subsystems and Model Reference > Convert Subsystem To > Variant Subsystem**.

Simulink creates a new variant subsystem and an appropriate number of inports and outports that match the converted subsystem.

Masking improvements, including the ability to reuse masks, delete existing masks on blocks, and use the shortcut operator `||` in mask callback code

Use classes `Simulink.Mask` and `Simulink.MaskParameter` to control masks programmatically. With these classes, you can perform the following mask operations:

- Create, copy, and delete masks
- Add, edit, and delete mask parameters
- Get mask owner and set properties for masks and mask parameters

In addition, in this release, you can use the OR operator `||`, which was previously prohibited, in mask callback code.

Default output data type of Logic blocks changed to boolean

The following blocks now use a default value of `boolean` for **Output data type**. In previous releases, the blocks used `uint8` as the default output data type.

- Compare To Constant
- Compare To Zero
- Detect Change
- Detect Decrease
- Detect Fall Negative
- Detect Fall Nonpositive
- Detect Increase
- Detect Rise Nonnegative
- Detect Rise Positive

Signal Attributes tab of dialog box for Operator blocks renamed to Data Type

For the Logical Operator and Relational Operator blocks, the name of the **Signal Attributes** tab of the block dialog box has changed to **Data Type**.

Parameter name changes for Unit Delay block

For the Unit Delay block, the following parameters have been renamed:

Version History

Old Name	New Name
X0	InitialCondition
StateIdentifier	StateName
RTWStateStorageClass	CodeGenStateStorageClass
RTWStateStorageTypeQualifier	CodeGenStateStorageTypeQualifier

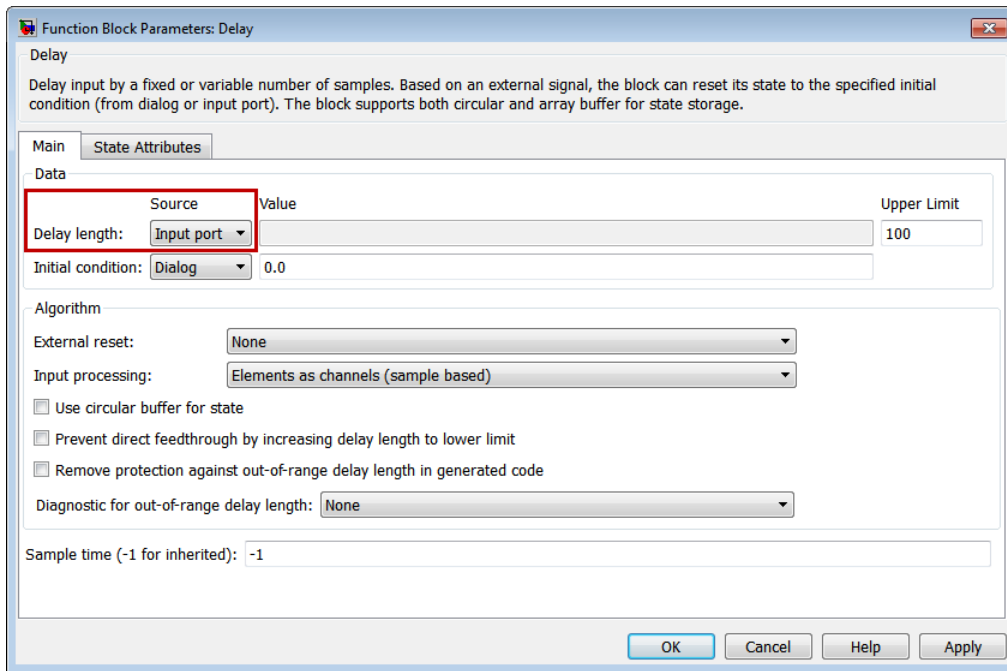
New variants of Delay block in Discrete library

The Discrete library now contains the following additional variants of the Delay block:

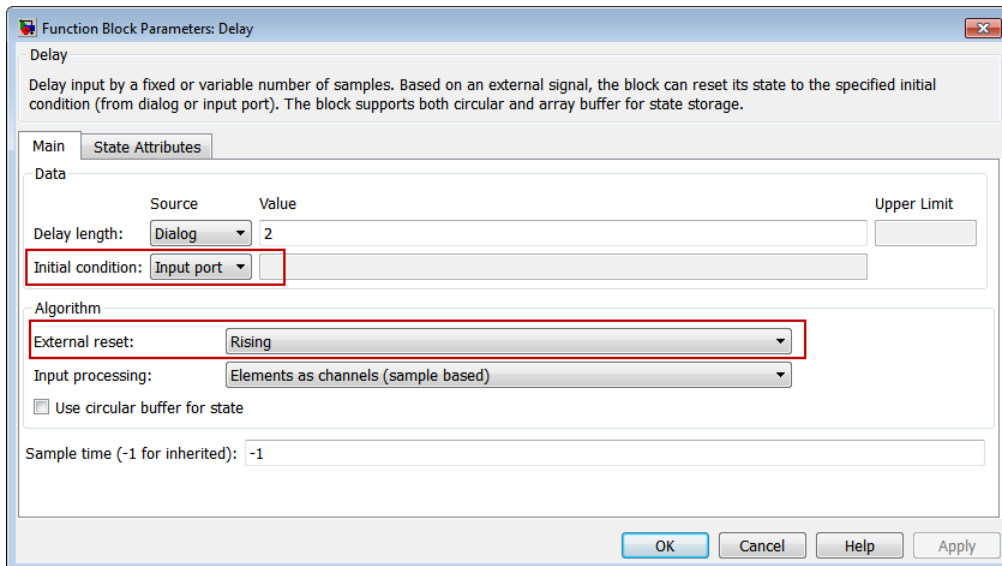
- Variable Integer Delay block
- Resettable Delay block

You can configure the Delay block to work in the same way as either of these variants.

With the source of the delay length set to `Input port`, the Delay block works as a Variable Integer Delay block.



To configure the Delay block to have a resettable delay, set the source of the initial condition to **Input port** and the external reset algorithm to **Rising**.



Some Probe block parameters no longer support boolean data type

The following parameters of the Probe block no longer support the boolean data type:

- Data type for width
- Data type for sample time
- Data type for signal dimensions

If the width, sample time, or dimensions of the input signal has a value greater than zero, the `boolean` data type implicitly represents the output as `1`, which is not a useful result. Setting any of the listed parameters to `Same as input` while the block's input signal data type is `boolean` results in a simulation error.

Internationalization of block dialog box titles and buttons and block tooltips

To enable translation in localized versions of the Simulink software, in this release, the following items relating to Simulink blocks were internationalized:

- Titles of block dialog boxes
- Text for block tooltips

As a result of these changes, in the Japanese version of Simulink, and in future localized versions of the product, these items will display in translated form.

Enabled and triggered subsystems

For triggered and enabled subsystems, the Simulink software now performs zero-crossing detection and zero-crossing state updates of the trigger port outside the enable check.

In previous releases, for triggered and enabled subsystems, the Simulink software performed zero-crossing detection and zero-crossing state updates of the trigger port inside the enable check. This behavior sometimes caused simulation and code generation data mismatches.

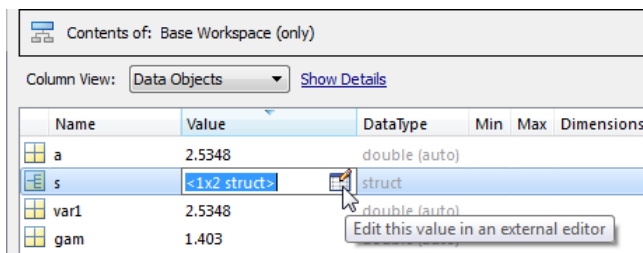
Data Management

Variable Editor access from within Model Explorer

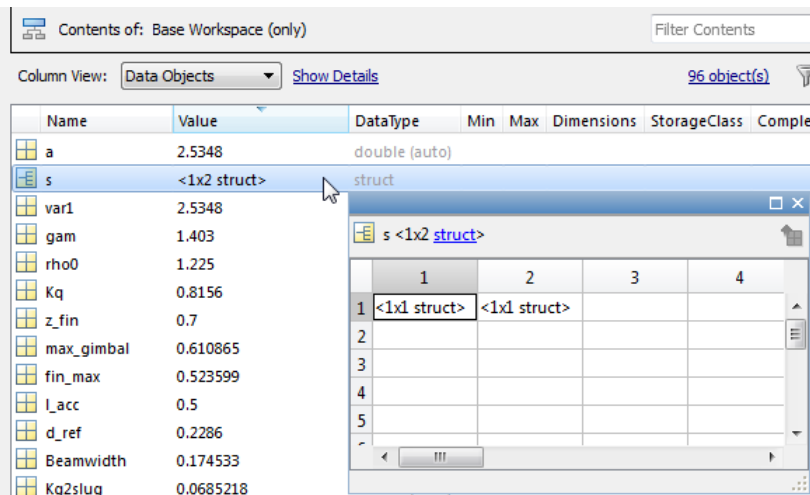
In the Model Explorer **Contents** pane, you can use the Variable Editor to edit variables from the MATLAB workspace or model workspace. The Variable Editor is available for editing large arrays and structures.

To open the Variable Editor for a variable that is an array or structure:

- 1 Click the Value cell for the variable.
- 2 Select the Variable Editor icon.



The Variable Editor opens:



You can resize and move the Variable Editor. The **Contents** pane reflects the edits that you make in the Variable Editor.

For details, see [Editing Workspace Variables](#).

Logged simulation data from Simulation Data Inspector accessible from Simulink toolbar

The record button for the Simulation Data Inspector tool is now accessible on the Simulink toolbar. Previously, the record button was a global setting for all models. The record button now applies per model. Click the record button to select it and then simulate the model to record and inspect logged

signal data. You can open the Simulation Data Inspector tool by clicking the down arrow and selecting **Open Simulation Data Inspector**. For more information, see [Record Simulation Data](#).

Version History

The record button no longer appears on the Simulation Data Inspector tool.

Specify `verifySignalAndModelPaths` action

You can specify the action that the `verifySignalAndModelPaths` method of the signal logging class `Simulink.SimulationData.ModelLoggingInfo` takes when it detects an invalid path.

Import and map data to root-level input ports

The Configuration Parameters dialog box **Data Import/Export > Input** parameter now has an **Edit Input** button. Use this button to start the Root Input Mapping tool. This tool lets you import data from a MAT-file and automatically map that data to root-level input ports. For more information, see [Import and Map Data to Root-Level Inports](#).

Dataset signal logging format for increased flexibility and ease of use

The default format for saving signal logging data is now `Dataset`.

With the `Dataset` format, you can do the following tasks, which you cannot do with the previous default format of `ModelDataLogs`:

- Work with logging data in MATLAB without a Simulink license
- Log multiple data values for a given time step, which can be important for Iterator subsystem and Stateflow signal logging
- Easily analyze logged signal data for models with deep hierarchies, bus signals, and signals with duplicate or invalid names
- Avoid the limitations of the `ModelDataLogs` format, which Bug Report 495436 describes.

To specify the signal logging format, use the **Configuration Parameters > Data Import/Export > Signal logging format** parameter. For more information, see [Specify the Signal Logging Data Format](#).

Version History

Before R2012b, the default signal logging format was `ModelDataLogs`. In R2012b, the default format is `Dataset`. The `ModelDataLogs` format will be removed in a future release.

In R2012b, Simulink displays a warning if you run a model that meets *both* of the following conditions:

- The **Configuration Parameters > Data Import/Export > Signal logging format** parameter is set to `ModelDataLogs`.
- The model has signal logging enabled for at least one signal or uses signal viewer scopes.

Use the Upgrade Advisor to upgrade a model to use `Dataset` format, using *one* of these approaches:

- In the Simulink Editor, select **Analysis > Model Advisor > Upgrade Advisor**
- From the MATLAB command line, use the `upgradeadvisor` function.

For more information about how to update models to use `Dataset`, including how to address issues that you might encounter after converting a model to use `Dataset` format, see [Migrate from ModelDataLogs to Dataset Format](#).

Data type field displays user-defined data types

Previously, the data type fields in dialog boxes for various data entities such as data objects and blocks displayed only built-in data types. For `mpt` signal and parameter objects, you could customize this list to include user-defined data types. For this customization, you had to modify the Simulink customization file `sl_customization.m` to add user-defined data types to the list.

In R2012b, the data type field displays both user-defined and built-in data types, provided these user-defined data types exist in the base workspace. This enhancement is not restricted to `mpt` data objects. All dialog boxes that contain the data type field will display user-defined data types.

Any modifications you make to `sl_customization.m` in order to display user-defined data types will still be supported.

Simulink.VariableUsage to get variable information

In R2012b, use `Simulink.VariableUsage` to determine which blocks use a variable defined in the model, mask, or base workspace.

Previously, you used `Simulink.WorkspaceVar` to get this information.

Version History

`Simulink.WorkspaceVar` will not be supported in a future release. If you use `Simulink.WorkspaceVar` in your code to programmatically get variable information, replace it with `Simulink.VariableUsage`.

Customizable line specification in Simulation Data Inspector

In the Simulation Data Inspector tool, the line specification includes customizable color selection and more marker specifiers for plotting data points. To view the line specification, in the Signal Browser table, click the **Line** column of a signal. For more information, see [Specify the Line Configuration](#).

Simulation Data Inspector report includes harness model information

A Simulation Data Inspector report of a recorded simulation of a Simulink Verification and Validation harness model now includes model information and a model diagram of the system under test and the test harness model. For more information on generating a report, see [Create Simulation Data Inspector Report](#).

Component-Based Modeling

Model configuration for targets with multicore processors

This capability has the following changes:

- Models configured for concurrent execution can now contain blocks that require implicit ODE solvers, such as physical modeling blocks. In previous releases, models that contained such blocks returned an error message during simulation and code generation.
- Code generation for models configured for concurrent execution, and which contain a large number of continuous states, now have improved performance and decreased memory usage.

New Simulink.GlobalDataTransfer class

To configure data transfers for models configured for concurrent execution, use the `Simulink.GlobalDataTransfer` class. This class contains the properties:

- `DefaultTransitionBetweenSyncTasks`
- `DefaultTransitionBetweenContTasks`
- `DefaultExtrapolationMethodBetweenContTasks`
- `AutoInsertRateTranBlk`

To access the properties of this class, use the `get_param` function to get the handle for this class, then use dot notation to access the properties, for example:

```
dt=get_param(gcs, 'DataTransfer');
dt.DefaultTransitionBetweenContTasks
```

```
ans =
```

```
Ensure deterministic transfer (minimum delay)
```

Reduced memory usage in models with many library links

Simulink now saves memory by closing partially loaded libraries on subsequent simulations. In previous releases, Simulink loaded linked blocks by partially loading their source libraries when you opened, updated, or simulated the model. These partially loaded libraries were never closed again, resulting in unnecessary memory use. Now Simulink closes partially loaded libraries once they are no longer needed, after loading the linked blocks. Reducing memory use can increase performance in large models with many library links.

Version History

If you have scripts that assume libraries are loaded and try to access the library, these scripts will now produce errors. You must update scripts to load libraries using `load_system` before running commands such as `set_param` on a library.

Configuration Reference dialog box to propagate and undo configuration settings to all referenced models

To share a configuration reference among referenced models in a model hierarchy, the Configuration Reference Propagation dialog box provides:

- A list of referenced models in the top model
- The ability to select only specific referenced models for propagation
- After propagation, a display of the status for the converted configuration for each referenced model
- The ability to undo the configuration reference and restore the previous configuration settings for a referenced model

For more information, see [Manage Configuration Reference Across Referenced Models and Share a Configuration Across Referenced Models](#).

Context-dependent function-call subsystem input handling improved

Executing a function-call subsystem that has context-dependent inputs can result in nondeterministic simulation results. Detecting dependent input at simulation time helps to avoid unexpected code generation results.

Before R2012b, if you wanted Simulink to flag such cases as errors, you needed to set the `HiliteFcnCallInpInsideContext` model parameter each time you load the model. You could not save the setting for that parameter in the model.

In R2012b, to generate an error whenever Simulink has to compute any of a function-call subsystem's inputs directly or indirectly during execution of the function-call subsystem, you can use the new `FcnCallInpInsideContextMsg` parameter argument setting of `EnableAllAsError`. The parameter setting is stored with the model. Set the `FcnCallInpInsideContextMsg` parameter with the **Configurations Parameters > Diagnostics > Connectivity > Context-dependent inputs** parameter.

Version History

In R2012b, the `HiliteFcnCallInpInsideContext` parameter has been removed. The new `FcnCallInpInsideContextMsg` parameter settings eliminate the need for the `HiliteFcnCallInpInsideContext` parameter, which you could not store with the model.

In R2012b, the `FcnCallInpInsideContextMsg` parameter setting of `Enable All` argument has been replaced by two settings: `EnableAllAsWarning` and `EnableAllAsError`. The `EnableAllAsError` setting is now the default. In R2012a and R2011b, `Enable All` was the default, and in R2011a and earlier, `Use local settings` was the default.

If you have existing code that set the `HiliteFcnCallInpInsideContext` parameter, you need to change that code in R2012b for the following conditions.

Existing Code	R2012b Equivalent Code
HiliteFcnCallInpInsideContext set to on FcnCallInpInsideContextMsg set to Enable All	Set FcnCallInpInsideContextMsg to EnableAllAsError and remove HiliteFcnCallInpInsideContext.
HiliteFcnCallInpInsideContext set to off FcnCallInpInsideContextMsg set to Enable All	Set FcnCallInpInsideContextMsg to EnableAllAsWarning and remove HiliteFcnCallInpInsideContext
FcnCallInpInsideContextMsg set to Use local settings	FcnCallInpInsideContextMsg set to UseLocalSettings
FcnCallInpInsideContextMsg set to Disable All	Set FcnCallInpInsideContextMsg to DisableAll.

Note The FcnCallInpInsideContextMsg settings of Use local settings and Disable all are maintained for backward compatibility, but may be deprecated in a future release. If code from before R2012b used get_param with the FcnCallInpInsideContextMsg parameter for the string comparison, then when you run that code in R2012b, the returned results of UseLocalSettings and DisableAll no longer match the Use local settings and Disable all strings in the earlier code.

The Model Advisor **Check usage of function-call connections** checks based on the settings of the **Configurations Parameters > Diagnostics > Connectivity > Invalid function-call connection** and **Configurations Parameters > Diagnostics > Connectivity > Context-dependent inputs** parameters. In R2012b, the recommended action was to set **Context-dependent inputs** to Enable All. In R2012b, the recommended action is to set it to Enable all as errors.

When you save a model that has FcnCallInpInsideContextMsg parameter set to EnableAllAsWarning or EnableAllAsError to an earlier release, Simulink saves the earlier-release model with the Enable all setting. The EnableAllAsError behavior of generating an error message is not available in the earlier-release model.

Simulink.Variant object and the model InitFcn

Simulink.Variant objects used by a model must be created before simulation is started. If you create or update a Simulink.Variant object in the model's callback InitFcn function, then at diagram update time, Simulink ignores that object creation or update.

Version History

In earlier versions of Simulink, if you created or updated Simulink.Variant objects with the InitFcn function, Simulink incorrectly processed the object creation or update, which could lead to incorrect model behavior.

Signal Management

Sample time propagation changes

The way that Simulink software uses the sample time of an enable signal during sample time propagation has been improved for models that contain enabled subsystems with:

- No Inport blocks
- All blocks inside the enabled subsystem specifying an inherited sample time

Simulink now sets the sample times of the contents of enabled subsystems with these conditions to the sample times of their Enable blocks. In previous releases, Simulink did not propagate the sample time of the enable signal to the subsystem contents. Instead, Simulink determined the sample time of the subsystem contents using backpropagation from outside the subsystem.

Version History

This change helps you avoid unintentional multirate enabled subsystems. However, if existing models have Merge blocks whose inputs are driven by enabled subsystem outputs, Model Advisor checks might return errors. Follow the Model Advisor guidelines to resolve the issue.

If you want your model to behave as before, manually set the sample times in your enabled subsystems.

Signal Builder

Signal Builder has the following changes:

- You can now import signal data formatted in a custom format to the Signal Builder block. In previous releases, you could import data only if it complied with the existing format guidelines. For more information, see [Importing Data with Custom Formats](#).
- The Signal Builder block has had minor graphical updates. For more information, see [Signal Groups](#).
- The `signalbuilder` function now enables you to get the active group label.


User Interface Enhancements

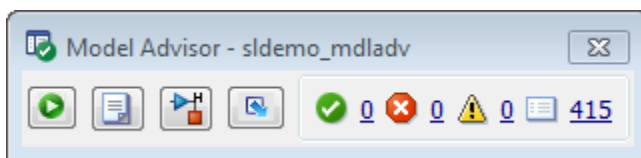
Model Advisor Dashboard


The Model Advisor dashboard provides a way for you to efficiently check that your model complies with modeling guidelines. You can use the Model Advisor dashboard to run a set of checks on your model without opening the Model Advisor window and reloading checks, saving analysis time. To open the Model Advisor dashboard, from the Model Editor, you can either:

- Select **Analysis > Model Advisor > Model Advisor Dashboard**.

-

Select Model Advisor Dashboard from the Model Editor toolbar  drop-down list.



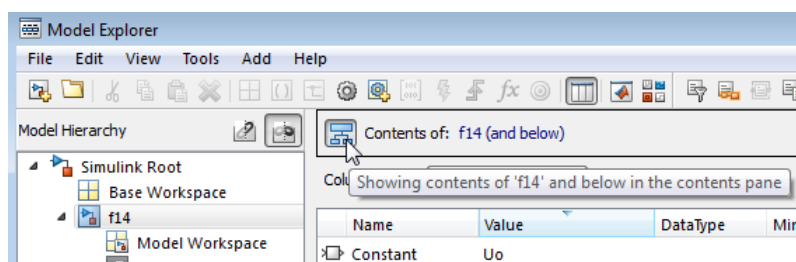
When you use the Model Advisor dashboard, you can select and view checks by clicking the **Switch to Model Advisor** toggle button (). For more information, see Overview of the Model Advisor Dashboard.

Show partial or whole model hierarchy contents

By default, the Model Explorer displays objects for the system that you select in the Model Hierarchy pane. It does not display data for child systems.

Now you can override that default, so that the Model Explorer displays objects for the whole hierarchy of the currently selected system. To toggle between displaying only the current system and displaying the whole system hierarchy of the current system. Use one of these techniques:

- Select **View > Show Current System and Below**.
- Click the **Show Current System and Below** button at the top of the **Contents** pane.



To indicate that you have selected the **Show Current System and Below** option, the Model Explorer:

- In the **Model Hierarchy** pane, highlights in pale blue the current system and its child systems
- After the path in the **Contents of** field, includes (and below)
- Changes the **Show Current System and Below** button at the top of the **Contents** pane and in the **View** menu

- In the status bar, indicates the scope of the displayed objects when you hover over the **Show Current System and Below** button

Loading very large models for the current system and below can be slow. To stop the loading process at any time, either click the **Show Current System and Below** button or click another node in the tree hierarchy.

If you show the current system and below, you may want to change the view to better reflect the displayed system contents.

The setting for the **Show Current System and Below** option is persistent across Simulink sessions.

For details, see [Displaying Partial or Whole Model Hierarchy Contents](#).

Version History

The Model Explorer search bar no longer provides the **Search Current System and Below** option. Instead, the search honors the **Show Current System and Below** setting that you set with the **View** menu or **Show Current System and Below** button.

Improved icons for model objects

Improved icons for model objects that the Model Explorer displays (for example, blocks, signals, variables) better represent the objects and are more consistent with icons used in other Simulink tools.

Simulink Debugger

The following capabilities are not available in the debugger. For more information on the debugger, see [Introduction to the Debugger](#):

- Animations
- Adding breakpoints while in the initialization phase
- Displaying I/O while in the initialization phase

Multiple modifiers for custom accelerators

You can now use multiple modifiers when defining custom accelerators. In previous releases, you used only the **Ctrl** key as a modifier for customer accelerators. In R2012b, you can use multiple modifiers for custom accelerators, for example **Ctrl+Alt+T**. See [Add Items to Model Editor Menus](#).

Model Advisor Checks

Verify Syntax of Library Models

There are Model Advisor checks available to verify the syntax of library models. When you use the Model Advisor to check a library model, the Model Advisor window indicates (~) checks that do not check libraries. To determine if you can run the check on library models, you can also refer to the check documentation, in the Capabilities and Limitations section. You cannot use checks that require model compilation. If you have a Simulink Verification and Validation license, you can use an API to create custom checks that support library models.

MATLAB Function Blocks

New toolbox functions supported for code generation

To view implementation details, see [Functions Supported for Code Generation — Alphabetical List](#).

Computer Vision System Toolbox

- `integralImage`

Image Processing Toolbox

- `bwlookup`
- `bwmorph`

Interpolation and Computational Geometry

- `interp2`

String Functions

- `deblank`
- `hex2num`
- `isletter`
- `isspace`
- `isstrprop`
- `lower`
- `num2hex`
- `strcmpi`
- `strjust`
- `strncmp`
- `strncmpi`
- `strtok`
- `strtrim`
- `upper`

Trigonometric Functions

- `atan2d`

New System objects supported for code generation

The following System objects are now supported for code generation. To see the list of System objects supported for code generation, see [System Objects Supported for Code Generation](#).

Communications System Toolbox

- `comm.ACPR`

- `comm.BCHDecoder`
- `comm.CCDF`
- `comm.CPMCarrierPhaseSynchronizer`
- `comm.GoldSequence`
- `comm.LDPCDecoder`
- `comm.LDPCEncoder`
- `comm.LTEMIMOChannel`
- `comm.MemorylessNonlinearity`
- `comm.MIMOChannel`
- `comm.PhaseNoise`
- `comm.PSKCarrierPhaseSynchronizer`
- `comm.RSDecoder`

DSP System Toolbox

- `dsp.AllpoleFilter`
- `dsp.CICDecimator`
- `dsp.CICInterpolator`
- `dsp.IIRFilter`
- `dsp.SignalSource`

R2012a

Version: 7.9

New Features

Bug Fixes

Version History

Component-Based Modeling

Interactive Library Forwarding Tables for Updating Links

Use the new Forwarding Table to map old library blocks to new library blocks. In previous releases, you could create forwarding tables for a library only at the command line. Now you can interactively create forwarding tables for a library to specify how to update links in models to reflect changes in the parameters. To set up a forwarding table for a library, select **File > Library Properties**.

You can also specify transformation functions to update old link parameter data using a MATLAB file on the MATLAB path. Transforming old link parameter data for the new library block enables you to load old links and preserve parameter data.

After you specify the forwarding table, any links to old library blocks are updated when you open a model containing links to the library. Library authors can use the forwarding tables to automatically transform old links into updated links without any loss of functionality and data. Use the forwarding table to solve compatibility issues with models containing old links that cannot load in the current version of Simulink. Library authors do not need to run `supdate` to upgrade old links, and can reduce maintenance of legacy blocks.

For details, see [Making Backward-Compatible Changes to Libraries](#) in the Simulink documentation.

Automatic Refresh of Links and Model Blocks

When you save changes to a library block, Simulink now automatically refreshes all links to the block in open Model Editor windows. You no longer need to manually select **Edit > Links and Model Blocks > Refresh**. When you edit a library block (in the Model Editor or at the command line), Simulink now indicates stale links which are open in the Model Editor by showing that the linked blocks are unavailable. When you click the Model Editor window, Simulink refreshes any stale links to edited blocks, even if you have not saved the library yet.

For details, see [Updating a Linked Block](#) in the Simulink documentation.

Model Configuration for Targets with Multicore Processors

You can now configure models for concurrent execution using configuration reference objects or configuration sets. In the previous release, you could use only configuration sets. Existing configuration sets continue to work.

The workflow to configure a model for concurrent execution has these changes:

- To preserve existing configuration settings for your model, in Model Explorer, expand the model node. Under the model, right-click **Configuration**, then select the **Show Concurrent Execution** option. This action updates the Solver pane to display a **Concurrent execution options** section.
- To create new configuration settings, in Model Explorer, right-click the model and select **Configuration > Add Configuration for Concurrent Execution**. This action updates the Solver pane to display a **Concurrent execution options section**.

The following changes have also been made:

- The **Ensure deterministic transfer (minimum delay)** data transfer now supports continuous and discrete signals. In the previous release, this data transfer type supported only continuous signals.
- Data transfer has been enhanced to allow signal branch points outside of referenced models. In previous releases, signal branching was supported only within referenced models.
- The following Simulink.SoftwareTarget.TaskConfiguration methods have new names. Use the new method names.

Old Name	New Name
addAperiodicTaskGroup	addAperiodicTrigger
deleteTaskGroup	deleteTrigger
findTaskGroup	findTrigger

- The `sldemo_concurrent_execution` demo has been updated to reflect the updated software. It also now contains an example of how to configure the model for an asynchronous interrupt.
- In the Concurrent Execution dialog box, the Map Blocks To Tasks node has changed to Tasks and Mapping. The Periodic and Interrupt nodes are now hierarchically under the Tasks and Mapping node.

For more information, see *Configuring Models for Targets with Multicore Processors* in the Simulink User's Guide.

MATLAB Function Blocks

Integration of MATLAB Function Block Editor into MATLAB Editor

There is now a single editor for developing all MATLAB code, including code for the MATLAB Function block.

Code Generation for MATLAB Objects

There is preliminary support in MATLAB Function blocks for code generation for MATLAB classes targeted at supporting user-defined System objects. For more information about generating code for MATLAB classes, see Code Generation for MATLAB Classes. For more information about generating code for System objects, see the DSP System Toolbox™, Computer Vision System Toolbox™, or the Communications System Toolbox™ documentation.

Specification of Custom Header Files Required for Enumerated Types

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

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

Version History

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

Data Management

New Infrastructure for Extending Simulink Data Classes Using MATLAB Class Syntax

Previously, you could use only the Data Class Designer to create user-defined subclasses of Simulink data classes such as `Simulink.Parameter` or `Simulink.Signal`.

The Data Class Designer, which is based on *level-1 data class infrastructure*, allows you to create, modify, or delete user-defined packages containing user-defined subclasses.

In a future release, support for level-1 data class infrastructure is being removed.

There are disadvantages to using the level-1 data class infrastructure:

- The syntax for defining data classes using this infrastructure is not documented.
- The data classes are defined in P-code.
- The infrastructure offers limited capability for defining data classes.
 - It does not allow you to add methods to your data classes.
 - It does not allow you to add private or protected properties to your data classes.
- It permits partial property matching and does not enforce case sensitivity. For example, after creating a `Simulink.Parameter` data object

```
a = Simulink.Parameter;
```

you could set the property `Value` of the data object by using the following command:

```
a.value = 5;
```

In R2012a, a replacement called *level-2 data class infrastructure* is being introduced. This infrastructure allows you to extend Simulink data classes using MATLAB class syntax.

Features of level-2 data class infrastructure:

- Ability to upgrade data classes you defined using level-1 data class infrastructure. To learn how to upgrade, see [Upgrade Level-1 Data Classes to Level-2](#).
- Complete flexibility in defining your data classes, which can now have their own methods and private properties.
- Simplified mechanism for defining custom storage classes for your data classes.
- Strict matching for properties, methods, and enumeration property values.
- Ability to define data classes as readable MATLAB code, not P-code. With class definitions in MATLAB code, it is easier to understand how these classes work, to integrate code using configuration management, and to perform peer code reviews.

See the detailed example, [Define Level-2 Data Classes Using MATLAB Class Syntax](#).

Version History

When you migrate your level-1 data classes to level-2 data classes, the way that MATLAB code is generated and model files are loaded remains the same. However, you may encounter errors if your code includes the following capabilities specific to level-1 data classes:

- Inexact property names such as `a.value` instead of the stricter `a.Value`.
- The `set` method to get a list of allowable values for an enumeration property.
- Vector matrix containing `Simulink.Parameter` and `Simulink.Signal` data objects. Previously, using level-1 data classes, you could define a vector matrix `v` as follows:

```
a = Simulink.Signal;  
b = Simulink.Parameter;  
v = [a b];
```

However, level-2 data classes do not support such mixed vector matrices.

In these cases, modify your code to replace these capabilities with those supported by level-2 data classes. For more information on how to make these replacements, see [MATLAB Object Oriented Programming](#).

Change in Behavior of `isequal`

Previously, when you used function `isequal` to compare two Simulink data objects, the function compared only the handles of the two objects. This behavior was incorrect and did not conform to the intended behavior of `isequal` in MATLAB. Consider the following example:

```
a = Simulink.Parameter;  
b = Simulink.Parameter;  
isequal(a,b)  
ans = false
```

In R2012a, the behavior of `isequal` has changed to conform to the intended behavior of `isequal` in MATLAB. Now, `isequal` compares two Simulink data objects by comparing their individual property values. Based on the above example, provided objects `a` and `b` have similar property values, the new result will be as follows.

```
a = Simulink.Parameter;  
b = Simulink.Parameter;  
isequal(a,b)  
ans = true
```

`isContentEqual` Will Be Removed in a Future Release

Previously, you could use method `isContentEqual` to compare the property values of two Simulink data objects.

In this release, the behavior of `isequal` has been changed so that it can replace `isContentEqual`.

In a future release, support for `isContentEqual` will be removed. Use `isequal` instead.

Version History

If you are using the `isContentEqual` method in your MATLAB code to compare Simulink data objects, replace all instances of `isContentEqual` with `isequal`.

Change in Behavior of int32 Property Type

Previously, when you created `int32` properties for a level-1 data class using the Data Class Designer, the property value was stored as a double-precision value.

In R2012a, the behavior of `int32` properties has changed. Now, `int32` properties for a level-2 data classes are stored as a single-precision values.

RTWInfo Property Renamed

In R2012a, the property `RTWInfo` of a Simulink data object has been renamed as `CoderInfo`.

Version History

If your code uses the `RTWInfo` property to access data object parameters such as `StorageClass`, replace instances of `RTWInfo` in your code with `CoderInfo`. Your existing code will continue to work as before.

deepCopy Method Will Be Removed in a Future Release

Previously, you could use a Simulink data object's `deepCopy` method to create a copy of the data object along with its properties.

```
a = Simulink.Parameter;
b = a.deepCopy;
```

In a future release, the `deepCopy` method will be removed. Use the `copy` method instead.

```
a = Simulink.Parameter;
b = a.copy;
```

The `copy` does not create a reference. To create a reference, use the following commands.

```
a = Simulink.Parameter;
b = a;
```

New Methods for Querying Workspace Variables

Previously, you could query model workspace variables using the `evalin` method, but you had to resave your model after using this method.

In R2012a, use two new `Simulink.Workspace` methods to query workspace variables without having to resave your model:

- `hasVariable`: Determines if a variable exists in the workspace.
- `getVariable`: Gets the value of a variable from the workspace.

Default Package Specification for Data Objects

In R2012a, you can specify a default data package other than **Simulink**. Set the default package in the Data Management Defaults pane of the Simulink Preferences dialog box.

Simulink applies your default package setting to the Model Explorer, Data Object Wizard, and Signal Properties dialog box.

Simulink.Parameter Enhancements

The following enhancements have been made to `Simulink.Parameter` data objects:

- You can now explicitly specify the data type property of a `Simulink.Parameter` object as `double`.
- When casting values to specified data types, the values of `Simulink.Parameter` objects are now cast using the casting diagnostic used for block parameters.
 - Scalar value is cast to fixed-point data type.
 - Array or matrix value is cast to fixed-point data type.
 - Structure value is cast to bus data type.

For more information on creating typesafe models, see [Data Typing Rules](#)

Custom Storage Class Specification for Discrete States on Block Dialog Box

Previously, you could specify custom storage classes (CSCs) for discrete states only by creating a signal object in the base workspace, associating it with the discrete state, and assigning the CSC to the signal object.

In R2012a, you can specify CSCs for discrete states directly on the block dialog box.

For example, you can specify a CSC for the discrete state of a Unit Delay block as follows:

- 1 Open the block dialog box.
- 2 Click the **State Attributes** tab.
- 3 Select a **Package**.
- 4 Select the desired CSC from the `Storage class` drop-down list.
- 5 Set **Custom attributes** for the storage class.

Enhancement to `set_param`

Previously, when you used `set_param` to make changes to the value of a parameter, Simulink allowed the change to be committed even if the `set_param` operation failed. Consequently, an invalid value persisted in the parameter and an error was generated during model simulation.

In this release, the behavior of `set_param` has been enhanced so that Simulink does not change the value of a parameter if the `set_param` operation fails. Instead, the parameter retains its original value.

Version History

- 1 Sections of your code might not work if they depend on the value of a parameter set using `set_param` within a `try-catch` block. Consider revising such sections of code to account for the new behavior.
- 2 If you are setting dependent parameters using separate `set_param` commands for each parameter, consider revising the code so that all dependent parameters are set using a single command. Setting individual parameters might cause the `set_param` operation to fail for the dependent parameters.

For example, consider three dependent parameters of the Integrator block: Lower Saturation, Upper Saturation, and Initial Condition. The dependency condition among these parameters is as follows: Lower Saturation \leq Initial Condition \leq Upper Saturation.

Here, setting one parameter at a time might cause the `set_param` operation to fail if a dependency condition is not satisfied. It might be better to set all parameters together using a single `set_param` command.

Avoid

```
try
    set_param(Handle, 'Param1', Value1)
end
    set_param(Handle, 'Param2', Value2)
```

Better

```
set_param(Handle, 'Param1', Value1, 'Param2', Value2)
```

Simulink.findVars Support for Active Configuration Sets

`Simulink.findVars` now searches for variables that are used in a model's active configuration set. For example, you can now use `Simulink.findVars` to search for variables that are used to specify configuration parameters such as Start time and Stop time.

Use either the Model Explorer or the `Simulink.findVars` command-line interface to search for variables used by an active configuration set.

Bus Support for To File and From File Blocks

The To File block supports saving virtual and nonvirtual bus data.

The From File block supports loading nonvirtual bus data.

Bus Support for To Workspace and From Workspace Blocks

The To Workspace block supports saving bus data, with a new default save format, `Timeseries`. For bus data, the `Timeseries` format uses a structure of MATLAB timeseries objects, and for non-bus data, a MATLAB timeseries object.

The From Workspace block now supports loading bus data. To do so, specify a bus object as the output data type.

Logging Fixed-Point Data to the To Workspace Block

If you configure the To Workspace block to log fixed-point data as `fi` objects, then the workspace variable should use the same data type as the input. To preserve the data type of scaled doubles inputs, Simulink logs them to `fi` objects.

Version History

In releases prior to R2012a, when you configured the To Workspace block to log fixed-point data as `fi` objects and the input data type was scaled doubles, then Simulink discarded the scaled doubles data type and logged the data as doubles.

Improved Algorithm for Best Precision Scaling

In R2012a, using best-precision scaling is less likely to result in proposed data types that could result in overflows. The new algorithm prevents overflows for all rounding modes except `Ceiling`.

For example, consider a Data Type Conversion block with an **Output minimum** of `-128.6`, and rounding mode `Floor`, and data type specified as `fixdt(1,8)`. In previous releases, for an input signal value of `-128.6`, best precision scaling set the output data type to `fixdt(1,8,0)` which resulted in an overflow. In R2012a, for the same model, best precision scaling now prevents such overflows by setting the output data type to `fixdt(1,8,-1)` and the signal value becomes `-128`.

Version History

Best-precision scaling in R2012a calculates a different data type from that calculated in R2011b only if the value being scaled is between $(\text{RepMin} - \text{LSB})$ and `RepMin`, where `RepMin` is the representable minimum of the proposed data type. Under these conditions, the output data type used by the block might change to avoid overflow. This change might reduce precision and result in the propagation of different data types. It might also affect the data types proposed by the Fixed-Point Advisor and Fixed-Point Tool.

Enhancement of Mask Parameter Promotion

The manner in which promoted variables are named in the mask editor has changed.

Consider the following example.

You mask a subsystem that contains two parameters that are candidates for promotion: **Upper Limit** and **Lower Limit**. You promote parameter **Upper Limit**, but later decide to promote a different parameter. So you remove your original promotion of **Upper Limit** and promote parameter **Lower Limit** instead.

In previous releases, even though you changed the promotion to parameter **Lower Limit**, the auto-generated name and prompt for this parameter remained **UpperLimit**.

In this release, when you change the promotion to parameter **Lower Limit**, the variable name and the prompt of the parameter change to **LowerLimit**. However, if you had manually changed the variable name for your originally promoted parameter **Upper Limit** to **ChangedLimit**, the variable name for the new promotion will also be **ChangedLimit**.

File Management

SLX Format for Model Files

In R2012a, Simulink provides a new option to save your model file in the SLX format, with file extension `.slx`. The SLX file format contains the same information as an MDL file and is a compressed package that conforms to the Open Packaging Conventions (OPC) interoperability standard. SLX stores model information using Unicode UTF-8 in XML and other international formats.

Saving Simulink in the SLX format:

- Typically reduces file size. The file size reduction between MDL and SLX varies depending on the model.
- Solves some problems in previous releases with loading and saving MDL files containing Korean and Chinese characters.
- Supports new features in future releases not supported with MDL format.

The default file format remains MDL, and the MDL format will remain available for the foreseeable future.

To use the SLX format, see File format for new models and libraries in the Simulink Preferences documentation.

Version History

SLX will become the default file format in a future release. In R2012a you can optionally save your models in the SLX format. All functionality and APIs that currently exist for working with models, such as the `get_param` and `set_param` commands, are also available when using the SLX file format.

The MDL file format will continue to be supported, but, after R2012a, new features might be available only if you use the SLX file format.

When you use the SLX file format, the new file extension `.slx` might cause compatibility issues if your scripts contain hard-coded references to file names with extension `.mdl`. To check for future problems, verify that your code works with both the MDL and SLX formats. If you find any places in your scripts that need to be updated, use functions like `which` and `what` instead of strings with `.mdl`.

Caution If you use third-party source control tools, be sure to register the model file extension `.slx` as a binary file format. If you do not, these third-party tools might corrupt SLX files when you submit them.

Release	Operations with Possible Compatibility Considerations	What Happens	Action
R2012a	In 12a, SLX is optional. No compatibility considerations, unless you choose to save as SLX.	Nothing, unless you choose to try SLX. If you try SLX, see the following rows for possible impact.	None.
Future release with SLX default.	Hard-coded references to file names with extension <code>.mdl</code> .	Scripts cannot find or process models saved with new file extension <code>.slx</code> .	Make your code work with both the <code>.mdl</code> and <code>.slx</code> extension. Use functions like <code>which</code> and <code>what</code> instead of strings with <code>.mdl</code> .
	Third-party source control tools that assume a text format by default.	Binary format of SLX files can cause third-party tools to corrupt the files when you submit them.	Register <code>.slx</code> as a binary file format with third-party source control tools.

For more information, see Saving Models in the SLX File Format in the Simulink documentation.

The format of content within MDL and SLX files is subject to change. Use documented APIs (such as `get_param`, `find_system`, and `Simulink.MDLInfo`) to operate on model data.

Simulink Project Enhancements

In R2012a, Simulink projects include the following enhancements:

- New export to Zip file capability to package and share project files.
- Dependency analysis graph views to visualize project file dependencies.
- Easily compare and merge project file labels and shortcuts to resolve conflicts during peer review workflow.
- New ability to load a project and use the project API to run setup tasks on a MATLAB worker.
- Extended source control support with the Source Control Adapter SDK for authoring integration with third-party tools.

For more information on using projects, see Managing Projects.

Version History

Functionality	What Happens When You Use This Functionality?	Use This Functionality Instead	Compatibility Considerations
getRootDirectory	Warns	getRootFolder	Replace all instances of getRootDirectory with getRootFolder

See the `Simulink.ModelManagement.Project.CurrentProject` reference page.

Signal Management

Signal Hierarchy Viewer

To display the signal hierarchy for a signal:

- 1 Right-click a signal.
- 2 Select the **Signal Hierarchy** option to open the new Signal Hierarchy Viewer.

For details, see Signal Hierarchy Viewer.

Signal Label Propagation Improvements

Prior to R2012a, signal label propagation behaved inconsistently in different modeling contexts. Signal label propagation is the process that Simulink uses when it passes signal labels to downstream connection blocks (for example, Subsystem and Signal Specification blocks).

In R2012a, signal label propagation is consistent:

- For different modeling constructs (for example, different kinds of signals, different kinds of buses, model referencing, variants, and libraries)
- In models with or without hidden blocks, which Simulink inserts in certain cases to enable simulation
- At model load, edit, update, and simulation times

For details, see Signal Label Propagation.

Version History

In the Signal Properties dialog box, for the **Show propagated signals** parameter, you can no longer specify the `all` option. When you save a pre-R2012a model in R2012a, Simulink changes the `all` settings to `on`.

The following blocks no longer support signal label propagation. When you open legacy models that have signal label propagation enabled for these blocks, Simulink does not display a warning or error, and does not propagate the signal label.

- Assignment
- Bus Assignment
- Bus Creator
- Bus Selector
- Demux
- Matrix Concatenate
- Mux
- Selector
- Vector Concatenate
- Bus-capable blocks (Memory, Merge, Multiport Switch, Permute Dimensions, Probe, Rate Transition, Reshape, S-Function, Switch, Unit Delay, Width, and Zero-Order Hold)

You can name the output of a Bus Creator block and choose to have that name propagated to any downstream connection blocks.

To view the hierarchy for any bus signal, use the new Signal Hierarchy Viewer.

Frame-Based Processing: Inherited Option of the Input Processing Parameter Now Provides a Warning

Some Simulink blocks are able to process both sample- and frame-based signals. After the transition to the new way of handling frame-based processing, signals will no longer carry information about their frame status. Blocks that can perform both sample- and frame-based processing will have a new parameter that allows you to specify the appropriate processing behavior.

To prepare for this change, many blocks received a new **Input processing** parameter in previous releases. You can set this parameter to `Columns as channels (frame based)` or `Elements as channels (sample based)`, depending upon the type of processing you want. The third choice, `Inherited (this choice will be removed - see release notes)`, is a temporary selection that is available to help you migrate your existing models from the old paradigm of frame-based processing to the new paradigm.

In this release, your model provides a warning when the following conditions are all met for any block in your model:

- The **Input processing** parameter is set to `Inherited (this choice will be removed - see release notes)`.
- The input signal is frame-based.
- The input signal is a vector, matrix, or N-dimensional array.

Version History

To eliminate this warning, you must upgrade your existing models using the `s_lupdate` function. The function detects all blocks that have `Inherited (this choice will be removed - see release notes)` selected for the **Input processing** parameter. It then asks you whether you would like to upgrade each block. If you select yes, the function detects the status of the frame bit on the input port of the block. If the frame bit is 1 (frames), the function sets the **Input processing** parameter to `Columns as channels (frame based)`. If the bit is 0 (samples), the function sets the parameter to `Elements as channels (sample based)`.

In a future release, the frame bit and the `Inherited (this choice will be removed - see release notes)` option will be removed. At that time, the **Input processing** parameter in models that have not been upgraded will automatically be set to either `Columns as channels (frame based)` or `Elements as channels (sample based)`. The option set will depend on the library default setting for each block. If the library default setting does not match the parameter setting in your model, your model will produce unexpected results. Additionally, after the frame bit is removed, you will no longer be able to upgrade your models using the `s_lupdate` function. Therefore, you should upgrade your existing modes using `s_lupdate` as soon as possible.

Logging Frame-Based Signals

In this release, a new warning message appears when a Simulink model is logging frame-based signals and the **Signal logging format** is set to `ModelDataLogs`. In `ModelDataLogs` mode, signals are logged differently depending on the status of the frame bit, as shown in the following table.

Status of Frame Bit	Today	When Frame Bit Is Removed
Sample-based	3-D array with samples in time in the third dimension	3-D array with samples in time in the third dimension
Frame-based	2-D array with frames in time concatenated in the first dimension	3-D array with samples in time in the third dimension

This warning advises you to switch your **Signal logging format** to `Dataset`. The `Dataset` logging mode logs all 2-D signals as 3-D arrays, so its behavior is not dependent on the status of the frame bit.

When you get the warning message, to continue logging signals as a 2-D array:

- 1 Select **Simulation > Configuration Parameters > Data Import/Export**, and change **Signal logging format** to `Dataset`. To do so for multiple models, click the link provided in the warning message.
- 2 Simulate the model.
- 3 Use the `dsp.util.getLogArray` function to extract the logged signal as a 2-D array.

Frame-Based Processing: Model Reference

In this release, the `Model` block has been updated so that its operation does not depend on the frame status of its input signals.

Version History

In a future release, signals will not have a `frameness` attribute, therefore models that use the `Model` block must be updated to retain their behavior. If you are using a model with a `Model` block in it, follow these steps to update your model:

- 1 For both the child and the parent models:
 - In the **Configuration Parameters** dialog box, select the **Diagnostics > Compatibility** pane.
 - Change the **Block behavior depends on input frame status** parameter to `warning`.
- 2 For both the child and the parent models, run the Simulink Upgrade Advisor. For details, see “Consult the Upgrade Advisor”.
- 3 For the child model only:
 - In the **Configuration Parameters** dialog box, select the **Diagnostics > Compatibility** pane.
 - Change the **Block behavior depends on input frame status** parameter to `error`.

Removing Mixed Frameness Support for Bus Signals on Unit Delay and Delay

This release phases out support for buses with mixed sample and frame-based elements on the Unit Delay and Delay blocks in Simulink. When the frame bit is removed in a future release, any Delay block that has a bus input of mixed frameness will start producing different results. This incompatibility is phased over multiple releases. In R2012a the blocks will start warning. In a future release, when the frame bit is removed, the blocks will error.

Block Enhancements

Delay Block Accepts Buses and Variable-Size Signals at the Data Input Port

In R2012a, the Delay block provides the following support for bus signals:

- The data input port `u` accepts virtual and nonvirtual bus signals. The other input ports do not accept bus signals.
- The output port has the same bus type as the data input port `u` for bus inputs.
- Buses work with:
 - Sample-based and frame-based processing
 - Fixed and variable delay length
 - Array and circular buffers

To use a bus signal as the input to a Delay block, you must specify the initial condition in the dialog box. In other words, the initial condition cannot come from the input port `x0`.

In R2012a, the Delay block also provides the following support for variable-size signals:

- The data input port `u` accepts variable-size signals. The other input ports do not accept variable-size signals.
- The output port has the same signal dimensions as the data input port `u` for variable-size inputs.

The rules for variable-size signal support depend on the input processing mode of the Delay block. See the block reference page for details.

n-D Lookup Table Block Has New Default Settings

In R2012a, the default values of the **Table data** and **Breakpoints 3** parameters of the n-D Lookup Table block have changed:

- **Table data** — `reshape(repmat([4 5 6;16 19 20;10 18 23],1,2),[3,3,2])`
- **Breakpoints 3** — `[5, 7]`

The default values of all other block parameters remain the same.

Blocks with Discrete States Can Specify Custom Storage Classes in the Dialog Box

In R2012a, the following blocks have additional parameters on the **State Attributes** tab to support specification of custom storage classes:

- Data Store Memory
- Delay
- Discrete Filter
- Discrete State-Space

- Discrete Transfer Fcn
- Discrete Zero-Pole
- Discrete-Time Integrator
- Memory
- PID Controller
- PID Controller (2 DOF)
- Unit Delay

In previous releases, specifying a custom storage class for a block required creating a signal object in the base workspace. In R2012a, you can specify the custom storage class on the **State Attributes** tab of the block dialog box.

Inherited Option of the Input Processing Parameter Now Provides a Warning

Some Simulink blocks are able to process both sample- and frame-based signals. After the transition to the new way of handling frame-based processing, signals will no longer carry information about their frame status. Blocks that can perform both sample- and frame-based processing will have a new parameter that allows you to specify the appropriate processing behavior. To prepare for this change, many blocks received a new **Input processing** parameter in previous releases. See Version 7.8 (R2011b) Simulink Software for details. You can set this parameter to `Columns as channels (frame based)` or `Elements as channels (sample based)`, depending on the type of processing you want. The third choice, `Inherited`, is a temporary selection that is available to help you migrate your existing models from the old paradigm of frame-based processing to the new paradigm.

In this release, your model will provide a warning for the following blocks when the **Input processing** parameter is set to `Inherited`, the input signal is frame-based, and the input signal is a vector, matrix, or N-dimensional array:

- Unit Delay
- Delay
- Bias
- Tapped Delay

Version History

To eliminate this warning, you must upgrade your existing models using the `slupdate` function. The function detects all blocks that have `Inherited` selected for the **Input processing** parameter, and asks you whether you would like to upgrade each block. If you select yes, the function detects the status of the frame bit on the input port of the block. If the frame bit is 1 (frames), the function sets the **Input processing** parameter to `Columns as channels (frame based)`. If the bit is 0 (samples), the function sets the parameter to `Elements as channels (sample based)`.

In a future release, the frame bit and the `Inherited` option will be removed. At that time, the **Input processing** parameter in models that have not been upgraded will automatically be set to either `Columns as channels (frame based)` or `Elements as channels (sample based)`, depending on the library default setting for each block. If the library default setting does not match the parameter setting in your model, your model will produce unexpected results. Also, after the

frame bit is removed, you will no longer be able to upgrade your models using the `supdate` function. Therefore, upgrade your existing models using `supdate` as soon as possible.

User Interface Enhancements

Model Advisor: Highlighting

When a Model Advisor analysis is complete, you can specify that the Model Advisor highlight blocks in a model diagram relevant to warning and failure conditions reported for individual Model Advisor checks. When you click a check, in the model window you can easily see which objects pass, receive a warning, or fail the check.

See Consulting the Model Advisor.

Model Explorer: Grouping Enhancements

In the object property table, you can now group data objects by the object type property. (As in earlier releases, you can also group data objects by other property columns.)

- 1 Right-click the empty heading in the first column (the column that displays icons such as the block icon (☐)).
- 2 In the context menu, select **Group By This Column**.

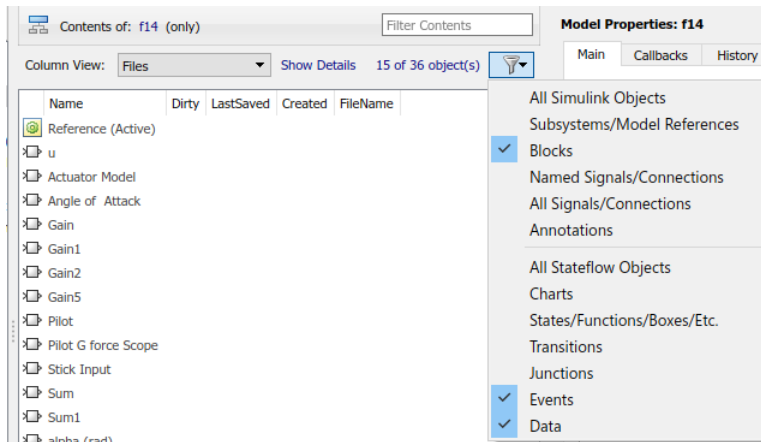
The object property table also displays the number of objects in each group.

Name	BlockType	Path	OutDataTypeStr
☐ : (1)			
Configuration (...)		f14	
☐ : (8)			
Code Generation		f14	
Data Import/Ex...		f14	
Diagnostics		f14	
Hardware Impl...		f14	
Model Referen...		f14	
Optimization		f14	
Simulation Target		f14	
Solver		f14	
☐ : (4)			
Copyright 1990...		f14	
F-14 Flight Cont...		f14	
Nz = (dq/dt*22...		f14...	
Nz pilot calcula...		f14...	

For details about grouping, see Grouping by a Property.

Model Explorer: Row Filter Button

You can access the row filter options by using the new **Row Filter** button, which is to the right of the object count at the top of the **Contents** pane.



As an alternative, you also can access the row filter options by selecting **View > Row Filter**.

For details, see Using the Row Filter Option.

Simulation Data Inspector Enhancements

Signal Data Organization

In R2012a, the **Group Signals** option allows you to customize the organization of the signal data in the Signal Browser table. By default, the data is first grouped by Run Name. You can then group the signal data by model hierarchy or by the logged variable name. Choose options that help you more easily find signals for viewing or comparing. For more information, see Modify Grouping in Signal Browser Table.

Block Name Column

The Signal Browser Table now includes a **Block Name** column. For the signal data, the **Block Name** column displays the name of the block that feeds the signal. To add this column to the table, right-click the Signal Browser table, and from the **Columns** list, select **Block Name**. For more information, see Add/Delete a Column in the Signal Browser Table.

Plot Check Box Moved

In the Signal Browser table, the plot check box is no longer in a separate **Plot** column. To select a signal for plotting, go to the left-most column where the plot check box is now located.

Parallel Simulation Support

The Simulation Data Inspector API now works with parallel simulations using the `parfor` command. To use the Simulation Data Inspector to record and view the results from parallel simulations, you can use the following methods to get and set the location of the Simulation Data Inspector repository:

- `Simulink.sdi.getSource`
- `Simulink.sdi.setSource`
- `Simulink.sdi.refresh`

For more information, see Record Data During Parallel Simulations

Port Value Displays

The behavior of port value displays for blocks has changed. In addition to performance improvements, the changes include:

- Port values are now port-based instead of block-based.
- **Block Output Display Options** dialog box has been changed to **Value Label Display Options**.
- **Show none** display option name has been changed to **Remove All**.
- You can now right-click a signal line and select **Show Port Value**. In previous releases, you enable port value displays only through the **Block Output Display Options** dialog box.
- The port value display is an empty box you toggle or hover on a block and have not yet run the simulation. In previous releases, it displayed `xx.xx`.
- The port value displays the string `wait` when you toggle or hover on a block that Simulink has optimized out of the simulation.

For more information, see [Displaying Port Values](#) in the Simulink User's Guide.

Modeling Guidelines

Modeling Guidelines for High-Integrity Systems

Following are the new modeling guidelines to develop models and generate code for high-integrity systems:

- hisl_0101: Avoid invariant comparison operations to improve MISRA-C:2004 compliance
- hisl_0102: Data type of loop control variables to improve MISRA-C:2004 compliance
- hisl_0202: Use of data conversion blocks to improve MISRA-C:2004 compliance
- hisl_0312: Specify target specific configuration parameters to improve MISRA-C:2004 compliance
- hisl_0313: Selection of bitfield data types to improve MISRA-C:2004 compliance
- hisl_0401: Encapsulation of code to improve MISRA-C:2004 compliance
- hisl_0402: Use of custom `#pragma` to improve MISRA-C:2004 compliance
- hisl_0403: Use of char data type improve MISRA-C:2004 compliance

For more information, see Modeling Guidelines for High-Integrity Systems.

MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow

The MathWorks Automotive Advisory Board (MAAB) working group created Version 2.2 of the MAAB Guidelines Using MATLAB, Simulink, and Stateflow. For more information, see MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow.

Execution on Target Hardware

New Feature for Running Models Directly from Simulink on Target Hardware

Use the new Run on Target Hardware feature to automatically run a Simulink model on target hardware.

The feature supports the following target hardware:

- BeagleBoard
- LEGO MINDSTORMS NXT

As part of the Run on Target Hardware feature, use the Target Installer to download and install support for your target hardware. To start the Target Installer, enter `targetinstaller` in a MATLAB Command Window, or open a Simulink model and select **Tools > Run on Target Hardware > Install/Update Support Package...**

When the Target Installer utility has finished, demos and a block library for your target hardware are available.

To view the demos, enter `doc` in the MATLAB Command Window. In the product help window that opens, look for **Other Demos** near the bottom, under the Contents tab.

To view the block library, enter `simulink` in the MATLAB Command Window. This action will launch the Simulink Library Browser. When the Simulink Library Browser opens, look for one of the following block libraries:

- Target for Use with BeagleBoard Hardware
- Target for Use with LEGO MINDSTORMS NXT Hardware

For more information, you can read the documentation for Running Models on Target Hardware.

R2011b

Version: 7.8

New Features

Bug Fixes

Version History

Simulation Performance

Accelerator Mode Now Supports Algebraic Loops

The Accelerator mode now works with models that contain algebraic loops. In previous releases, using Accelerator mode for models that contained algebraic loops returned error messages.

Component-Based Modeling

For Each Subsystem Support for Continuous Dynamics

For Each Subsystem blocks support continuous dynamics. This feature simplifies modeling a system of identical plant models.

The continuous dynamics support includes:

- Non-trigger sample time, including multi-rate and multitasking
- Continuous states
- Algebraic loops
- Blocks in the SimDriveline™, SimElectronics®, and SimHydraulics® products

To see an example using continuous dynamics with a For Each Subsystem block, run the `sldemo_metro_foreach` demo.

Enable Port as an Input to a Root-Level Model

You can add an Enable port to the root level of a model. The referenced model can also use a Trigger port.

Using a root-level Enable port takes advantage of model referencing benefits, without your having to do either of these extra steps:

- Put all blocks in an enabled subsystem into a referenced model
- Put the entire enabled subsystem in a referenced model

Version History

When you add an enable port to the root-level of a model, if you use the **File > Save As** option to specify a release before R2011b, then Simulink replaces the enable port with an empty subsystem.

Finder Option for Looking Inside Referenced Models

The Finder tool has a new **Look inside referenced models** option that allows you to search within a model reference hierarchy.

For details, see [Specifying Kinds of Systems To Search](#).

Improved Detection for Rebuilding Model Reference Targets

To determine when to rebuild model reference simulation and Simulink Coder targets, Simulink uses structural checksums of file contents. The use of checksums provides more accurate detection of file changes that require a rebuild. Checksum checking is particularly valuable in environments that store models in content management systems.

For details, see [Rebuild](#).

Model Reference Target Generation Closes Unneeded Libraries

When building model reference simulation and Simulink Coder targets, Simulink opens any unloaded libraries necessary for the build. Before R2011b, Simulink did not close the libraries that it opened during the build process.

In R2011b, Simulink closes all libraries no longer needed for target generation or simulation. Simulink leaves the following kinds of libraries open:

- Libraries used by referenced models running in Normal mode
- Libraries that were already open at the start of the target generation process

Concurrent Execution Support

This release extends the modeling capabilities within the Simulink product to capture and simulate the effects of deploying your design to multicore systems. In addition, you can deploy your designs to an actual multicore system using Simulink Coder, Embedded Coder, and Simulink Real-Time software. You can:

- Create a new model configuration or extend existing configurations for concurrent execution.
- Use Model blocks to define potential opportunities for concurrency in your design.
- Easily set up and configure concurrent on-target tasks using a task editing interface.
- Use either the GUI or command-line APIs to iteratively map design partitions (based on Model blocks) to tasks to find optimal concurrent execution scenarios.
- Generate code that leverages threading APIs for Windows, Linux, VxWorks®, and Simulink Real-Time platforms for concurrent on-target execution.

For further information, see *Configuring Models for Targets with Multicore Processors* in the *Simulink User's Guide*.

Finer Control of Library Links

Libraries and links have been enhanced with the following features:

- New option to lock links to libraries. Lockable library links enable control of end user editing, to prevent unintentional disabling of these links. This feature ensures robust usage of mature stable libraries.
- New check for read-only library files when you try to save, and option to try to make library writable.
- New options in Links Tool to push or restore individual edited links, in addition to existing option to push or restore entire hierarchies of links.
- `get_param` and `set_param` enhanced to perform loading of library links, making programmatic manipulation of models easier and more robust. For example, Simulink now loads links consistently if you use either `load_system` or `open_system` before using `get_param`.

For details, see *Working with Library Links* in the Simulink documentation.

Mask Built-In Blocks with the Mask Editor

You can now mask built-in blocks with the Mask Editor to provide custom icons and dialogs. In previous releases, you could mask only Subsystem, Model, and S-Function blocks. Now, in the Mask

Editor, you can choose to promote any underlying parameter of any block to the mask. For subsystems, you can choose to promote parameters from any child blocks. You can associate a single mask parameter with multiple promoted parameters if they are of the same type. Changing the value of the mask parameter also sets the value of the associated promoted parameters.

You cannot mask blocks that already have masks. For example, some Simulink blocks, such as Ramp and Chirp Signal in the Sources library, cannot be masked.

For details, see Masking Blocks and Promoting Parameters in the Simulink documentation.

Parameter Checking in Masked Blocks

Masked blocks now prevent you entering invalid parameter values by reporting an error when you edit the mask dialog values. Now the parameter checking behavior of built-in and masked blocks is unified. Both block types check for valid parameter values when you change block dialog values. Parameter checking at edit time prevents you saving a model with syntax errors or invalid values.

Previously only built-in blocks reported an error at the time you enter an invalid parameter with a syntax error, but masked blocks accepted invalid values. In previous releases you could enter invalid values in masked block dialogs and not see an error until you compiled the model. If the model was not compiled you could save a model with syntax errors or other invalid values. In R2011b, parameter checking prevents this problem.

Parameter checking applies both in the mask dialog and at the command line for invalid parameters due to syntax errors (e.g. a blank parameter or invalid parameter names). Parameter checking only applies in the mask dialog for errors defined by the block. Blocks can define valid parameters, for example, the upper limit must be higher than the lower limit, or the frequency of a signal cannot be negative etc. This type of parameter checking does not apply to changes you make at the command line. This allows you to set up blocks with multiple calls to `set_param`, without requiring that each step checks for errors.

Menu Options to Control Variants

You can now select or open Model Variants and Variant Subsystems with the **Edit** and context menus. You can use the menus to open any variant choice or override the block using any variant choice. These options were previously accessible only by opening the block dialog boxes.

For details, see Modeling Variant Systems in the Simulink documentation.

MATLAB Function Blocks

Simulation Supported When the Current Folder Is a UNC Path

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

Simulink Data Management

Default Design Minimum and Maximum are [], Not -inf/inf

In R2011b, the default design minimum and maximum values for `Simulink.Signal`, `Simulink.Parameter`, `Simulink.BusElement`, and all blocks are []/[] instead of the previous default -inf/inf. You can no longer specify design minimum and maximum values of -inf/inf for blocks and these data objects.

Version History

Simulink generates a warning or error depending on the scenario that led to -inf/inf being specified as design minimum and maximum values. The following scenarios are possible.

- When a Simulink data object is loaded from an old MAT-file or MATLAB file in which the design maximum and minimum values of the data object were specified as -inf/inf, Simulink generates a warning that -inf/inf is not supported and changes the design values to the new default, namely, []/[].
- If you set the design minimum and maximum values for the above mentioned data objects as -inf/inf, Simulink generates a warning that -inf/inf is not supported and changes the design values to the new default, namely, []/[].
- If the design minimum and maximum values evaluate to -inf/inf during compilation or at run-time, Simulink generates an error that -inf/inf is not supported.
- If your model contains an embedded signal object with design minimum and maximum values specified as -inf/inf, Simulink generates a warning that -inf/inf is not supported.

Bus Elements Now Have Design Minimum and Maximum Properties

In previous releases, you could specify design minimum and maximum for any data, including data with a bus data type. This was done by specifying the minimum and maximum parameters on the associated blocks or data objects.

In R2011b, you can specify design minimum and maximum for each element of a bus object. You can use this capability to check the values of the corresponding data elements during update diagram and simulation. With this change, Simulink no longer checks minimum or maximum specified on block dialogs or data objects for the whole bus.

Version History

If you specify the minimum or maximum for bus data on block dialogs or data objects, even if these values are scalar, Simulink generates a warning and does not use the minimum or maximum for checking the values of the corresponding data elements.

Compiled Design Minimum and Maximum Values Exposed on Block Inport and Outport

In R2011b, you can view the compiled design minimum and maximum values at a block outport from the Model Editor. See Design Ranges. In addition, you can access the compiled design minimum and

maximum values for a block's inport and outport from the command line. See Common Block Parameters.

Command-Line Interface for Accessing Compiled Design Minimum and Maximum

Use parameters `CompiledPortDesignMax` and `CompiledPortDesignMin` to access the design minimum of port signals at compile time. You must place the model in the compile state before querying this parameter. For example, to obtain the compiled design minimum at the outport of a block, use the following set of commands:

```
feval(gcs, [],[],[],'compile');
ports = get_param(gcb,'PortHandles');
outportMax = get_param(ports.Outport, 'CompiledPortDesignMax')
feval(model, [],[],[],'term');
```

`CompiledPortDesignMax` and `CompiledPortDesignMin` return different values depending on the type of signal.

- [] if none of the signals has compiled minimum or maximum
- scalar if all signals have the same specified compiled minimum or maximum
- cell array for Mux signals
- when the model is set to strict bus mode: structure for bus signals
- when the model is not set to strict bus mode: [] for virtual bus signals

Back-Propagated Minimum and Maximum of Portion of Wide Signal Are Now Ignored

In previous releases, Simulink back-propagated the design minimum and maximum of a portion of a wide signal to the source port of that portion. The back-propagated design minimum and maximum were used in range checking.

In R2011b, Simulink generates a warning and ignores the back-propagated design minimum and maximum of a portion of a wide signal during range checking.

If you want to use the back-propagated design minimum and maximum for range checking of a portion of a wide signal, insert a **Signal Conversion** block with its **Output** parameter set to **Signal copy** in front of that portion.

Easier Importing of Signal Logging Data

You can load logged signal data into a model more easily in R2011b.

You can load elements of a `Simulink.SimulationData.Signal` object. When you set the **Configuration Parameters > Data Import/Export > Signal logging format** parameter to **Dataset**, the signal logging output includes `Simulink.SimulationData.Signal` objects. You can then use the `Simulink.SimulationData.Dataset.getElement` method to specify signal elements for the **Configuration Parameters > Data Import/Export > Input** parameter.

For an example of loading logged signal data into a model, open the `sldemo_mdhref_bus` demo. For more information, see [Importing Signal Logging Data](#).

Partial Specification of External Input Data

You can load external data for a subset of root-level Inport ports, without having to create data structures for the ports for which you want to use ground values.

Using the **Configuration Parameters > Data Import/Export Input** parameter, in the comma-separated list, enter an empty matrix to specify ground values for a port.

Using an empty matrix for ports for which you want to use ground values simplifies the specification of external data to input. Also, you can use an empty matrix for an array of buses signal, which you cannot load into a root-level Inport block.

Command-Line Interface for Signal Logging


You can now use the MATLAB command line to perform the same signal logging tasks that you can perform with the Signal Logging Selector tool.

To configure signal logging from the command line, use methods for the following classes:

Simulink.SimulationData Class	Signal Logging Configuration Component
ModelLoggingInfo	Signals to log for a given simulation. Use to override the logging settings stored within a given model or referenced model.
SignalLoggingInfo	Logging settings for a single signal within a model.
LoggingInfo	Collection of signal logging properties. Use to change logging settings, such as decimation, for a signal.

For more information, see [Command-Line Interface for Overriding Signal Logging Settings](#).

Access to the Data Import/Export Pane from the Signal Logging Selector

The Signal Logging Selector toolbar includes a button () to open the **Configuration Parameters > Data Import/Export** pane. Use the **Data Import/Export** pane to configure the export of output signal and state data to the MATLAB® workspace during simulation.

Inexact Property Names for User-Defined Data Objects Will Not Be Supported in a Future Release

In previous releases, you could access a property of a user-defined data object using an inexact property name. For example, after creating a `Simulink.Parameter` data object

```
a = Simulink.Parameter;
```

you could set the property `Value` of the data object by using the following command.

```
a.v = 5;
```

In R2011b, Simulink generates a warning if you access a property of a user-defined data object using an inexact property name. While Simulink accesses the property using the inexact match, support for this type of matching will be removed in a future release.

Based on the example above, set the Value of the data object using the following command instead.

```
a.Value = 5;
```

Alias Types No Longer Supported with the `slDataTypeAndScale` Function

Simulink no longer supports calls to `slDataTypeAndScale` when:

- The first argument is a `Simulink.AliasType` object
- The first argument is a `Simulink.NumericType` object with property `IsAlias` set to true

Version History

If your model calls the internal function `slDataTypeAndScale`, you might encounter a compilation error for this model even though it previously compiled successfully. In this case, follow the advice of the error message to update your model to remove the call to `slDataTypeAndScale`.

Simulink.StructType Objects Will Not Be Supported in a Future Release

In a future release, support for `Simulink.StructType` objects will be removed. Use structured parameters or arrays of buses instead.

Old Block-specific Data Type Parameters No Longer Supported

In R2011b, Simulink generates a warning if you try to access any of these old block-specific data type parameters: `DataType`, `DataTypeMode`, `DataTypeScalingMode`, and `Scaling`. In a future release, support for these data type parameters will be removed. Use `DataTypeStr` instead.

Simulink.Signal and Simulink.Parameter Will Not Accept Input Arguments

Simulink generates an error if you pass an input argument to the classes `Simulink.Signal` and `Simulink.Parameter`.

Version History

`Simulink.Signal` and `Simulink.Parameter` classes accepted input arguments in previous versions. However, the arguments were ignored for both classes.

Data Import/Export Pane Changes

The following parameters of the **Configuration Parameters > Import/Export** pane have changed to improve their usability.

Pre-R2011b Parameter Name	Changed R2011b Parameter Name
Signal Logging Selector	Configure Signals to Log

Pre-R2011b Parameter Name	Changed R2011b Parameter Name
Return as single object	Save simulation output as single object
Inspect signal logs when simulation is paused/stopped	Record and inspect simulation output

Simulation Data Inspector Tool Replaces Time Series Tool

The Simulation Data Inspector is now the default browser for logged simulation results. Use the Simulation Data Inspector for viewing all Simulink logged data results, including as a replacement for the Time Series tool.

Version History

In R2011b, the Time Series tool (`ts_tool`) no longer supports Simulink data results.

Simulink File Management

Project Management

Organize large modelling projects with new Simulink Projects. Find all your required files, manage and share files, settings, and user-defined tasks, and interact with source control.

Projects can promote more efficient team work and local productivity by helping you:

- Find all the files that belong with your project
- Share projects using integration with external source control tool Subversion
- View and label modified files for peer review workflows
- Create standard ways to initialize and shutdown a project
- Create, store and easily access common operations

You can use projects to manage:

- Your design (.mdl, .m, .mat, and other files, source code for S-functions, data)
- The results or artifacts (simulation results, generated code, logfiles from code generation, reports).
- A set of user-defined actions to use with your project (e.g., run setup code; open models, simulate; build; run shutdown code).
- Change sets of modified files for review and interaction with source control (such as check out, compare revisions, tag or label, and check in)

For more information and a demo project to try, see [Managing Projects](#).

Simulink Signal Management

Signal Conversion Block Enhancements

The **Output** parameter of the Signal Conversion block now has a `Signal copy` option that replaces the pre-R2011b `Contiguous copy` and `Bus copy` options. The `Signal copy` option handles both non-bus and bus input signals, so that you do not need to update the setting if the input signal changes from a non-bus to a bus signal, or from a bus to a non-bus signal.

Also, setting the **Output** parameter to `Nonvirtual bus` enables the **Data type** parameter. You can use the **Data type** parameter to specify a `Simulink.Bus` object as the output data type for the Signal Conversion block. Using a bus object:

- Eliminates the need to use a `Simulink.Bus` object as the data type of an upstream Bus Creator block.
- Enables you to pass a virtual bus signal from a Bus Selector block and then create a nonvirtual bus signal.

Version History

The `Virtual bus` and `Nonvirtual bus` options for the **Output** parameter continue to work as they did in previous releases.

For models created in a release before R2011b, two compatibility issues can occur. Both of the compatibility issues occur when the Signal Conversion block a virtual bus as its input and has its **Output** parameter set to `Contiguous copy`.

The first compatibility issue occurs if the output of the Signal Conversion block has a `Simulink.Signal` object associated with it.

- Prior to R2011b, Simulink automatically performed a bus-to-vector conversion and did not report an error.
- If you open the pre-R2011b model in R2011b, then Simulink converts the `Contiguous copy` option setting to `Signal copy` and does not convert the bus signal to a vector. Because you cannot associate a `Simulink.Signal` object with a virtual bus signal, Simulink reports an error.

The second compatibility issue occurs if a virtual bus signal from a Signal Conversion block that has its **Output** parameter set to `Contiguous copy` is input to a Bus Creator block that has a `Simulink.Bus` object as its output data type.

- Prior to R2011b, Simulink considered the virtual bus signal to be a vector (as in the first compatibility issue), and did not report an error.
- If you open the model in R2011b, Simulink considers the virtual bus signal to be a bus signal. That bus signal and the bus object associated with Bus Creator block are inconsistent, so Simulink reports an error.

To avoid each of these compatibility issues, insert a Bus to Vector block at the input of the Signal Conversion block.

Environment Controller Block Support for Non-Bus Signals

You can use a non-bus signal as an input to the Environment Controller block, even if you set the **Configuration Parameters > Diagnostics > Connectivity > Non-bus signals treated as bus signals** diagnostic to error.

Sample Time Propagation Changes

The way that Simulink software propagates sample time has been improved for models with the **Optimization > Signals and Parameters > Inline parameters** check box cleared (off). This change:

- Reduces the difference in sample time propagation results between when **Inline parameters** is off and on.
- Improves the performance of your model.

Version History

This change is beneficial to the performance of your model. Not all models are affected by the sample time propagation change. To determine if your model is affected, see [Sample Time Propagation and Inline Parameters Incompatibility](#). That page provides guidelines, including information about a script, to help you evaluate your models.

To do this without the script,

- If **Inline parameters** is on for your model, your model is not affected by this change.
- If **Inline parameters** is off in your model, in R2011a or earlier, use the following procedure for each block in your model:
 - 1 With **Inline parameters** off for your model, select **Edit > Update Diagram**.
 - 2 Use `get_param` to collect the `CompiledSampleTime` value of the block.
 - 3 Turn **Inline parameters** on for your model.
 - 4 Update the diagram again.
 - 5 Use `get_param` to collect the `CompiledSampleTime` value of the block.
 - 6 Compare the results from steps 2 and 5. If they are different, and the result from step 5 is not `inf`, your model might be affected. To determine for certain if your model is affected, perform steps 1 and 2 in R2011b and compare the results with those of steps 1 and 2 from R2011a or earlier.

If you prefer the sample time propagation results from R2011a and earlier with **Inline parameters** off, you can ensure the desired sample times by manually specifying them on the affected block. If the block does not have a sample time parameter, use the Signal Specification block to specify sample times on the input or output signal.

Frame-Based Processing

In signal processing applications, you often need to process sequential samples of data at once as a group, rather than one sample at a time. Simulink documentation refers to the former as frame-based processing, and to the latter as sample-based processing. A frame is a collection of samples of data, sequential in time.

Historically, Simulink-family products that can perform frame-based processing propagate frame-based signals throughout a model. The frame status is an attribute of the signals in a model, just as data type and dimensions are attributes of a signal. The Simulink engine propagates the frame attribute of a signal by means of a frame bit, which can either be on or off. When the frame bit is on, Simulink interprets the signal as frame based and displays it as a double line, rather than the single line sample-based signal.

Beginning in R2010b, MathWorks started to significantly change the handling of frame-based processing. In the future, frame status will no longer be a signal attribute. Instead, individual blocks will control whether they treat inputs as frames of data or as samples of data. To learn how a particular block handles its input, you can refer to the block reference page.

To make the transition to the new paradigm of frame-based processing, the following Simulink blocks have a new **Input processing** parameter:

- Delay
- Detect Change
- Detect Decrease
- Detect Fall Negative
- Detect Fall Nonpositive
- Detect Increase
- Detect Rise Nonnegative
- Detect Rise Positive
- Difference
- Discrete Derivative
- Transfer Fcn Real Zero
- Unit Delay

You can specify three options with the **Input processing** parameter:

- Elements as channels (sample-based)
- Columns as channels (frame-based)
- Inherited

For more information about R2011b changes relating to frame-based processing, in the DSP System Toolbox release notes, see [Frame-Based Processing](#).

Version History

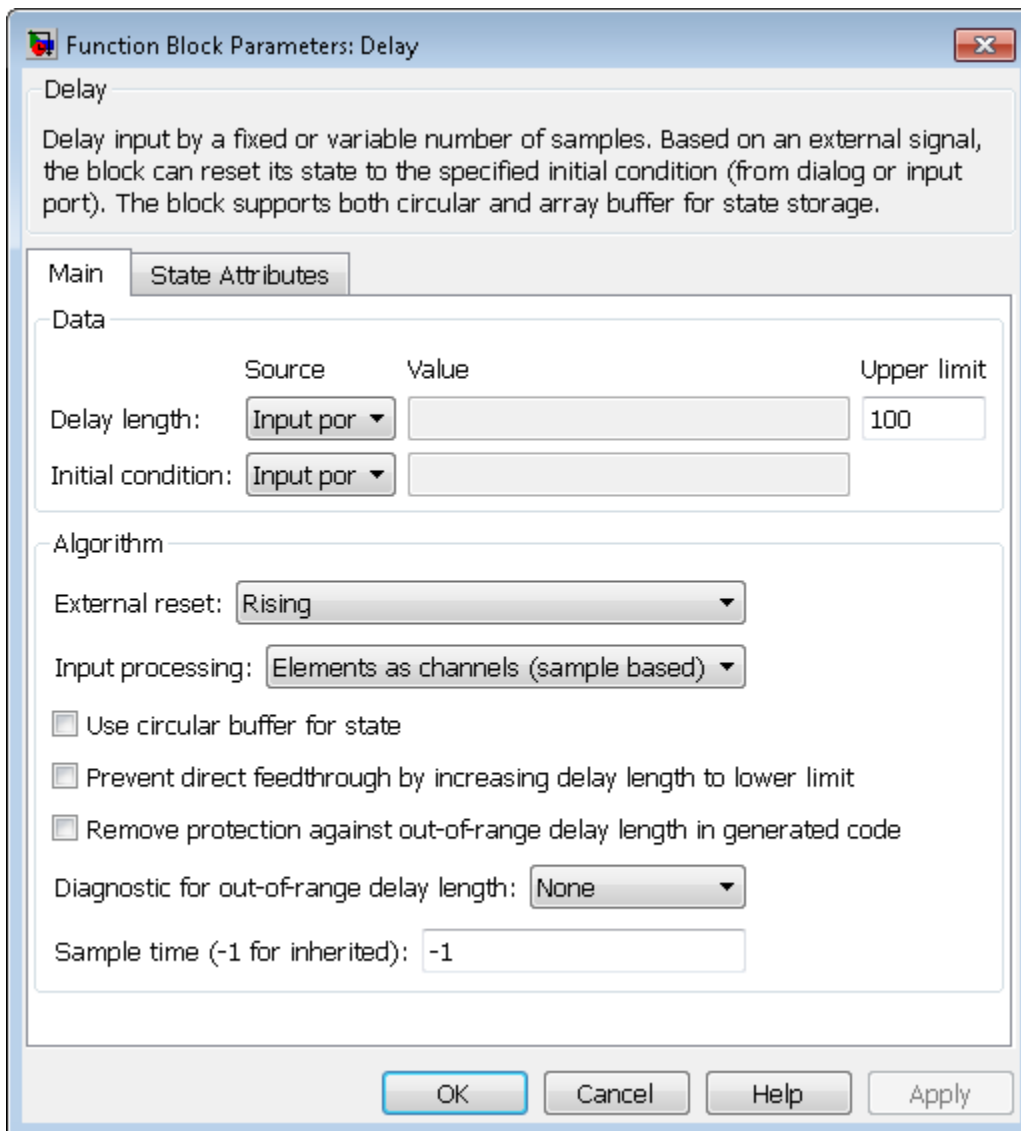
When you choose the Inherited option for the **Input processing** parameter and the input signal is frame-based, Simulink® will generate a warning or error in future releases.

Block Enhancements

New Delay Block That Upgrades the Integer Delay Block

In R2011b, the new Delay block in the Discrete library supports:

- Variable delay length
- Specification of initial condition from input port
- Reset of the state to the initial condition using an external reset signal
- State storage
- Use of a circular buffer instead of an array buffer for state storage



When you open models created in previous releases, the new Delay block replaces each instance of the Integer Delay block, which no longer appears in the Discrete library. The Delay block is an

upgrade of the Integer Delay block. Every parameter from the Integer Delay block maps directly to a parameter in the Delay block.

Version History

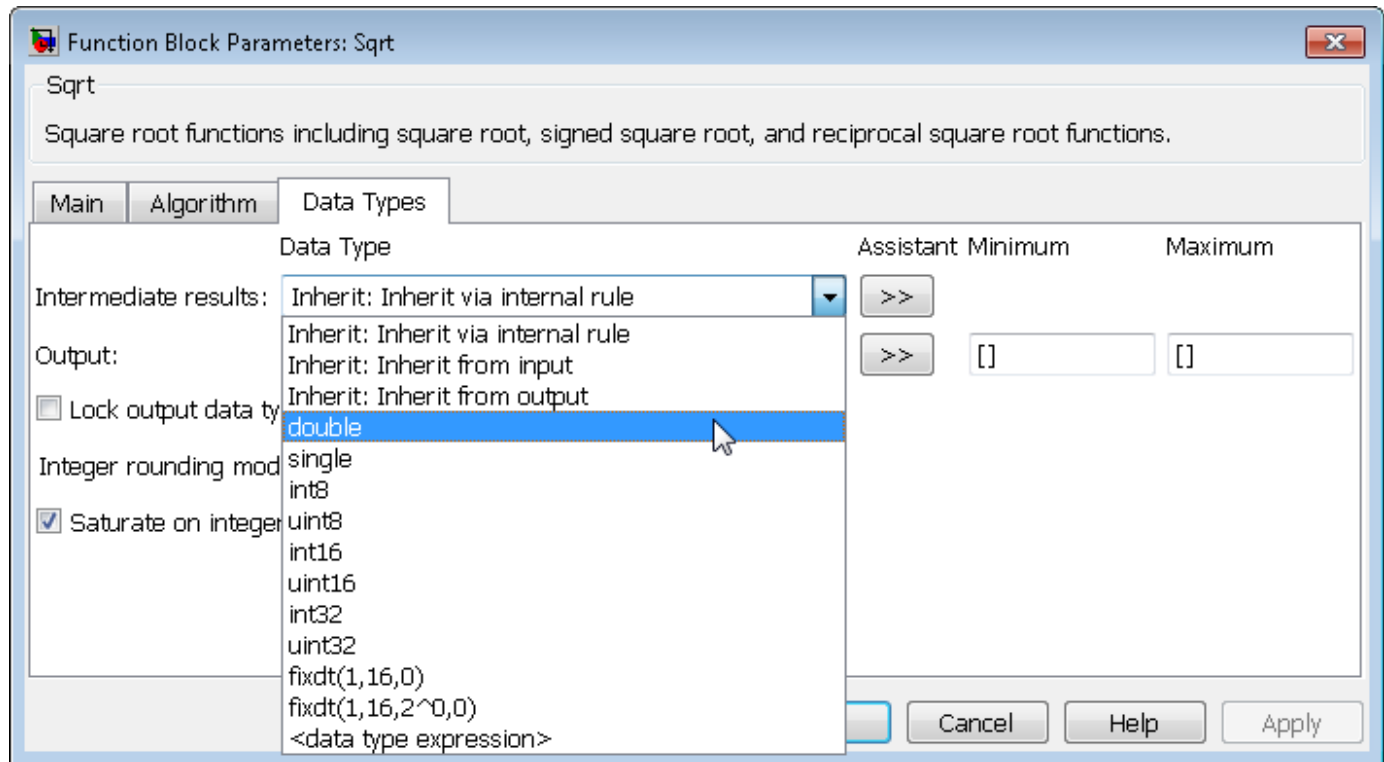
The following incompatibilities might affect simulation of pre-R2011b models that use the Integer Delay block:

Source of Incompatibility	Behavior in the Integer Delay Block	Behavior in the Delay Block	Rationale	How to Avoid an Error
Initial condition for input signals of N-by-1 or 1-by-N dimensions and sample-based processing	Suppose that delay length is D. For Initial condition , the Integer Delay block supports signal dimensions of D-by-N and N-by-D.	For Initial condition , the Delay block supports signal dimensions of N-by-1-by-D or 1-by-N-by-D. Using any other format causes an error during simulation.	The Delay block prevents misinterpretation of the dimensions for Initial condition by accepting only one format for signal dimensions.	Verify that Initial condition uses N-by-1-by-D or 1-by-N-by-D for the format of signal dimensions.
Initial condition for input signals of M-by-N dimensions and sample-based processing	Suppose that the delay length is D. For Initial condition , the Integer Delay block supports signal dimensions of D-by-M-by-N, M-by-N-by-D, and M-by-D-by-N.	For Initial condition , the Delay block supports signal dimensions of M-by-N-by-D. Using any other format causes an error during simulation.	The Delay block prevents misinterpretation of the dimensions for Initial condition by accepting only one format for signal dimensions.	Verify that Initial condition uses M-by-N-by-D for the format of signal dimensions.
Sample time	For Sample time , the Integer Delay block supports 0. In this case, the block output has continuous sample time, but fixed in minor time step.	Setting Sample time to 0 for the Delay block causes an error during simulation.	Because the Delay block belongs to the Discrete library, it should not support continuous sample time.	Use a discrete sample time, or set Sample time to -1 to inherit the sample time.
Rate transition usage	The Integer Delay block handles rate transitions for sample- and frame-based signals.	The Delay block handles rate transitions only for sample-based signals. For frame-based signals, simulation stops due to an error.	This usage of the Delay block is not recommended.	Do not use the Delay block for rate transitions with frame-based signals.

Sqrt and Reciprocal Sqrt Blocks Support Explicit Specification of Intermediate Data Type

In R2011b, both the Sqrt and Reciprocal Sqrt blocks enable specifying the data type for intermediate results. In previous releases, specifying this data type was available for the Reciprocal Sqrt block, but not the Sqrt block.

The Reciprocal Sqrt block now provides additional options for specifying the data type for intermediate results:



This enhancement enables explicit specification of the data type. In previous releases, specification of this data type was limited to inheritance rules.

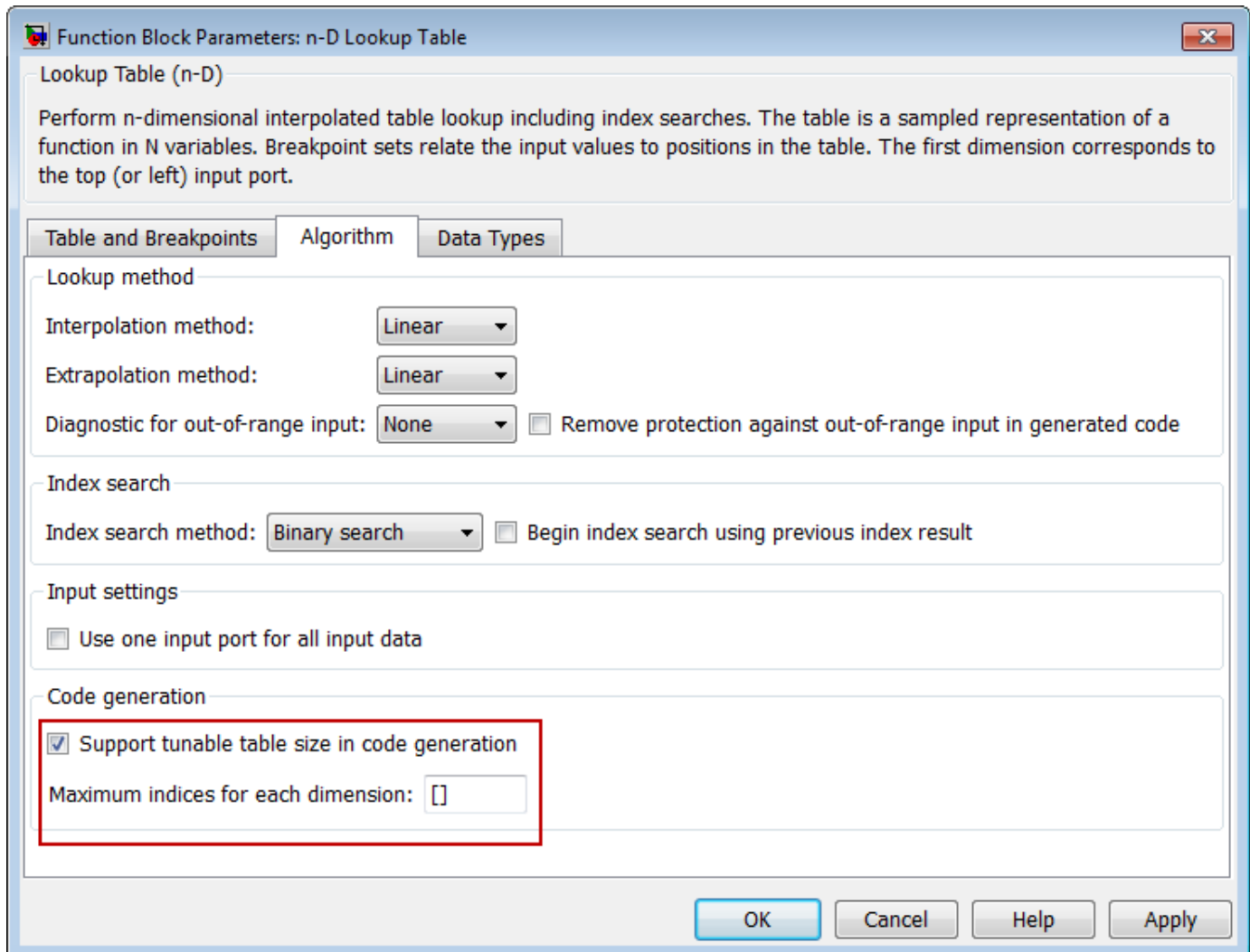
For a summary of data type configurations that are valid (input, output, and intermediate results), refer to the block reference page.

Discrete Zero-Pole Block Supports Single-Precision Inputs and Outputs

The Discrete Zero-Pole block now accepts and outputs signals of `single` data type.

n-D Lookup Table Block Supports Tunable Table Size

The n-D Lookup Table block provides new parameters for specifying a tunable table size in the generated code.



This enhancement enables you to change the size and values of your lookup table and breakpoint data without regenerating or recompiling the code.

Boolean Output Data Type Support for Logic Blocks

The following blocks now enable specification of **Output data type**, which can be `uint8` or `boolean`:

- Detect Change
- Detect Decrease
- Detect Fall Negative
- Detect Fall Nonpositive
- Detect Increase
- Detect Rise Nonnegative
- Detect Rise Positive

This enhancement enables you to specify the output data type to be `boolean`. In previous releases, the blocks always used `uint8` for the output data type.

Derivative Block Parameter Change

The block **Linearization Time Constant $s/(Ns + 1)$** parameter has changed to **Coefficient c in the transfer function approximation $s/(c.s + 1)$ used for linearization**. Correspondingly, the command-line parameter has changed from `LinearizePole` to `CoefficientInTFapproximation`.

User Interface Enhancements

Model Explorer: First Two Columns in Contents Pane Remain Visible

In the object property table, the behavior of the first two columns (the object icon and the Name property) has changed. These columns now remain visible, regardless of how far you scroll to the right. For an example that illustrates this feature, see Horizontal Scrolling in the Object Property Table.

Model Explorer: Subsystem Code View Added

Model Explorer provides an additional **Column View** option: Subsystem Code. The Subsystem Code view displays Subsystem block code generation properties. For details about views, see The Model Explorer: Controlling Contents Using Views.

Model Explorer: New Context Menu Options for Model Configurations

R2011b provides new context menu options for a model in the Model Hierarchy pane. A new menu option, **Configuration**, organizes previous and new model configuration operations. To view these configuration options, in the Model Hierarchy pane, right-click a model node and select **Configuration**. The following table describes the available configuration options.

To...	Select...
Load an existing configuration set to the model	Import
Save the model's active configuration set to a: <ul style="list-style-type: none"> • .m file (as MATLAB function or script) or • .mat file (Simulink.ConfigSet object) 	Export Active Configuration Set
Attach a new configuration set to the model	Add Configuration
Create a configuration reference and attach it to the model	Add Configuration Reference
Create a concurrent execution configuration set	Add Configuration for Concurrent Execution
Convert the model's active configuration set to a configuration reference, which then becomes active for the model	Convert Active Configuration to Reference

Two new context menu options are available for a configuration set node under a model node in the Model Hierarchy pane.

To...	Use...
Convert an active configuration set to a configuration reference	Convert to Configuration Reference (only enabled for active configuration sets)
Convert a configuration set to a concurrent execution configuration set	Convert to Configuration for Concurrent Execution

In R2011b, if a model has an active configuration reference, you can create a copy of the configuration reference for each referenced model. To perform this operation, in the Model Hierarchy

pane, right-click the active configuration reference node and select **Propagate to Referenced Models**.

For more information on model configurations, see [Managing Model Configurations](#).

Simulation Data Inspector Enhancements

Command-Line Interface

The Simulation Data Inspector command-line interface is now available to view and compare signal data, and compare two simulation runs of data. For more information, see [Record and Inspect Signal Data Programmatically](#).

Report Generation

Using the Simulation Data Inspector tool or the command-line interface, you can now generate a report of a Simulation Data Inspector session. To generate a report using the GUI, see [Create Simulation Data Inspector Report](#). To generate a report using the command-line interface, see the `Simulink.sdi.report` function.

Support of Scope, To File, and To Workspace Blocks

The Simulation Data Inspector now supports output from the following blocks:

- Scope and Floating Scope (Structure with time and Array format)
- To File (Timeseries format)
- To Workspace (Structure with time format)

Conversion of Error and Warning Message Identifiers

For R2011b, error and warning message identifiers have changed in Simulink.

Version History

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages, or in code that uses a `try/catch` statement and performs an action based on a specific error identifier.

For example, the `MATLAB:eigs:NonPosIntSize` identifier has changed to `MATLAB:eigs:RoundNonIntSize`. If your code checks for `MATLAB:eigs:NonPosIntSize`, you must update it to check for `MATLAB:eigs:RoundNonIntSize` instead.

To determine the identifier for a warning, run the following command just after you see the warning in the MATLAB Command Window.

```
[MSG,MSGID] = lastwarn;
```

This command saves the message identifier to the variable `MSGID`.

To determine the identifier for an error, run the following command just after you see the error in the MATLAB Command Window.

```
exception = MException.last;  
MSGID = exception.identifier;
```

Note Warning messages indicate a potential issue with your code. While you can turn off a warning, a suggested alternative is to change your code so it runs warning-free.

New Modeling Guidelines

Modeling Guidelines for High-Integrity Systems

Following are the new modeling guidelines to develop models and generate code for high-integrity systems:

- hisl_0201: Define reserved keywords to improve MISRA-C:2004 compliance
- hisf_0211: Protect against use of unary operators in Stateflow Charts to improve MISRA-C:2004 compliance
- hisf_0212: Data type of Stateflow for loop control variables to improve MISRA-C: 2004 compliance
- hisf_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA-C: 2004 compliance

Following are the new high-integrity modeling guidelines for configuration parameter diagnostics:

- hisl_0301: Configuration Parameters > Diagnostics > Compatibility
- hisl_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters
- hisl_0303: Configuration Parameters > Diagnostics > Data Validity > Merge block
- hisl_0304: Configuration Parameters > Diagnostics > Data Validity > Model Initialization
- hisl_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging
- hisl_0306: Configuration Parameters > Diagnostics > Connectivity > Signals
- hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses
- hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls
- hisl_0309: Configuration Parameters > Diagnostics > Type Conversion
- hisl_0310: Configuration Parameters > Diagnostics > Model Referencing
- hisl_0311: Configuration Parameters > Diagnostics > Stateflow

For more information, see Modeling Guidelines for High-Integrity Systems.

Modeling Guidelines for Code Generation

Following are the new modeling guidelines for code generation:

- cgsl_0104: Modeling global shared memory using data stores
- cgsl_0105: Modeling local shared memory using data stores

For more information, see Modeling Guidelines for Code Generation.

R2011a

Version: 7.7

New Features

Bug Fixes

Version History

Simulation Performance

Restore SimState in Models Created in Earlier Simulink Versions

Simulink 7.7 supports the restoring of a SimState from a MAT file saved in a previous version of Simulink. During this operation, Simulink restores as much of the SimState object as possible and automatically resets the simulation start time to the stop time of the SimState object.

You can choose to receive a warning or an error by setting a new diagnostic, **SimState object from earlier release**, on the Diagnostic Pane of the Configuration Parameters dialog.

Improved Absolute Tolerance Implementation

The processing of the absolute tolerance parameter in the Solver configuration pane, and of the absolute tolerance parameters for continuous blocks and S-functions with continuous states, has been enhanced. As a result, these parameters provide a more robust and consistent behavior. These error tolerances are used by variable-step solvers to control integration error for continuous states in a model.

A new SimStruct function `ssSetStateAbsTol` has been introduced to allow for setting the absolute tolerances for the S-Function continuous states in models using a variable-step solver. Use of `ssGetAbsTol` to either get or set absolute tolerances is not recommended. Instead, use `ssGetStateAbsTol` and `ssSetStateAbsTol` to get and set tolerances, respectively.

Component-Based Modeling

Refreshing Linked Blocks and Model Blocks

You can refresh linked blocks and Model blocks in a library or model using the Simulink Editor. Select the **Edit > Links and Model Blocks > Refresh**.

Refreshing the linked blocks updates the linked blocks to reflect any changes to the original library block. In releases before R2011a, to update linked blocks, you had to take one of the following actions:

- Close and reopen the library that contains the linked blocks that you want to refresh.
- Update the diagram (**Edit > Links and Update Diagram** or **Ctrl+D**).

You can update a specific Model block by right-clicking the Model block and selecting **Refresh**.

Version History

The new menu option, **Edit > Links and Model Blocks > Refresh** menu item replaces **Edit > Model Blocks > Refresh Model Blocks**. Both the old and new options update Model blocks in the same way.

Enhanced Model Block Displays Variant Model Choices

The Model Variants block now displays model names for all variant choices, making it easier to select and configure available variants.

See Setting Up Model Variants.

Creating a Protected Model Using the Simulink Editor

You can protect a model using the Simulink Editor. Right-click the Model block that references the model for which you want to generate protected model code. In the context menu, select **Code Generation > Generate Protected Model**. For details, see Creating a Protected Model.

In earlier releases, you had to use the `Simulink.ModelReference.protect` command to create a protected model.

MATLAB Function Blocks

Embedded MATLAB Function Block Renamed as MATLAB Function Block

In R2011a, Embedded MATLAB Function blocks were renamed as MATLAB Function blocks in Simulink models. The block also has a new look:



Version History

If you have scripts that refer to Embedded MATLAB library blocks by path, you need to update the script to reflect the new block name. For example, if your script refers to `simulink/User-Defined Functions/Embedded MATLAB Function` or `eml_lib/Embedded MATLAB Function`, change Embedded MATLAB Function to MATLAB Function.

Support for Buses in Data Store Memory

MATLAB Function blocks now support buses as shared data in Data Store Memory blocks.

Simulink Data Management

Signal Logging Selector

The Signal Logging Selector is a new centralized signal logging tool for:

- Reviewing all signals in a model hierarchy that are configured for logging (set with the Signal Properties dialog box)
- Overriding signal logging settings for specific signals
- Controlling signal logging throughout a model reference hierarchy in a more streamlined way than in previous releases

You can use the Signal Logging Selector with Simulink and Stateflow signals.

To open the Signal Logging Selector, in the **Configuration Parameters > Data Import/Export** pane, select the **Signal Logging Selector** button. For a Model block, you can right-click the block and select the **Log Referenced Signals** menu item. (The Signal Logging Selector replaces the Model Reference Signal Logging dialog box.)

See [Overriding Signal Logging Settings and Using the Signal Logging Selector to View the Signal Logging Configuration](#).

Dataset Format Option for Signal Logging Data

You can now select a format for signal logging data. Use the **Configuration Parameters > Data Import/Export > Signal logging format** parameter to select the format:

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

The `Dataset` format:

- Uses MATLAB `timeseries` objects to store logged data (rather than `Simulink.Timeseries` and `Simulink.TsArray` objects). MATLAB `timeseries` objects allow you to work with logging data in MATLAB without a Simulink license.
- Supports logging multiple data values for a given time step, which can be important for Iterator subsystem and Stateflow signal logging.
- Provides an easy-to-analyze format for logged signal data for models with deep hierarchies, bus signals, and signals with duplicate or invalid names.
- Supports the Simulation Data Inspector.
- Avoids the limitations of the `ModelDataLogs` format. For example, for a virtual bus, `ModelDataLogs` format logs only one of multiple signals that share the same source block. For a description of `ModelDataLogs` format limitations, see Bug Report 495436.

To convert a model that contains referenced models to use the `Dataset` format throughout the model reference hierarchy, use the `Simulink.SimulationData.updateDatasetFormatLogging` function.

If you have logged signal data in the `ModelDataLogs` format, you can use the `Simulink.ModelDataLogs.convertToDataset` function to convert the `ModelDataLogs` data to `Dataset` format.

To work with `Dataset` format data, you can use properties and methods of the following classes:

- `Simulink.BlockPath`
- `Simulink.SimulationData.BlockPath`
- `Simulink.SimulationData.Dataset`
- `Simulink.SimulationData.Signal`
- `Simulink.SimulationData.DataStoreMemory`

For information about the signal logging format, see [Specifying the Signal Logging Data Format](#)

From File Block Supports Zero-Crossing Detection

The From File block allows you to specify zero-crossing detection.

Signal Builder Block Now Supports Virtual Bus Output

You can now define the type of output to use on the Signal Builder block now outputs signals. With this release, the Signal Builder block has two options:

- **Ports**

Sends individual signals from the block. An output port named `Signal N` appears for each signal `N`. This option is the default setting. In previous releases, the block uses this type of signal output.
- **Bus**

Sends single, virtual, nonhierarchical bus of signals from the block. An output port named `Bus` appears. This `Bus` option enables you to change your model layout without having to reroute Signal Builder block signals. You cannot use this option to create a bus of nonvirtual signals.

For more information, see [Defining Signal Output in the Simulink User's Guide](#)

Signal Builder Block Now Shows the Currently Active Group

The Signal Builder block now shows the currently active group on its block mask.

signalbuilder Function Change

The `signalbuilder` function has a new command, `'annotategroup'`. This command enables the display of the current group name on the Signal Builder block mask.

Range-Checking Logic for Fixed-Point Data During Simulation Improved

The logic that Simulink uses to check whether design minimum and maximum values are within the specified data type range is now consistent with the logic that it uses to calculate best-precision scaling.

- Simulink now checks both real-world values and quantized values for a block parameter, `Simulink.Parameter` object, or `Simulink.Signal` object against design minimum and maximum values. Prior to R2011a, Simulink checked only real-world values against design minimum and maximum values.
- When Simulink checks the design minimum and maximum values for a `Simulink.Signal` object against the data type minimum and maximum values, it obtains the data type range in one of the following ways.
 - 1 If the data type for a `Simulink.Signal` object is set, Simulink uses the range defined in the specification of that data type
 - 2 If the data type for a `Simulink.Signal` object is set to `auto`, Simulink uses the range for the data type inferred from the initial value of the signal's `fi` object

Prior to R2011a, Simulink only used the data type range defined in the specification of that data type.

- Simulink now checks the run-time parameter value of an S-function against the design minimum and maximum values when the parameter is updated at run-time and during compilation. Prior to R2011a, Simulink checked run-time parameter values of an S-function against the design minimum and maximum only at run-time.

For more information about block parameter range checking, see [Checking Parameter Values](#).

Version History

- An error is generated if the quantized value of a block parameter, `Simulink.Parameter` object, or `Simulink.Signal` object in your model is different from the real-world value and if this difference causes the quantized value to lie outside the design minimum and maximum range.
- An error is generated if the initial value of a `Simulink.Signal` object in your model is a `fi` object and if this initial value is outside the range associated with that `fi` object.
- An error is generated at compile time if the run-time parameter value of an S-function in your model is outside the design minimum and maximum range.

Data Object Wizard Now Supports Boolean, Enumerated, and Structured Data Types for Parameters

In this release, the Data Object Wizard is enhanced to suggest parameter objects for variables with the following data types:

- Boolean
- Enumerations
- Structures

For information, see [Working with Data Objects](#) and [Data Object Wizard](#).

Error Now Generated When Initialized Signal Objects Back Propagate to Output Port of Ground Block

Prior to this release, Simulink generated an error when the output of a Ground block was a signal object with an initial value, but did not do the same for such signal objects back propagated to the output port of a Ground block. As of R2011a, Simulink generates an error under both conditions.

No Longer Able to Set RTWInfo or CustomAttributes Property of Simulink Data Objects

You can no longer set the `RTWInfo` or `CustomAttributes` property of a Simulink data object from the MATLAB Command Window or a MATLAB script. Attempts to set these properties generate an error.

Although you cannot set `RTWInfo` or `CustomAttributes`, you can still set subproperties of `RTWInfo` and `CustomAttributes`.

Version History

Operations from the MATLAB Command Window or a MATLAB script, which set the data object property `RTWInfo` or `CustomAttributes`, generate an error.

For example, a MATLAB script might set these properties by copying a data object as shown below:

```
a = Simulink.Parameter;
b = Simulink.Parameter;
b.RTWInfo = a.RTWInfo;
b.RTWInfo.CustomAttributes = a.RTWInfo.CustomAttributes;
.
.
.
```

To copy a data object, use the object's `deepCopy` method.

```
a = Simulink.Parameter;
b = a.deepCopy;
.
.
.
```

Global Data Stores Now Treat Vector Signals as One or Two Dimensional

Simulink now uses the **Dimensions** attribute of a source signal object to determine whether to register a global data store as a vector (1-D) or matrix (2-D). For example, if the **Dimensions** attribute of a source signal object is set to `[1 N]` or `[N 1]`, Simulink registers the global data store as a matrix. Prior to R2011a, Simulink treated all global data stores as vectors.

The following table lists possible signal object dimension settings with what Simulink registers for a corresponding global data store:

Source Signal Object Dimensions	Registered for Global Data Store
-1	Get dimensions from <i>InitialValue</i> and interpret vectors as 1-D
N	Vector with N elements
[1 N]	1xN matrix
[N 1]	Nx1 matrix

Version History

If you specify the dimensions of the source signal object for a global data store as [1 N] or [N 1], Simulink now registers the data store as a matrix. Although this change has no impact on numeric results of simulation or execution of generated code, the change can affect the following:

- Propagation of dimensions (for example, signals might propagate as [1 N] or [N 1] instead of N).
- Signal and state logging
 - Vectors are logged as 2D matrices - [*nTimeSteps* *width*]
 - 2-D matrices are logged as 3-D matrices - [M N *nTimeSteps*]

No Longer Able to Use Trigger Signals Defined as Enumerations

You can no longer use trigger signals that are defined as enumerations. A trigger signal represents an external input that initiates execution of a triggered subsystem. Prior to R2011a, Simulink supported enumerated trigger signals for simulation, but produced an error during code generation. This change clarifies triggered subsystem modeling semantics by making them consistent across simulation and code generation.

Version History

Use of enumerated trigger signals during simulation now generates an error. To work around this change, compare enumeration values, as appropriate, and apply the resulting Boolean or integer signal value as the subsystem trigger.

Conversions of Simulink.Parameter Object Structure Field Data to Corresponding Bus Element Type Supported for double Only

If you specify the `DataType` field of a `Simulink.Parameter` object as a bus, you must specify `Value` as a numeric structure. Prior to R2011a, Simulink would convert the data types of all fields of that structure to the data types of corresponding bus elements. As of R2011a, Simulink converts the data type of structure fields of type `double` only. If the data type of a field of the structure does not match the data type of the corresponding bus element and is not `double`, an error occurs.

This change does not affect the `InitialValue` field of `Simulink.Signal` objects. Data types of fields of a numeric structure for an initial condition *must* match data types of corresponding bus elements.

Version History

If the data type of a field of a numeric structure that you specify for `Simulink.Parameter` does not match the data type of the corresponding bus element and is not `double`, an error occurs. To correct the condition, set the data types of all fields of the structure to match the data types of all bus elements or set them to type `double`.

For more information, see `Simulink.Parameter`.

Simulink.CustomParameter and Simulink.CustomSignal Data Classes To Be Deprecated in a Future Release

In a future release, data classes `Simulink.CustomParameter` and `Simulink.CustomSignal` will no longer be supported because they are equivalent to `Simulink.Parameter` and `Simulink.Signal`.

Version History

If you use the data class `Simulink.CustomParameter` or `Simulink.CustomSignal`, Simulink posts a warning that identifies the class and describes one or more techniques for eliminating it. You can ignore these warnings in R2011a, but consider making the described changes now because the classes will be removed in a future release.

Parts of Data Class Infrastructure No Longer Available

Simulink has been generating warnings for usage of the following data class infrastructure features for several releases. As of R2011a, the features are no longer supported.

- Custom storage classes not captured in the custom storage class registration file (`csc_registration`) - *warning displayed since R14SP2*
- Built-in custom data class attributes `BitFieldName` and `FileName+IncludeDelimiter` - *warning displayed since R2008b*

Instead of...	Use...
<code>BitFieldName</code>	<code>StructName</code>
<code>FileName+IncludeDelimiter</code>	<code>HeaderFile</code>

- Initial value of MPT data objects inside `mpt.CustomRTWInfoSignal` - *warning displayed since R2006a*

Version History

- When you use a removed feature, Simulink now generates an error.
- When loading a MAT-file that uses an unsupported feature, the load operation suppresses the generated error such that it is not visible. In addition, MATLAB silently deletes data that had been associated with the unsupported feature. To prevent loss of data when loading a MAT-file, load and resave the file with R2010b or earlier.

Simulink Signal Management

Data Store Support for Bus Signals

The following blocks support the use of bus and array of buses signals with data stores:

- Data Store Memory
- Data Store Read
- Data Store Write

Benefits of using buses and arrays of buses with data stores include:

- Simplifying the model layout by associating multiple signals with a single data store
- Producing generated code that represents the data store data as structures that reflect the bus hierarchy
- Writing to and reading from data stores without creating data copies, resulting in more efficient data access

For details, see [Using Data Stores with Buses and Arrays of Buses](#).

Version History

Pre-R2011a models that use data stores work in R2011a without any modifications.

To save a model that uses buses with data stores to a pre-R2011a version, you need to restructure that model to not rely on using buses with data stores.

Accessing Bus and Matrix Elements in Data Stores

You can select specific bus or matrix elements to read from or write to a data store. To do so, use the **Element Selection** pane of the Data Store Read block and the **Element Assignment** pane of the Data Store Write block. Selecting bus or matrix elements offers the following benefits:

- Reducing the number of blocks in the model. For example, you can eliminate a Data Store Read and Bus Selector block pair or a Data Store Write and Bus Assignment block pair for each specific bus element that you want to access.
- Faster simulation of models with large buses and arrays of buses.

See [Accessing Data Stores with Simulink Blocks](#).

Array of Buses Support for Permute Dimensions, Probe, and Reshape Blocks


The following blocks now support the use of an array of buses as an input signal:

- Permute Dimensions
- Probe
- Reshape


For details about arrays of buses, see [Combining Buses into an Array of Buses](#).

Using the Bus Editor to Create Simulink.Parameter Objects and MATLAB Structures

You can use the Bus Editor to:

- Define or edit a `Simulink.Parameter` object with a bus object for its data type. In the Bus Editor, select the parameter and use one of these approaches:
 - Select the **File > Create/Edit a Simulink.Parameter object** menu item.
 - Click the **Create/Edit a Simulink.Parameter object** icon () from the toolbar.

You can then edit the `Simulink.Parameter` object in the MATLAB Editor.

- Invoke the `Simulink.Bus.createMATLABStruct` function for a bus object for which you want to create a full MATLAB structure. In the Bus Editor, select the bus object and use one of these approaches:
 - Select the **File > Create a MATLAB structure** menu item.
 - Click the **Create a MATLAB structure** icon () from the toolbar.

You can then edit the MATLAB structure in the MATLAB Editor.

Block Enhancements

Lookup Table, Lookup Table (2-D), and Lookup Table (n-D) Blocks Replaced with Newer Versions in the Simulink Library

In R2011a, the following lookup table blocks have been replaced with newer versions, which differ from the previous versions as follows:

Block	Enhancements to the Previous Version	Other Changes
Lookup Table	<ul style="list-style-type: none"> • Default integer rounding mode changed from Floor to Simplest • Support for the following features: <ul style="list-style-type: none"> • Specification of parameter data types different from input or output signal types • Reduced memory use and faster code execution for nontunable breakpoints with even spacing • Cubic-spline interpolation and extrapolation • Table data with complex values • Fixed-point data types with word lengths up to 128 bits • Specification of data types for fraction and intermediate results • Specification of index search method • Specification of diagnostic for out-of-range inputs 	<ul style="list-style-type: none"> • Block renamed as 1-D Lookup Table • Icon changed
Lookup Table (2-D)	<ul style="list-style-type: none"> • Default integer rounding mode changed from Floor to Simplest • Support for the following features: <ul style="list-style-type: none"> • Specification of parameter data types different from input or output signal types • Reduced memory use and faster code execution for nontunable breakpoints with even spacing • Cubic-spline interpolation and extrapolation • Table data with complex values • Fixed-point data types with word lengths up to 128 bits • Specification of data types for fraction and intermediate results • Specification of index search method • Specification of diagnostic for out-of-range inputs • Check box for Require all inputs to have the same data type now selected by default 	<ul style="list-style-type: none"> • Block renamed as 2-D Lookup Table • Icon changed
Lookup Table (n-D)	<ul style="list-style-type: none"> • Default integer rounding mode changed from Floor to Simplest 	<ul style="list-style-type: none"> • Block renamed as n-D Lookup Table • Icon changed

When you load models from earlier versions of Simulink that contain the Lookup Table, Lookup Table (2-D), and Lookup Table (n-D) blocks, those versions of the blocks appear. In R2011a, the new versions of the lookup table blocks appear only when you drag the blocks from the Simulink Library Browser into new models.

When you use the `add_block` function to programmatically add the Lookup Table, Lookup Table (2-D), or Lookup Table (n-D) blocks to a model, those versions of the blocks appear. If you want to add the *new* versions of the blocks to your model, change the source block path for `add_block` as follows:

Block	Old Block Path	New Block Path
Lookup Table	simulink/Lookup Tables/Lookup Table	simulink/Lookup Tables/1-D Lookup Table
Lookup Table (2-D)	simulink/Lookup Tables/Lookup Table (2-D)	simulink/Lookup Tables/2-D Lookup Table
Lookup Table (n-D)	simulink/Lookup Tables/Lookup Table (n-D)	simulink/Lookup Tables/n-D Lookup Table

To upgrade your model to use new versions of the Lookup Table and Lookup Table (2-D) blocks, follow these steps:

Step	Description	Reason
1	Run the Simulink Model Advisor check for Check model, local libraries, and referenced models for known upgrade issues requiring compile time information.	Identify blocks that do not have compatible settings with the new 1-D Lookup Table and 2-D Lookup Table blocks.
2	For each block that does not have compatible settings: <ul style="list-style-type: none"> Decide how to address each warning. Adjust block parameters as needed. 	Modify each Lookup Table or Lookup Table (2-D) block to make them compatible with the new versions.
3	Repeat steps 1 and 2 until you are satisfied with the results of the Model Advisor check.	Ensure that block replacement works for the entire model.
4	Run the <code>supdate</code> function on your model.	Perform block replacement with the 1-D Lookup Table and 2-D Lookup Table blocks.

Note that after block replacement, the block names that appear in the model remain the same. However, the block icons match the new ones for the 1-D Lookup Table and 2-D Lookup Table blocks.

Version History

The Model Advisor check groups all Lookup Table and Lookup Table (2-D) blocks into three categories:

- Blocks that have compatible settings with the new 1-D Lookup Table and 2-D Lookup Table blocks
- Blocks that have incompatible settings with the new 1-D Lookup Table and 2-D Lookup Table blocks

- Blocks that have repeated breakpoints

Blocks with Compatible Settings

When a block has compatible parameter settings with the new block, automatic block replacement can occur without backward incompatibilities.

Lookup Method in the Lookup Table or Lookup Table (2-D) Block	Parameter Settings in the New Block After Automatic Block Replacement	
	Interpolation	Extrapolation
Interpolation-Extrapolation	Linear	Linear
Interpolation-Use End Values	Linear	Clip
Use Input Below	Flat	Not applicable

Depending on breakpoint characteristics, the new block uses one of two index search methods.

Breakpoint Characteristics in the Lookup Table or Lookup Table (2-D) Block	Index Search Method in the New Block After Automatic Block Replacement
Not evenly spaced	Binary search
Evenly spaced and tunable	A prompt appears, asking you to select Binary search or Evenly spaced points.
Evenly spaced and not tunable	

The new block also adopts other parameter settings from the Lookup Table or Lookup Table (2-D) block. For parameters that exist only in the new block, the following default settings apply after block replacement:

Parameter in the New Block	Default Setting After Block Replacement
Breakpoint data type	Inherit: Same as corresponding input
Diagnostic for out-of-range input	None

Blocks with Incompatible Settings

When a block has incompatible parameter settings with the new block, the Model Advisor shows a warning and a recommended action, if applicable.

- If you perform the recommended action, you can avoid incompatibility during block replacement.
- If you use automatic block replacement without performing the recommended action, you might see numerical differences in your results.

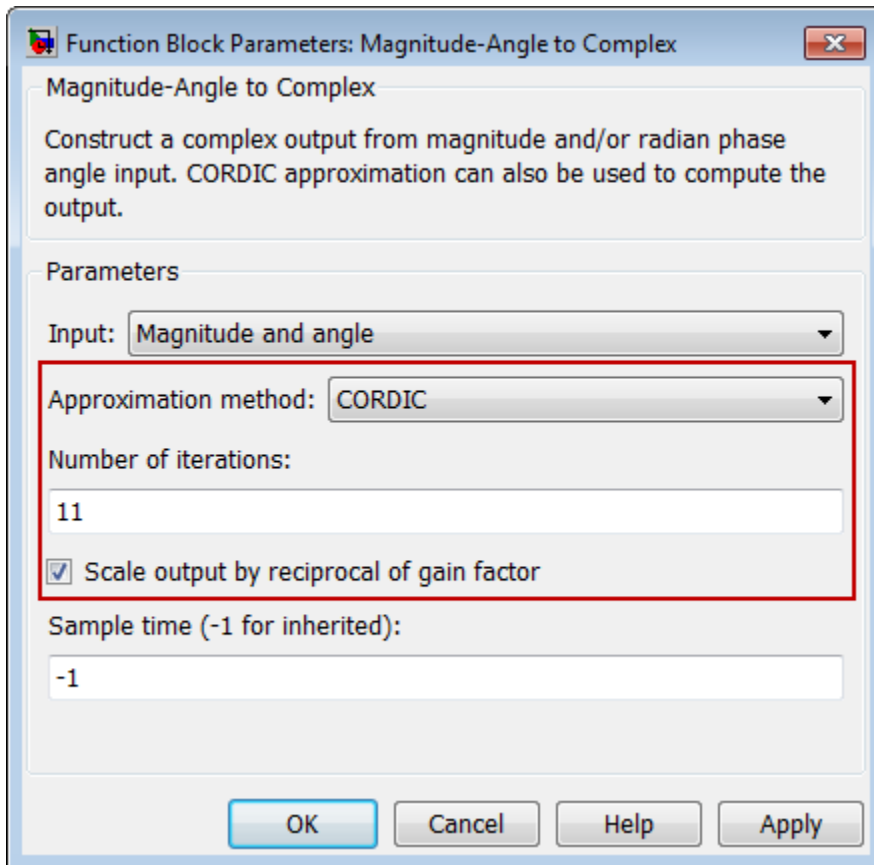
Incompatibility Warning	Recommended Action	What Happens for Automatic Block Replacement
The Lookup Method is Use Input Nearest or Use Input Above. The new block does not support these lookup methods.	Change the lookup method to one of the following: <ul style="list-style-type: none"> • Interpolation - Extrapolation • Interpolation - Use End Values • Use Input Below 	The Lookup Method changes to Interpolation - Use End Values. In the new block, this setting corresponds to: <ul style="list-style-type: none"> • Interpolation set to Linear • Extrapolation set to Clip
The Lookup Method is Interpolation - Extrapolation, but the input and output are not the same floating-point type. The new block supports linear extrapolation only when all inputs and outputs are the same floating-point type.	Change the extrapolation method or the port data types of the block.	You also see a message that explains possible numerical differences.
The block uses small fixed-point word lengths, so that interpolation uses only one rounding operation. The new block uses two rounding operations for interpolation.	None	You see a message that explains possible numerical differences.

Blocks with Repeated Breakpoints

When a block has repeated breakpoints, the Model Advisor recommends that you change the breakpoint data and rerun the check. You cannot perform automatic block replacement for blocks with repeated breakpoints.

Magnitude-Angle to Complex Block Supports CORDIC Algorithm and Fixed-Point Data Types

The Magnitude-Angle to Complex block now supports the following parameters:



The benefits of the new block parameters are as follows:

New Block Parameter	Purpose	Benefit
Approximation method	Specify the type of approximation the block uses to compute output: None or CORDIC.	Enables you to use a faster method of computing block output for fixed-point and HDL applications.
Number of iterations	For the CORDIC algorithm, specify how many iterations to use for computing block output.	Enables you to adjust the precision of your block output.
Scale output by reciprocal of gain factor	For the CORDIC algorithm, specify whether or not to scale the real and imaginary parts of the complex output.	Provides a more accurate numerical result for the CORDIC approximation.

This block now accepts and outputs fixed-point signals when you set **Approximation method** to CORDIC.

Trigonometric Function Block Supports Complex Exponential Output

The Trigonometric Function block now supports complex exponential output: $\cos + j\sin$. This function works with the CORDIC algorithm.

This block also accepts inputs with unsigned fixed-point data types when you use the CORDIC approximation. In previous releases, only signed fixed-point inputs were supported.

Shift Arithmetic Block Supports Specification of Bit Shift Values as Input Signal

The Shift Arithmetic block now supports specification of bit shift values from an input port. Previously, you could specify bit shift values only on the dialog box. This enhancement enables you to change bit shift values without stopping a simulation.

The block now also supports the following functionality:

Enhancement	Benefit
Specification of diagnostic for out-of-range bit shift values	Flags out-of-range bit shift values during simulation
Option to check for out-of-range bit shift values in the generated code	Enables you to control the efficiency of the generated code

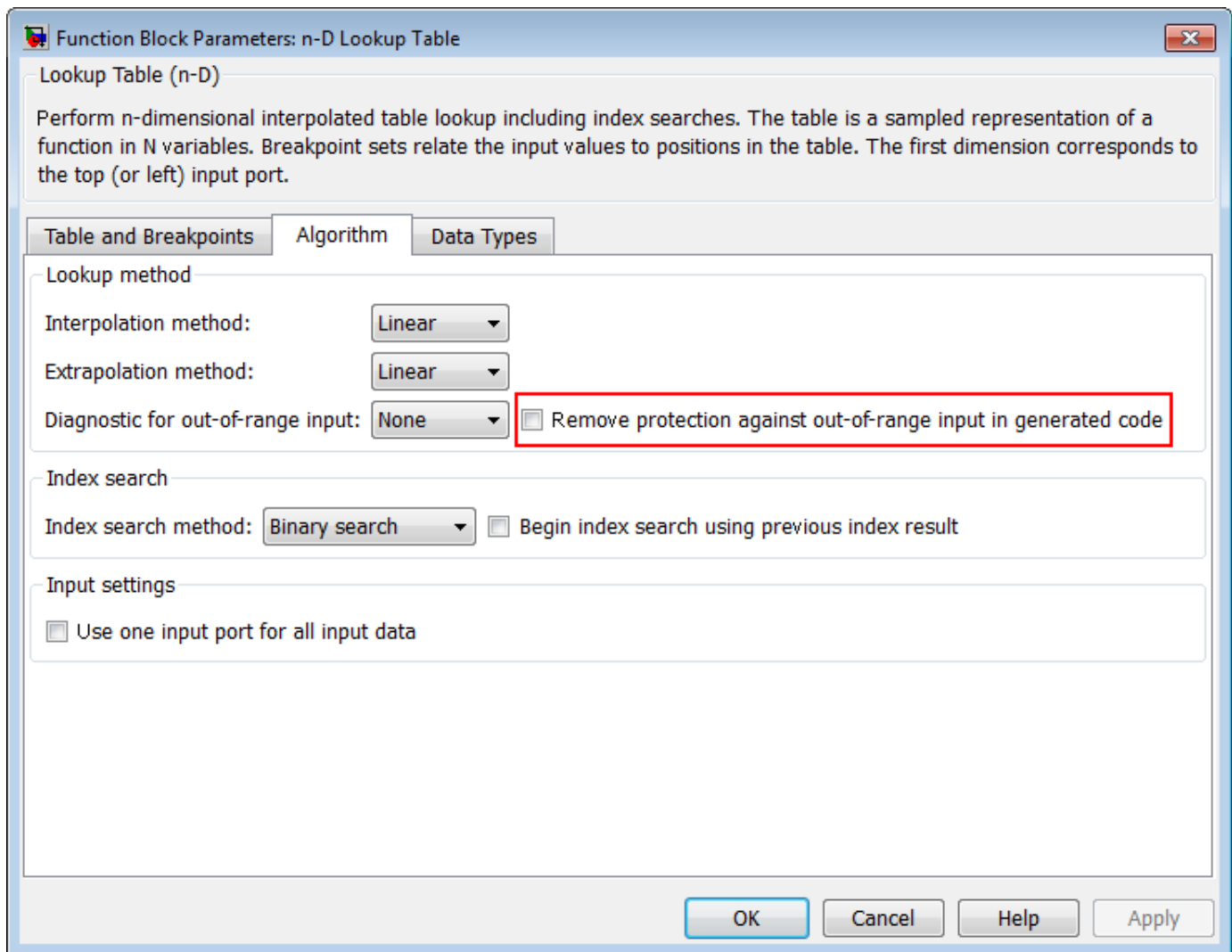
The following parameter changes apply to the Shift Arithmetic block. For backward compatibility, the old command-line parameters continue to work.

Old Prompt on Block Dialog Box	New Prompt on Block Dialog Box	Old Command-Line Parameter	New Command-Line Parameter
Number of bits to shift right	Bits to shift: Number	nBitShiftRight	BitShiftNumber
Number of places by which binary point shifts right	Binary points to shift: Number	nBinPtShiftRight	BinPtShiftNumber

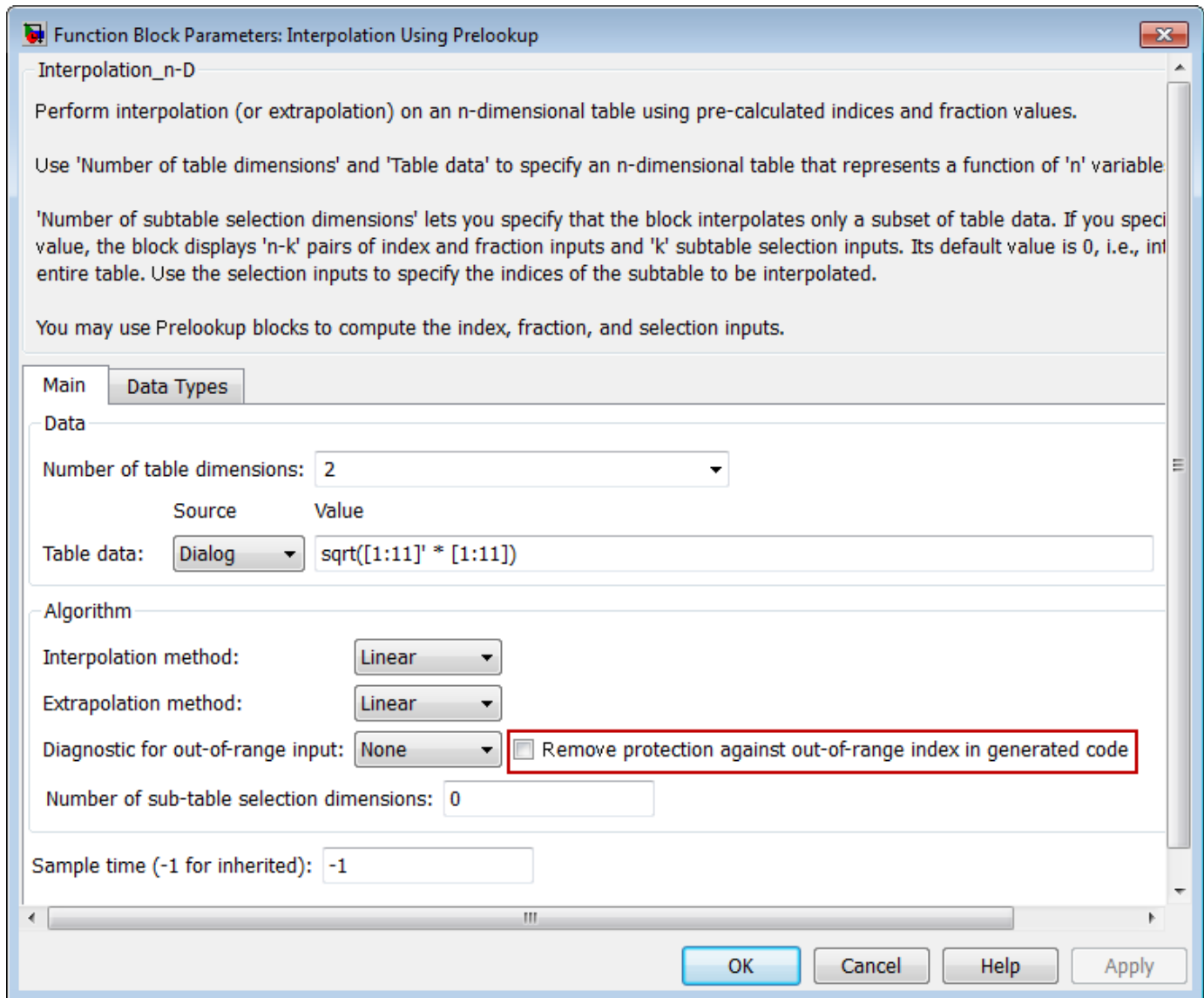
The read-only **BlockType** property has also changed from `SubSystem` to `ArithShift`.

Multiple Lookup Table Blocks Enable Removal of Range-Checking Code

When the breakpoint input to a Prelookup, 1-D Lookup Table, 2-D Lookup Table, or n-D Lookup Table block falls within the range of valid breakpoint values, you can disable range checking in the generated code. By selecting **Remove protection against out-of-range input in generated code** on the block dialog box, your code can be more efficient.



Similarly, when the index input to an Interpolation Using Prelookup block falls within the range of valid index values, you can disable range checking in the generated code. By selecting **Remove protection against out-of-range index in generated code** on the block dialog box, your code can be more efficient.



The **Remove protection against out-of-range index in generated code** check box replaces the **Check index in generated code** check box from previous releases. When you load models with the Interpolation Using Prelookup block from previous releases, the following parameter mapping applies:

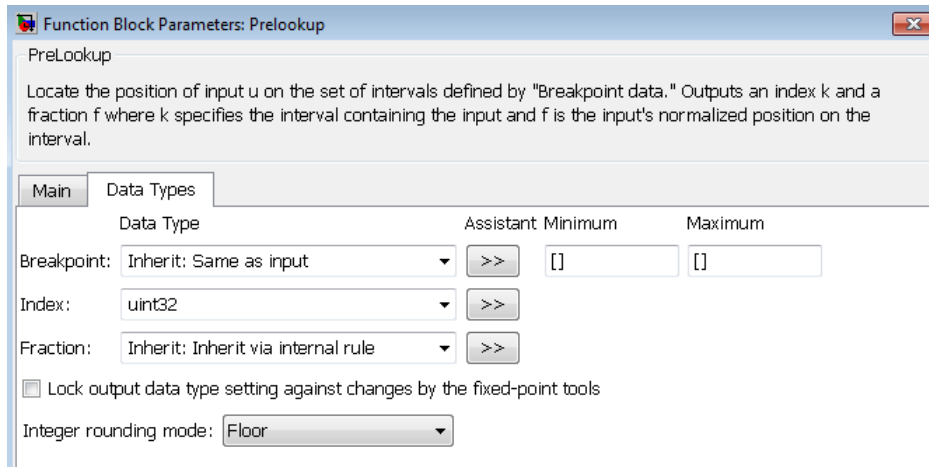
Parameter Setting from Previous Releases	Parameter Setting for R2011a
Check index in generated code is selected.	Remove protection against out-of-range index in generated code is not selected.
Check index in generated code is not selected.	Remove protection against out-of-range index in generated code is selected.

For backward compatibility, the command-line parameter `CheckIndexInCode` continues to work.

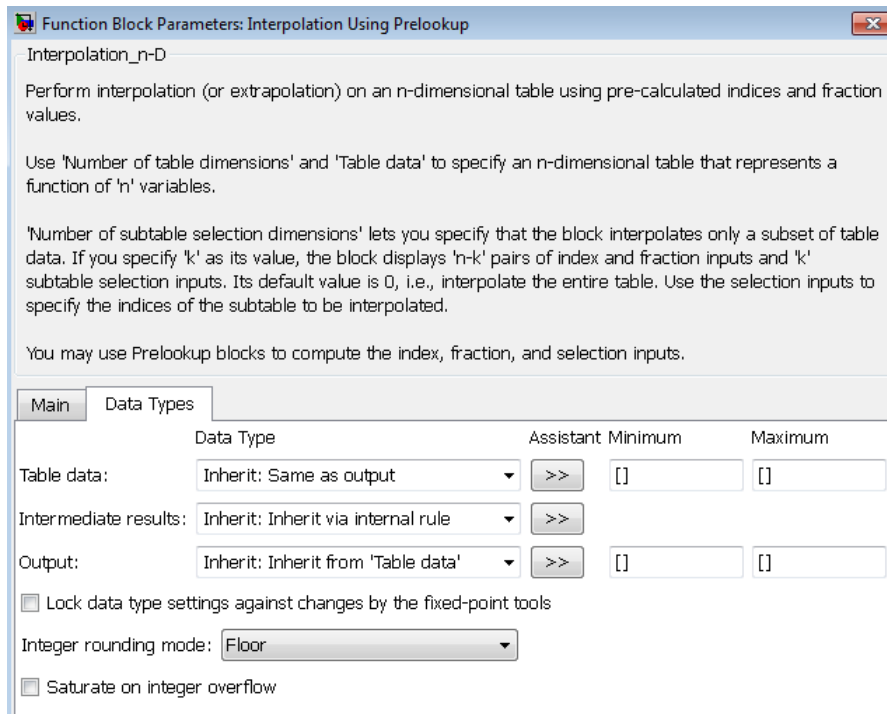
Enhanced Dialog Layout for the Prelookup and Interpolation Using Prelookup Blocks

In R2011a, the dialog boxes for the Prelookup and Interpolation Using Prelookup blocks consolidate parameters related to data type attributes on a single tab named **Data Types**. This enhancement enables you to specify data type attributes more quickly on the block dialog box.

- For the Prelookup block, you can now specify breakpoint, index, and fraction attributes on a single tab.



- For the Interpolation Using Prelookup block, you can now specify table, intermediate results, and output attributes on a single tab.



Product of Elements Block Uses a Single Algorithm for Element-Wise Complex Division

In previous releases, the Product of Elements block used two different algorithms for handling element-wise complex division. For example, for a matrix input with four elements (u_1 , u_2 , u_3 , and u_4), the following behavior would apply:

- For inputs with built-in integer and floating-point data types, the order of operations was $1/(u_1 * u_2 * u_3 * u_4)$.
- For inputs with fixed-point data types, the order of operations was $((((1/u_1)/u_2)/u_3)/u_4)$.

Starting in R2011a, the Product of Elements block uses a single algorithm for handling element-wise complex division. For inputs of integer, floating-point, or fixed-point type, the order of operations is always $((((1/u_1)/u_2)/u_3)/u_4) \dots /u_N$.

Sign Block Supports Complex Floating-Point Inputs

The Sign block now supports complex inputs of type `double` or `single`. The block output matches the MATLAB result for complex floating-point inputs.

When the input u is a complex scalar, the block output is:

$$\text{sign}(u) = u ./ \text{abs}(u)$$

When an element of a vector or matrix input is complex, the block uses the same formula that applies to scalar input.

MATLAB Fcn Block Renamed to Interpreted MATLAB Function Block

In R2011a, the MATLAB Fcn block has been renamed to Interpreted MATLAB Function block. The icon has also changed to match the new block name. However, all functionality and block parameters remain the same. The read-only **BlockType** property is also unchanged.

Existing scripts that use the `add_block` function to programmatically add the MATLAB Fcn block to models do not require any changes.

When you load existing models that contain the MATLAB Fcn block, the block name that appears in the model remains unchanged. However, the block icon matches the new one for the Interpreted MATLAB Function block.

Environment Controller Block Port Renamed from RTW to Coder

In R2011a, the Environment Controller block has renamed the RTW port to Coder. This enhancement better reflects the purpose of that input port, which designates signals to pass through the block when code generation occurs for a model.

Block Parameters on the State Attributes Tab Renamed

In R2011a, the block parameters **Real-Time Workshop storage class** and **Real-Time Workshop storage type qualifier** have been renamed to **Code generation storage class** and **Code**

generation storage type qualifier, respectively. These two parameters appear on the State Attributes tab of the following block dialog boxes:

- Discrete Filter
- Discrete PID Controller
- Discrete PID Controller (2DOF)
- Discrete State-Space
- Discrete Transfer Fcn
- Discrete Zero-Pole
- Discrete-Time Integrator
- Memory
- Unit Delay

Block Parameters and Values Renamed for Lookup Table Blocks

In R2011a, the **Action for out-of-range input** parameter has been renamed as **Diagnostic for out-of-range input** for the following blocks:

- Direct Lookup Table (n-D)
- Interpolation Using Prelookup
- n-D Lookup Table
- Prelookup

Also, the **Process out-of-range input** parameter has been renamed as **Extrapolation method** for the Prelookup block.

For lookup table blocks that provide **Interpolation method** or **Extrapolation method** parameters, the following changes apply:

Parameter Value from Previous Releases	Parameter Value in R2011a
None - Flat	Flat
None - Clip	Clip

Performance Improvement for Single-Precision Computations of Elementary Math Operations

In R2011a, single-precision computations for elementary math operations are faster. This enhancement applies to the following simulation modes:

- Normal
- Accelerator

Dead Zone Block Expands the Region of Zero Output

In R2011a, the Dead Zone block expands the region of zero output, or the dead zone, to include inputs (U) that equal the lower limit (LL) or upper limit (UL):

Input	Output
$U \geq LL$ and $U \leq UL$	Zero
$U > UL$	$U - UL$
$U < LL$	$U - LL$

In previous releases, the dead zone excluded inputs that equal the lower or upper limit.

Enhanced PID Controller Blocks Display Compensator Formula in Block Dialog Box

The PID Controller and PID Controller (2 DOF) blocks now display the current compensator formula in the block dialog box. This display reflects the current settings for controller type, controller form, and time domain.

Ground Block Always Has Constant Sample Time

In R2011a, the sample time of the Ground block is now constant (`inf`) regardless of the setting for **Inline parameters** in the Configuration Parameters dialog box.

Version History

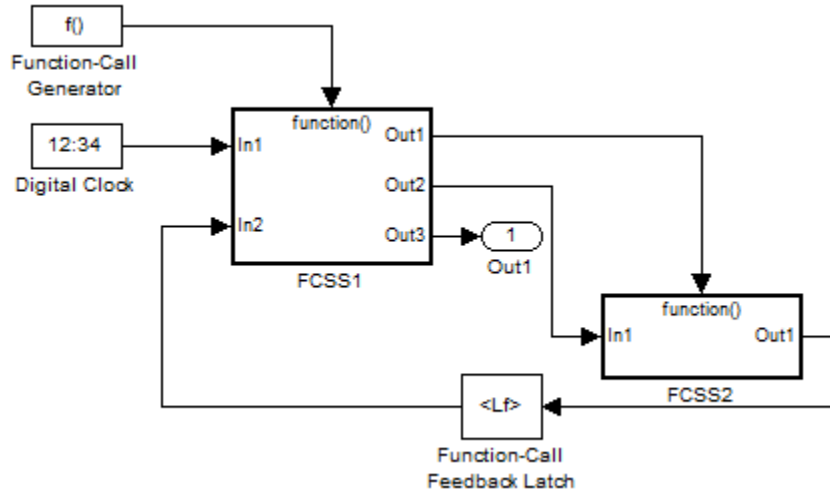
Previously, if **Inline parameters** was `off`, the sample time of the Ground block depended on sample-time propagation. Now, the following conditions hold true:

- Function-call subsystem blocks that have an unconnected function-call port now have the correct sample time of constant (`inf`) regardless of the setting for **Inline parameters**.
- Function-call subsystem blocks that have a function-call port connected to a Ground block now have the correct sample time of constant (`inf`) regardless of the setting for **Inline parameters**.
- Function-call subsystem blocks that have the **Sample time type** set to `periodic` now correctly error out when they are connected to a Ground block or unconnected.

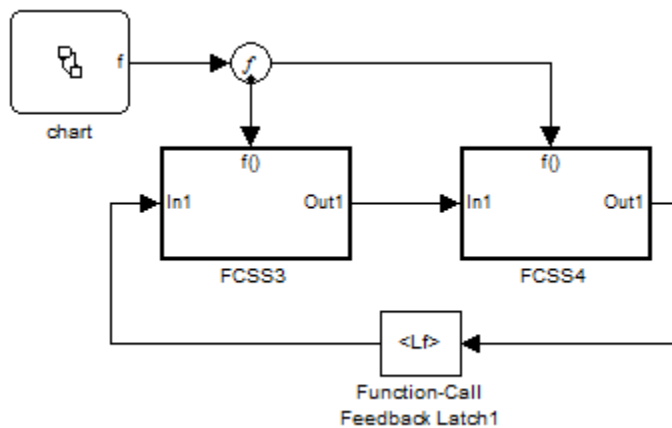
New Function-Call Feedback Latch Block

The Function-Call Feedback Latch block allows you to break a feedback loop involving data signals between function-call signals. You can use this block for two specific scenarios:

- If a loop involves parent and child function-call blocks (that is, the initiator of the child function-call block is inside the parent function-call block), then place this block on the feedback signal from the child to the parent. You can thus ensure that the value of the signal does not change during execution of the child.



- If a loop involves function-call blocks connected to branches of the same function-call signal, then this block latches the signal at the input of the destination function-call block, and thereby allows it to execute prior to the source function-call block.



In either case, the latching results in the destination block reading a delayed signal from the previous execution of the source function-call block.

Output Driving Merge Block Does Not Require IC in Simplified Initialization Mode

If an Output block of a conditionally executed subsystem directly drives a Merge block, then the Output block no longer requires the specification of an Initial Condition (IC) in simplified initialization mode. Simulink still expects the Merge block to specify an IC. This enhancement applies only when the Output and Merge blocks are in the same model.

Discrete Filter, Discrete FIR Filter, and Discrete Transfer Fcn Blocks Now Have Input Processing Parameter

The Discrete Filter, Discrete FIR Filter, and Discrete Transfer Fcn blocks now have an **Input processing** parameter. This parameter enables you to specify whether the block performs sample- or frame-based processing on the input. To perform frame-based processing, you must have a DSP System Toolbox license.

Model Blocks Can Now Use the GetSet Custom Storage Class

The GetSet custom storage class can now be used for the inports and outports of Model blocks. To assign a GetSet custom storage class to the inport or outport of a referenced model block, use one of the following methods.

- 1** Assign the GetSet custom storage class to the root-level inport or outport of the referenced model.
- 2** Assign the GetSet custom storage class to scalar signals entering an inport of the referenced model block in the parent model, provided one of the following conditions is met.
 - a** The referenced model uses function prototype control to specify that the inport should be passed by value instead of being passed by pointer to the Model block's step function.
 - b** The inport to which the GetSet custom storage class is assigned should be passed by value.
- 3** Assign the GetSet custom storage class to a scalar signal leaving one of the outports of the referenced model block in the parent model. In this case, the referenced model must use function prototype control to specify that the outport should be the returned value of the function.

User Interface Enhancements

Model Explorer: Hiding the Group Column

By default, the property column that you use for grouping (the group column) appears in the property table. That property also appears in the top row for each group.

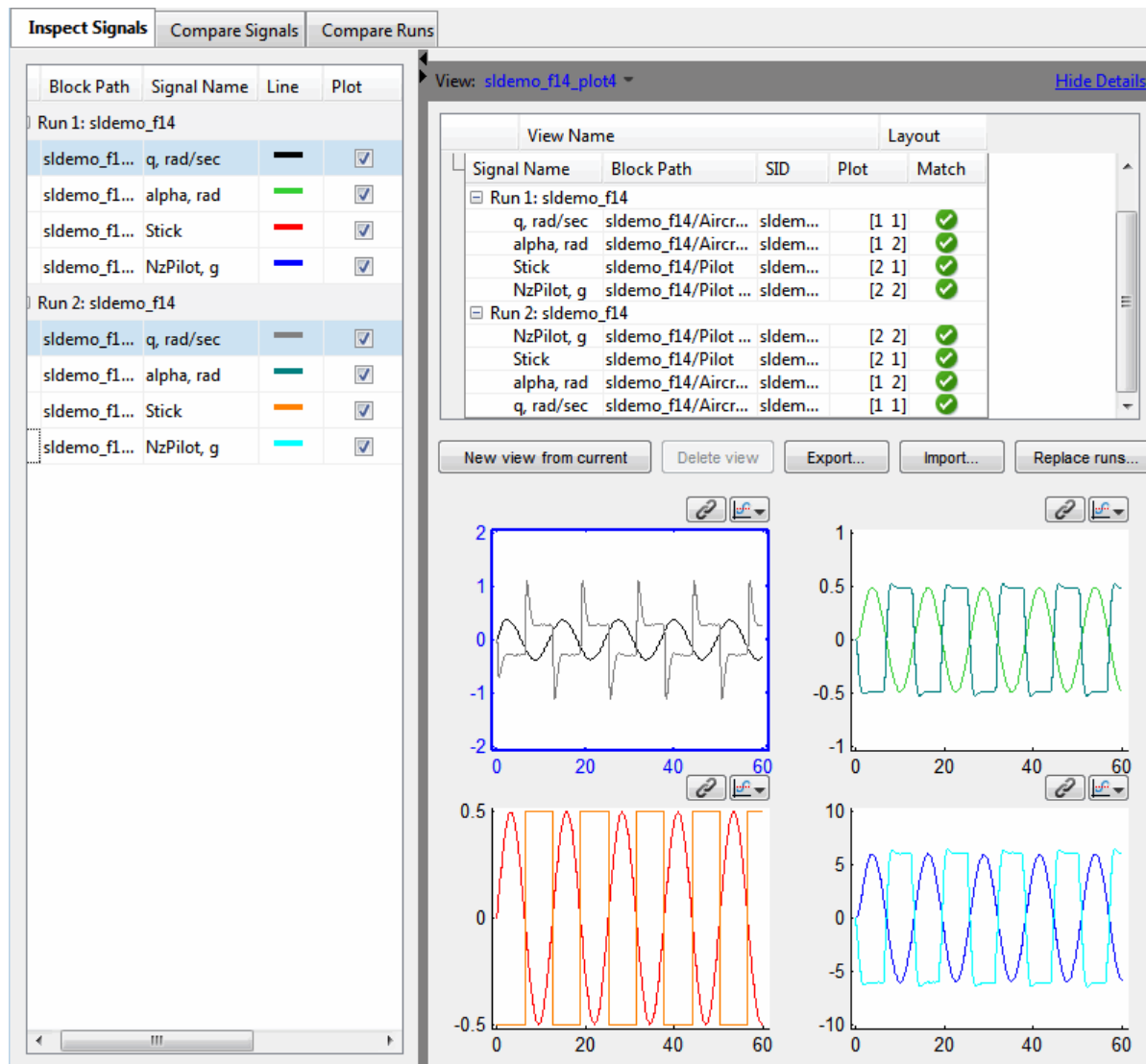
To hide the group column, use one of the following approaches:

- From the **View** menu, clear the **Show Group Column** check box.
- Within the property table, right-click a column heading and clear the **Show Group Column** check box.

Simulation Data Inspector Enhancements

Multiple Plots in a View

The Simulation Data Inspector tool now supports the configuration of multiple plots into one *view*. On the **Inspect Signals** pane, on the View toolbar, select **Show Details** to display the View Details table.



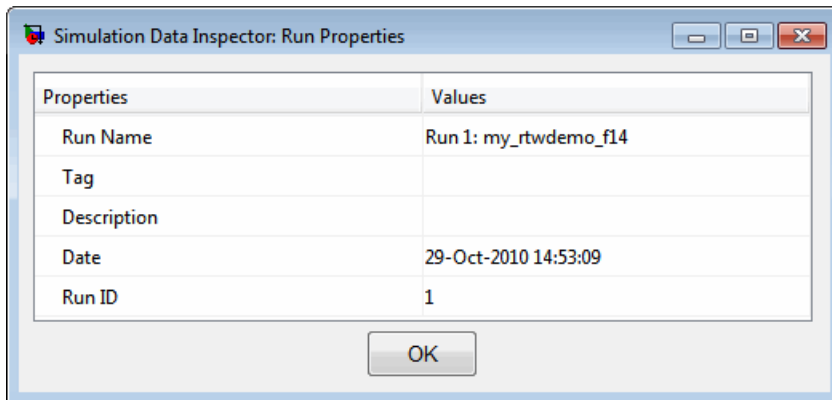
You can create multiple views by clicking the **New view from current** button. In each view, you can:

- Modify the number of plots by clicking the **Layout** column to display the plot matrix.
- Name, save, and reload the view using the corresponding buttons.
- Replace signal data for a run with the corresponding signal data of another run by clicking the **Replace runs** button.



For more information, see Visual Inspection of Signal Data in the Simulation Data Inspector Tool.

Display Run Properties

In R2011a, you can view the properties of a run. In the Signal Browser table, right-click a run name to view a list of options. To open the Run Properties dialog box, from the options list, select **Properties**.



New Toolbar Icons

The Simulation Data Inspector toolbar includes a new icon  for zooming out a section of a plot. The previous zoom out icon  now performs a fit to view operation, which enlarges a plot to fill the graph. To perform either operation, select the icon, and click on a plot.

Model Advisor

In R2011a, the Model Advisor tool now includes easier control of the **By Product** and **By Task** folders. In the Model Advisor, select **View > Show By Product Folder** or **Show By Task Folder** to show or hide each folder. These settings are persistent across MATLAB sessions.

In the **By Task** folder, there are two new subfolders:

- **Modeling and Simulation**
- **Code Generation Efficiency**

For more information on the Model Advisor GUI, see Consulting the Model Advisor.

Configuration Parameters Dialog Box Changes

The Configuration Parameters dialog box layout is improved to better support your workflows. The **Optimization** pane is reorganized into three panes:

- **General**
- **Signals and Parameters**
- **Stateflow**

These panes make it easier to find parameters.

In R2011a, all tree nodes are collapsed by default. For details, see Configuration Parameters Dialog Box.

S-Functions

S-Functions Generated with `legacy_code` function and `singleCPPMexFile` S-Function Option Must Be Regenerated

Due to an infrastructure change, if you have generated an S-function with a call to `legacy_code` that defines the S-function option `singleCPPMexFile`, you must regenerate the S-function to use it with this release of Simulink.

For more information, see the description of `legacy_code` and Integrating Existing C Functions into Simulink Models with the Legacy Code Tool.

Version History

If you have generated an S-function with a call to `legacy_code` that defines the S-function option `singleCPPMexFile`, regenerate the S-function to use it with this release of Simulink.

R2010bSP2

Version: 7.6.2

Bug Fixes

R2010bSP1

Version: 7.6.1

Bug Fixes

R2010b

Version: 7.6

New Features

Bug Fixes

Version History

Simulation Performance

Elimination of Regenerating Code for Rebuilds

For models that contain Model Reference blocks and that have not changed between Rapid Acceleration simulations, the rebuild process is more efficient.

Previously, if an Accelerator simulation or a Code Generation ERT/GRT was performed between two Rapid Acceleration simulations, then Simulink partially built the code a second time during the second Rapid Acceleration simulation.

Now, providing the model checksum remains constant, Simulink does not generate code for the second Rapid Accelerator simulation.

Component-Based Modeling

Model Workspace Is Read-Only During Compilation

During the compilation of a model, Simulink enforces that the model workspace is read-only, by issuing an error if there is an attempt to change a model workspace variable during compilation. This enforcement of a read-only workspace prevents the simulation from failing or producing incorrect results due to changes to the model workspace.

Version History

In previous releases, you could change model workspace variables when compiling a model (for example, this could occur when compiling referenced models). Rewrite any code that changes model workspace variables during compilation of a model.

Support for Multiple Normal Mode Instances of a Referenced Model

You can use Normal mode for multiple instances of a referenced model. Prior to R2010b, a model with model references could use Normal mode for at most one instance of each referenced model.

A referenced model must be in Normal mode for you to be able to use several important Simulink and Stateflow features, including linearization and model coverage analysis. Using Normal mode also can make editing and testing models more efficient.

In the `sldemo_mdhref_depgraph` demo, see the “Interacting with the Dependency Viewer in Instance View” section for an example of how to use multiple Normal mode instances of a referenced model. For additional information about using multiple Normal mode instances of a referenced model, see [Using Normal Mode for Multiple Instances of Referenced Models](#).

Version History

The **Save As** feature preserves the **Simulation mode** setting of the Model block as much as possible.

If the both of the following conditions are true, then the saved model does not simulate:

- The R2010b model has multiple Normal mode instances of a referenced model.
- You use the **Save As** feature to save the model to a release earlier than R2010b that supports model reference Normal mode.

In this situation, the saved model does not simulate because only one instance of a referenced model could be in Normal mode in that earlier release.

Also, in releases before R2010b, you could select the **Refresh Model Blocks** menu item directly from the **Edit** menu in the Model Editor. In R2010b, access the **Refresh Model Blocks** menu item from the **Edit > Model Blocks** menu item.

New Variant Subsystem Block for Managing Subsystem Design Alternatives

A Variant Subsystem block provides multiple implementations for a subsystem where only one implementation is active during simulation. You can programmatically swap implementations without

modifying the model. When the model is compiled, the Simulink engine chooses the active subsystem from a selection of subsystems. The active subsystem is determined by the values of the variant control variables and variant objects, which you define in the base workspace. By modifying the values of the variant control variables, you can easily specify which subsystem runs in your simulation.

For more information, see [Setting Up Variant Subsystems](#). If you use the Model Advisor to check a system containing a variant subsystem, see [Model Advisor Limitations](#), for more information.

Support for Bus and Enumerated Data Types on Masks

For the Masked Parameters dialog box, you can now create data type parameters that support the specification of bus or enumerated (enum) data types.

To create a data type parameter that supports bus data types, in the Mask Editor, select the **Parameters** pane.

For information about how to specify bus and enumerated data type parameters, see [Data Type Control](#).

sl_convert_to_model_reference Function Removed

The `sl_convert_to_model_reference` function is obsolete and has been removed from the Simulink software.

To convert an atomic subsystem to a model reference, right-click the atomic subsystem and select the **Convert to Model Block** menu item, or use the `Simulink.SubSystem.convertToModelReference` function. See [Atomic Subsystem and Converting a Subsystem to a Referenced Model](#) for more information.

Verbose Accelerator Builds Parameter Applies to Model Reference SIM Target Builds in All Cases

For referenced models, the **Configuration Parameter > Optimization > Verbose accelerator build** parameter is no longer overridden by the **Configuration Parameter > Real-Time Workshop > Debug > Verbose build** parameter setting when building model reference SIM targets.

Embedded MATLAB Function Blocks

Specialization of Embedded MATLAB Function Blocks in Simulink Libraries

You can now create library instances of the same Embedded MATLAB Function block with distinct properties, including:

- Data type, size, complexity, sampling mode, range, and initial value
- Block sample time
- Fixed-point data type override mode
- Resolution to different MATLAB files on the path

With this capability, you can create custom block libraries using Embedded MATLAB Function blocks. For more information, see [Creating Custom Block Libraries with MATLAB Function Blocks](#).

Support for Creation and Processing of Arrays of Buses

The Embedded MATLAB Function block now supports arrays of buses.

Ability to Include MATLAB Code as Comments in Generated C Code

You can now select to include MATLAB source code as comments in code generated for an Embedded MATLAB Function block. This capability improves traceability between generated code and the original source code.

Note This option requires a Real-Time Workshop® license.

For more information, see [MATLAB source code as comments in the Real-Time Workshop documentation](#).

Data Properties Dialog Box Enhancements

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

Parameters	Location in R2010a	Location in R2010b	Benefit of Location Change
Limit range <ul style="list-style-type: none"> • Minimum • Maximum 	Value Attributes tab	General tab	Consistent with blocks in the Simulink library that specify these parameters on the same tab as the data type.
Save final value to base workspace	Value Attributes tab	Description tab	Consolidates parameters related to the data description.

Parameter Being Removed in Future Release

The **Save final value to base workspace** will be removed in a future release.

Simulink Data Management

Enhanced Support for Bus Objects as Data Types

The following blocks have added support for specifying a bus object as a data type:

- Constant
- Signal Specification

For the Constant block, if you use a bus object as a data type, you can set the **Constant value** to be one of these values:

- A full MATLAB structure corresponding to the bus object
- 0 to indicate a structure corresponding to the ground value of the bus object

See the Constant block reference page for an example that shows how to use a structure to simplify a model.

The following blocks and Simulink classes now use a consistent **Data type** parameter option, **Bus : <object name>**, for specifying a bus object as a data type:

- Constant block
- Bus Creator block
- Inport block
- Outport block
- Signal Specification block
- Simulink.BusElement class
- Simulink.Parameter class
- Simulink.Signal class

Version History

The interface for specifying a bus object as a data type is now consistent for the blocks that support that capability. Making the interface consistent involves removing some block parameters that existed in releases prior to R2010b. The following table summarizes the changes.

Block	Removed Pre-R2010b Parameters	Replacement R2010b Parameter
Bus Creator	Bus object Specify properties via bus object	Output data type
Inport	Specify properties via bus object Bus object for validating input bus	Data type

Block	Removed Pre-R2010b Parameters	Replacement R2010b Parameter
Outport	<p>Specify properties via bus object</p> <p>Bus object for validating input bus</p>	Data type

Enhancements to Simulink.NumericType Class

The Simulink.NumericType class now has the following methods:

- isboolean
- isdouble
- isfixed
- isfloat
- isscalingbinarypoint
- isscalingslopebias
- isscalingunspecified
- issingle

Importing Signal Data Sets into the Signal Builder Block

The Signal Builder block can now accept existing signal data sets. In previous releases, you had to enter existing signal data one by one in the Signal Builder dialog box or with the `signalbuilder` function.

In the Signal Builder block dialog box, you can now use the **File > Import from File** to import files that contain data sets. These data sets can contain test data that you have collected, or you can manually create these files. The block accepts the appropriately formatted file types:

- Excel (.xls, .xlsx)
- Comma-separated value (CSV) text files (.csv)
- MAT-files (.mat)

For further information, see Working with Signal Groups in the Simulink User's Guide.

signalbuilder Function Changes

The `signalbuilder` function has improved functionality:

To...	Use...
Add new groups to the Signal Builder block.	'append'
Append signals to existing signal groups in the Signal Builder block.	'appendsignal'

To...	Use...
Make visible signals that are hidden in the Signal Builder block.	'showsignal'
Make invisible signals that are visible in the Signal Builder block.	'hidesignal'

From File Block Enhancements

The From File block includes the following new features:

- You can specify the method that the From File block uses to handle situations where there is not an exact match between a Simulink sample time hit and a time in the data file that the From File block reads.
 - In previous releases, the From File block automatically applied a linear interpolation and extrapolation method.
 - In R2010b, you can set the interpolation method independently for each of these situations:
 - Data extrapolation before the first data point
 - Data interpolation within the data time range
 - Data extrapolation after the last data point
 - The choices for the interpolation methods are (as applicable):
 - Linear interpolation
 - Zero-order hold
 - Ground value
- The From File block now can read signal data that has an enumerated (enum) data type, in addition to previously supported data types.

Finding Variables Used by a Model or Block

You can get a list of variables that a model or block uses.

In the Simulink Editor, right-click a block, subsystem, or the canvas and select the **Find Referenced Variables** menu item.

Simulink returns the results in the Model Explorer.

As an alternative, you can use the Model Explorer interface directly to find variables used by a model or block, as described in Finding Variables That Are Used by a Model or Block.

enumeration Function Replaced With MATLAB Equivalent

Starting with R2010b, when you invoke the enumeration function, you will be invoking a MATLAB equivalent of the Simulink function with the same name available in earlier releases.

See the description of the new MATLAB enumeration function introduced with new support for enumeration classes.

Programmatic Creation of Enumerations

The new `Simulink.defineIntEnumType` function provides a way to programmatically import enumerations defined externally—for example, in a data dictionary—to MATLAB. The function creates and saves an enumeration class definition file on the MATLAB path.

For more information, see the description of `Simulink.defineIntEnumType` and Enumerations and Modeling.

Simulink.Signal and Simulink.Parameter Objects Now Obey Model Data Type Override Settings

`Simulink.Signal` and `Simulink.Parameter` objects now honor model-level data type override settings. This capability allows you to share fixed-point models that use `Simulink.Signal` or `Simulink.Parameter` objects with users who do not have a Simulink Fixed Point™ license.

To simulate a model without using Simulink Fixed Point, use the Fixed-Point Tool to set the model-level **Data type override** setting to `Double` or `Single` and the **Data type override applies to** parameter to `All numeric types`. If you use `fi` objects or embedded numeric data types in your model, set the `fipref.DataTypeOverride` property to `TrueDoubles` or `TrueSingles` and the `DataTypeOverrideAppliesTo` property to `All numeric types` to match the model-level settings. For more information, see `fxptdlg`.

Simulink File Management

Autosave Upgrade Backup

New Autosave option to backup Simulink models when upgrading to a newer release. Automatically saving a backup copy can be useful for recovering the original file in case of accidental overwriting with a newer release.

You can set this Autosave option in the Simulink Preferences Window. See Autosave.

Model Dependencies Tools

Enhanced file dependency analysis has the following new features:

- Find workspace variables that are required by your design but not defined by a file in the manifest
- Store code analysis warnings in the manifest
- Validate manifests before exporting to a ZIP file, to check for missing files and data
- Compare manifests with ZIP files and folders

For details see Model Dependencies.

Simulink Signal Management

Arrays of Buses

You can now use arrays of buses to represent structured data compactly, eliminating the need to include multiple copies of the same buses. You can iteratively process each element of the bus array, for example, by using a For Each subsystem.

The following blocks now support arrays of buses:

- Virtual blocks (see Virtual Blocks)
- These bus-related blocks:
 - Bus Assignment
 - Bus Creator
 - Bus Selector
- These nonvirtual blocks:
 - Merge
 - Multipoint Switch
 - Rate Transition
 - Switch
 - Zero-Order Hold
- Assignment
- MATLAB Function (formally called Embedded MATLAB Function)
- Matrix Concatenate
- Selector
- Vector Concatenate
- Width
- Two-Way Connection (a Simscape block)

Create an array of buses with either a Vector Concatenate or Matrix Concatenate block. The input bus signals to these blocks must be nonvirtual and of the same type (that is, have the same names, hierarchies, and attributes for the leaf elements).

The generated code creates arrays of C structures that represent arrays of buses. You can use the Legacy Code Tool to integrate legacy C code that uses arrays of structures.

In an Embedded MATLAB[®] Function block, you can process arrays of bus signals using regular MATLAB syntax.

The use of arrays of buses does not support the following:

- Virtual buses
- Data loading or logging
- Stateflow action language

For details about using arrays of buses, see [Combining Buses into an Array of Buses](#).

Version History

If you specify a bus object as the data type for a root Inport or Outport block, the **Dimensions** parameter is enabled, to allow you to specify dimensions other than 1 or -1 for an array of buses.

In previous releases, the **Dimensions** parameter was ignored if you specified a bus object as the data type for a root Inport or Outport block. If you specified a dimension other than 1 or -1, then do one of the following, depending on whether you want to use an array of buses or you want to output as a virtual bus:

- To use an array of buses:
 - In the **Signal Attributes** pane of the block parameters dialog box for a root Inport or Outport block, select the **Output as nonvirtual bus** option.
 - In the **Configuration Parameters > Diagnostics > Connectivity>>** pane, set **Mux blocks used to create bus signals** to error.
- To output as a virtual bus, set the **Dimensions** parameter to 1 or -1.

Loading Bus Data to Root Input Ports

You can now use MATLAB structures and `timeseries` objects when defining root-level input port signals. Using a structure of `timeseries` objects for bus signals simplifies loading bus data to root input ports.

To specify the input, use the **Configuration Parameters > Data Import/Export > Input** parameter. For more information, see [Importing MATLAB timeseries Data and Importing Structures of MATLAB timeseries Objects for Bus Signals to a Root-Level Input Port](#).

Block Enhancements

Prelookup Block Supports Dynamic Breakpoint Data

The Prelookup block now supports specification of breakpoint data from an input port. Previously, you could specify breakpoint data only on the dialog box.

This enhancement enables you to change breakpoint data without stopping a simulation. For example, you can incorporate new breakpoint data if the physical system you are simulating changes.

Interpolation Using Prelookup Block Supports Dynamic Table Data

The Interpolation Using Prelookup block now supports specification of table data from an input port. Previously, you could specify table data only on the dialog box.

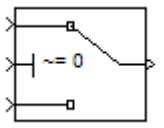
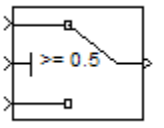
This enhancement enables you to change table data without stopping a simulation. For example, you can incorporate new table data if the physical system you are simulating changes.

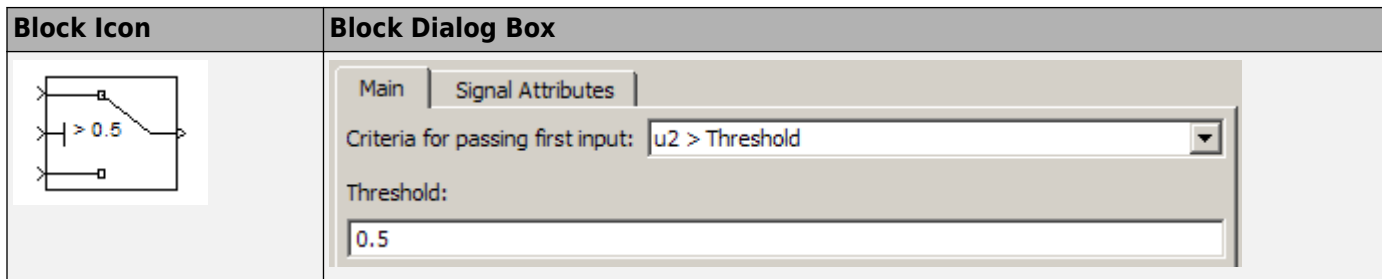
Multiport Switch Block Supports Specification of Default Case for Out-of-Range Control Input

When the control input of the Multiport Switch block does not match any data port indices, you can specify the last data port as the default or use an additional data port. This enhancement enables you to avoid simulation errors and undefined behavior in the generated code.

Switch Block Icon Shows Criteria and Threshold Values

This enhancement helps you identify the **Criteria for passing first input** and **Threshold** values without having to open the Switch block dialog box.

Block Icon	Block Dialog Box
	<p>Main Signal Attributes</p> <p>Criteria for passing first input: <input type="text" value="u2 ~= 0"/></p> <p>Threshold: <input type="text" value="0.5"/></p>
	<p>Main Signal Attributes</p> <p>Criteria for passing first input: <input type="text" value="u2 >= Threshold"/></p> <p>Threshold: <input type="text" value="0.5"/></p>



Trigonometric Function Block Supports Expanded Input Range for CORDIC Algorithm

The Trigonometric Function block now supports an input range of $[-2\pi, 2\pi)$ radians when you set **Function** to `sin`, `cos`, or `sincos` and set **Approximation method** to `CORDIC`. Previously, the input range allowed was $[0, 2\pi)$ radians.

This enhancement enables you to use a wider range of input values that are natural for problems that involve trigonometric calculations.

Repeating Sequence Stair Block Supports Enumerated Data Types

The Repeating Sequence Stair block now supports enumerated data types. For more information, see *Enumerations and Modeling in the Simulink User's Guide*.

Abs Block Supports Specification of Minimum Output Value

The Abs block now supports specification of an **Output minimum** parameter. This enhancement enables you to specify both minimum and maximum values for block output. In previous releases, you could specify the maximum output value but not the minimum, which Simulink assumed to be 0 by default.

Saturation Block Supports Logging of Minimum and Maximum Values for the Fixed-Point Tool

When you set **Fixed-point instrumentation mode** to `Minimums`, `maximums` and `overflows` in the Fixed-Point Tool, the Saturation block logs minimum and maximum values. In previous releases, this block did not support min/max logging.

Vector Concatenate Block Now Appears in the Commonly Used and Signal Routing Libraries

In the Simulink Library Browser, the Vector Concatenate block now appears in the Commonly Used and Signal Routing libraries. This block continues to appear in the Math Operations library.

Model Discretizer Support for Second-Order Integrator Block

You can now discretize a model containing a Second-Order Integrator block using the Model Discretizer. Based on your block parameter settings, the tool replaces the continuous Second-Order Integrator block with one of the four discrete subsystems in the z -domain.

Integer Delay and Unit Delay Blocks Now Have Input Processing Parameter

The Integer Delay and Unit Delay blocks now have an **Input processing** parameter. This parameter enables you to specify whether the block performs sample- or frame-based processing on the input. To perform frame-based processing, you must have a Signal Processing Blockset license.

Version History

Beginning this release, MathWorks is changing how our products control frame-based processing. Previously, signals themselves were sample or frame based. Our blocks inherited that information from the signal, and processed the input accordingly, either as individual samples or as frames of data. Beginning this release, signals are no longer responsible for carrying information about their frame status. The blocks themselves now control whether they perform sample- or frame-based processing on the input.

Some blocks can do only one type of processing and thus require no changes. Other blocks can do both sample- and frame-based processing and thus require a new parameter. The Integer Delay and Unit Delay blocks fall into the latter category.

If you have any Integer Delay or Unit Delay blocks in an R2010a or earlier model, those blocks will continue to produce the same results in R2010b. When you open an existing model with an Integer Delay or Unit Delay block in R2010b, the **Input processing** parameter of those blocks will be set to **Inherited**. Your models will continue to run in this mode, but it is recommended that you run the `slupdate` function to set the **Input processing** parameter to the equivalent non-inherited mode. The non-inherited modes are **Elements as channels (sample based)** and **Columns as channels (frame based)**.

If you do not run the `slupdate` function on your model before the **Inherited** option is removed, any **Input processing** parameter set to **Inherited** on an Integer Delay or Unit Delay block will be set automatically to **Elements as channels (sample based)**.

Data Store Read Block Sample Time Default Changed to -1

In R2010b, the Data Store Read block uses a default value of -1 for the **Sample time**, for consistency with the Data Store Write block and most other blocks. In previous releases, the default sample time was 0.

Version History

The **Sample time** default for the Data Store Read block has changed from 0 in previous releases to -1 in R2010b.

Support of Frame-Based Signals Being Removed From the Bias Block

Starting in R2010b, frame-based signal support is being removed from the Bias block. In a future release, the block will no longer support frame-based processing. To offset a frame-based signal in R2010b or later releases, you can use the Signal Processing Blockset Array-Vector Add block.

Version History

If you have any R2010a or earlier models that use the Bias block to offset a frame-based signal, you can use the `slookup` function to upgrade your model. For each instance where you use a Bias block with a frame-based input signal, the `slookup` function replaces the Bias block with an Array-Vector Add block.

Relaxation of Limitations for Function-Call Split Block

Two limitations of the Function-Call Split block have been relaxed for R2010b.

- Previously, the direct children of a branched function-call had to have periodic or asynchronous sample time. Now the direct children can also be triggered. Therefore, the branched function-call can trigger a Stateflow chart directly.
- Previously, if a branched function-call initiator was a Stateflow event, then the Stateflow function-call output event had to be bound to a particular state. Now the event can be bound or unbound to a state when invoking a branched function-call.

User Interface Enhancements

Model Explorer and Command-Line Support for Saving and Loading Configuration Sets

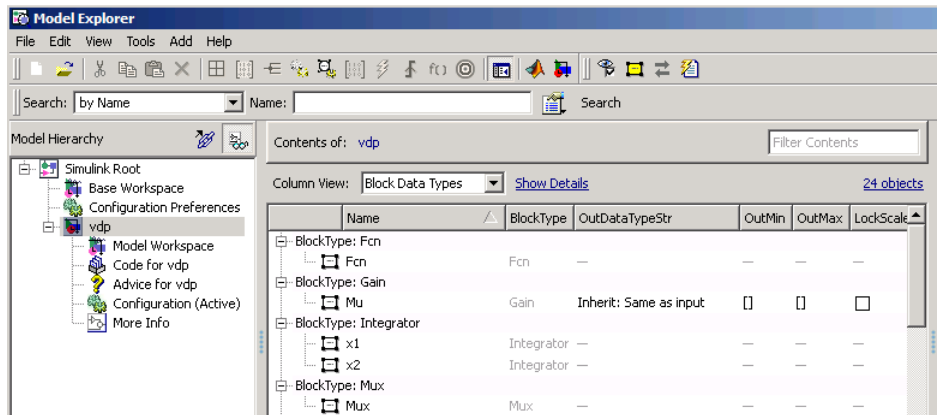
Previously, you could save and load a configuration set from the command line only, requiring many steps. Now you can save and load a configuration set using the Model Explorer. You can also save or load the active configuration set using one function, the `Simulink.BlockDiagram.saveActiveConfigSet` or `Simulink.BlockDiagram.loadActiveConfigSet` function.

For details, see the following sections in the *Simulink User's Guide*:

- Save a Configuration Set
- Load a Saved Configuration Set

Model Explorer: Grouping by a Property

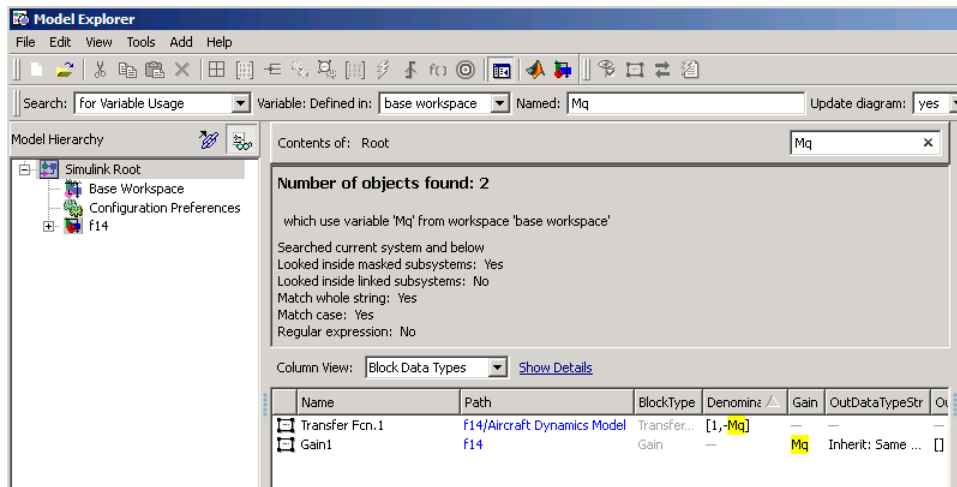
In the Contents pane, you can group data based on a property values. For example, you can group by the `BlockType` property by right-clicking that column heading and selecting the **Group by This Column** menu item. The result looks similar to this:



For details, see [Grouping by a Property](#).

Model Explorer: Filtering Contents

In the **Contents** pane, you can specify a text string that the Model Explorer uses to filter the displayed objects. Use the **Filter Contents** text box at the top of the **Contents** pane to specify the text for filtering.



For details, see Filtering Contents.

Model Explorer: Finding Variables That Are Used by a Model or Block

In the Model Explorer, you can get a list of variables that a model or block uses. For example, one way to get that list of variables is:

- 1 In the **Contents** pane, right-click the block for which you want to find what variables it uses.
- 2 Select the **Find Referenced Variables** menu item.

You can also use the following approaches to find variables that are used by a model or block:

- In the Model Explorer, in the **Model Hierarchy** pane, right-click a model or block and select the **Find Referenced Variables** menu item.
- In the Model Explorer, in the search bar, use the for **Referenced Variables** search type option.
- In the Model Editor, right-click a block, subsystem, or the canvas and select the **Find Referenced Variables** option.

For details, see Finding Variables That Are Used by a Model or Block.

Model Explorer: Finding Blocks That Use a Variable

You can use the Model Explorer to get a list of blocks that use a workspace variable. One way to get that list of blocks is to right-click a variable in the **Contents** pane and select the **Find Where Used** menu item.

You can also find blocks that use a variable using one of these approaches:

- In the **Search** bar, select the for **Variable Usage** search type option.
- In the **Search Results** tab, right-click a variable and select the **Find Where Used** menu item.

For details, see Finding Blocks That Use a Specific Variable.

Model Explorer: Exporting and Importing Workspace Variables

You can export workspace variables from the Model Explorer to a MATLAB file or MAT-file.

One way to select the variables to export is by right-clicking the workspace node (for example, Base Workspace) and selecting the **Export** menu item.

Another way to select variables to export is to:

- 1 In the **Contents** pane, select the variables that you want to export.
- 2 Right-click on one of the highlighted variables and select the **Export Selected** menu item.

Also, you can import variables into a workspace in the Model Explorer:

- 1 In the **Model Hierarchy** pane, right-click the workspace into which you want to import variables.
- 2 Select the **Import** menu item.

For details, see Exporting Workspace Variables and Importing Workspace Variables.

Model Explorer: Link to System

The **Contents of** link at the top left side of the **Contents** pane links to the currently selected node in the **Model Hierarchy** pane.

Lookup Table Editor Can Now Propagate Changes in Table Data to Workspace Variables with Nonstandard Data Format

In R2010b, the Lookup Table Editor can propagate changes in table data to workspace variables with nonstandard data format when you:

- Use `sl_customization.m` to register a customization function for the Lookup Table Editor.
- Store this customization function on the MATLAB search path.

For more information, see Lookup Table Editor in the Simulink User's Guide.

Enhanced Designation of Hybrid Sample Time

Because of a new sample time enhancement, a block or signal with a continuous and a fixed in minor step sample time is no longer designated as hybrid. Instead, the block or signal is continuous and colored black. This enhancement assists in identifying hybrid subsystems that require attention.

Inspect Solver Jacobian Pattern

You can now inspect the solver Jacobian pattern in MATLAB and thereby determine if the pattern for your model is sparse. If so, the Sparse Perturbation Method and the Sparse Analytical Method may be able to take advantage of this sparsity pattern to reduce the number of computations necessary and thereby improve performance. For a demonstration that explains how to inspect and assess the sparsity pattern, see Exploring the Solver Jacobian Structure of a Model.

Inspection of Values of Elements in Checksum

You can now use `Simulink.BlockDiagram.getChecksum` to inspect the individual values of the elements making up the `ConfigSet` checksum.

Conversion of Error and Warning Messages Identifiers

In R2010b, all error and warning message identifiers that Simulink issues have a converted format. As part of this conversion, error and warning identifiers changed from a two-part format to a three-part format. For example, the message identifier 'Simulink:SL_SetParamWriteOnly' is now 'Simulink:Command:SetParamWriteOnly'.

Version History

Scripts that search for specific message identifiers or that turn off warning messages using an identifier must be updated with the new error and warning message identifiers. For an example script and a complete mapping of the new identifiers to the original identifiers, see <https://www.mathworks.com/support/solutions/en/data/1-CNY5F6/index.html>.

View and Compare Logged Signal Data from Multiple Simulations Using New Simulation Data Inspector Tool

This release introduces the new Simulation Data Inspector tool for quickly viewing and comparing logged signal data. You can use the tool to:

- View signal data in a graph
- View a comparison of specified signal data in a graph, including a plot of their differences
- Store signal data for multiple simulations so that you can specify and compare signal data between multiple simulations

For more information, see [Inspecting and Comparing Logged Signal Data and Basic Simulation Workflow](#).

Viewing Requirements Linked to Model Objects

If your model, or blocks in your model, has links to requirements in external documents, you can now perform the following tasks without a Simulink Verification and Validation license:

- Highlight objects in a model that have links to requirements
- View information about a requirement
- Navigate from a model object to associated requirements
- Filter requirements highlighting based on keywords

S-Functions

Legacy Code Tool Support for Arrays of Simulink.Bus

The Legacy Code Tool now supports arrays of `Simulink.Bus` objects as valid data types in function specifications. For more information see Supported Data Types under Declaring Legacy Code Tool Function Specifications.

S-Functions Generated with `legacy_code` function and `singleCPPMexFile` S-Function Option Must Be Regenerated

Due to an infrastructure change, if you have generated an S-function with a call to `legacy_code` that defines the S-function option `singleCPPMexFile`, you must regenerate the S-function to use it with this release of Simulink.

For more information, see the description of `legacy_code` and Integrating Existing C Functions into Simulink Models with the Legacy Code Tool.

Version History

If you have generated an S-function with a call to `legacy_code` that defines the S-function option `singleCPPMexFile`, regenerate the S-function to use it with this release of Simulink.

Level-2 M-File S-Function Block Name Changed to Level-2 MATLAB S-Function

Level-2 MATLAB S-Function is the new name for the Simulink block previously called Level-2 M-File S-Function. In the Function Block Parameters dialog box, **S-function name** is the new name for the parameter previously called **M-file name**. The block type M-S-Function remains unchanged.

Version History

If you have a MATLAB script that uses the `add_block` function with the old block name, you need to update your script with the new name.

Functions Removed

Function Being Removed in a Future Release

This function will be removed in a future release of Simulink software.

Function Name	What Happens When You Use This Function?	Compatibility Considerations
simplot	Still works in R2010b	Use the Simulation Data Inspector to plot simulation data.

R2010a

Version: 7.5

New Features

Bug Fixes

Version History

Simulation Performance

Computation of Sparse and Analytical Jacobian for Implicit Simulink Solvers

The implicit Simulink solvers now support numerical and analytical methods for computing the Jacobian matrix in one of the following representations: sparse perturbation, full perturbation, sparse analytical, and full analytical. The sparse methods attempt to improve performance by taking advantage of sparsity information associated with the Jacobian matrix. Similarly, the analytical methods attempt to improve performance by computing the Jacobian using analytical equations rather than the perturbation equations.

Since the applicability of these representations is highly model dependent, an auto option directs Simulink to use a heuristic to choose an appropriate representation for your model. In the case of a model that has a large number of states and for which the Jacobian is computed in sparse analytical form, the performance improvement may be substantial. In general, the performance improvement achieved varies from model to model.

Sparse Perturbation Support for RSim and Rapid Accelerator Mode

For implicit Simulink solvers, the numerical sparse perturbation method for solving the Jacobian supports both RSim and Rapid Accelerator mode.

Increased Accuracy in Detecting Zero-Crossing Events

The zero-crossing bracketing algorithm now uses a smaller tolerance for defining the interval in which an event occurs. The resulting increased accuracy of locating an event means that existing models may exhibit slightly different numerical results.

Saving Code Generated by Accelerating Models to slprj Folder

In Accelerator mode and in Rapid Accelerator mode, a build has historically resulted in the creation of generated code, respectively, in the *modelName_accel_rtw* and the *modelName_raccel_rtw* folders in the current working folder. However, in order to be more consistent with other builds, in R2010a and future releases, these files will be created in the *slprj/accel/modelname* and the *slprj/raccel/modelname* folders.

Component-Based Modeling

Defining Mask Icon Variables

For model efficiency, use the **Icon & Ports** pane to run MATLAB code and to define variables used by the mask icon drawing commands. In releases earlier than R2010a, you had to use the **Initialization** pane to define variables used for icon drawing.

Simulink executes the MATLAB code in the **Icon & Ports** pane only when the block icon needs to be drawn. If you include variables used by mask icon drawing commands in the **Initialization** pane, Simulink evaluates the variables as part of simulation and code generation.

For more information, see *Defining a Mask Icon*.

Version History

Starting in R2010a, you can execute any MATLAB function in the **Ports & Icons** pane of the Mask Editor. If a variable in the mask workspace has the same name as a function in the **Ports & Icons** pane, Simulink returns an error.

For Each Subsystem Block

The For Each Subsystem block is very useful for modeling scenarios where you need to repeat the same algorithm on individual elements (or submatrices) of an input signal. The set of blocks within the subsystem represent the algorithm that is to be applied to a single element (or submatrix) of the original signal. You can configure the inputs of the subsystem to decompose the corresponding inputs into elements (or submatrices), and configure the outputs to suitably concatenate the processed results. Additionally, each block that has states inside this subsystem maintains separate sets of states for each element or submatrix it processes. Consequently, the operation of this subsystem is akin to copying the contents of the subsystem as many times as the number of elements in the original input signal, and then processing each element through its respective subsystem copy.

An additional benefit of this subsystem is that it may be utilized to improve code reuse in Real-Time Workshop generated code for certain models. Consider a model containing two reusable Atomic Subsystems with the same scalar algorithm applied to each element of the signal. If the input signal dimensions for these subsystems are different, you will find that two distinct functions are produced in the code generated by Real-Time Workshop for this model. Now, if you were to convert the two subsystems to For Each Subsystems such that the contents of each processes a single scalar element, then you will find that the two subsystems produce a single function in the code generated by Real-Time Workshop. This function is parameterized by the number of elements to be processed.

New Function-Call Split Block

A new Function-Call Split block allows you to branch periodic and asynchronous function-call signals and connect them to multiple function-call subsystems (or models). These subsystems (or models) are guaranteed to execute in the order determined by their data dependencies. If a deterministic order cannot be computed, the model produces an error.

To test the validity of your function-call connections, use the Model Advisor diagnostic, **Check usage of function-call connections**. This diagnostic determines if:

- **Configurations > Diagnostics > Connectivity > Invalid function-call connection** is set to error
- **Configuration Parameters > Diagnostics > Connectivity > Context-dependent inputs** is set to Enable All

Trigger Port Enhancements

You can use trigger ports, which you define with a Trigger block, in new ways:

- Place edge-based (rising, falling, or either), as well as function-call, trigger ports at the root level of a model. Before R2010a, to place a trigger port in a root-level model, you had to set the trigger type to function-call.
- Place triggered ports in models referenced by a Model block. See Triggered Models.
- Lock down the data type, port dimension, and trigger signal sample time. To specify these values, use the new **Signal Attributes** pane of the Block Parameters dialog box of the Trigger block. Specifying these attributes is useful for unit testing and running standalone simulation of a subsystem or referenced model that has an edge-based trigger port. See Triggered Models.

Version History

When you add a trigger port to a root-level model, if you use the **File > Save As** option to specify a release before R2010a, Simulink replaces the trigger port with an empty subsystem.

Embedded MATLAB Function Blocks

New Ability to Use Global Data

Embedded MATLAB Function blocks are now able to use global data within a Simulink model and across multiple models.

This feature provides these benefits:

- Allows you to share data between Embedded MATLAB Function blocks and other Simulink blocks without introducing additional input and output wires in your model. This reduces unnecessary clutter and improves the readability of your model.
- Provides a means of scoping the visibility of data within your model.

For more information, see [Using Global Data with the MATLAB Function Block](#) in the Simulink documentation.

Support for Logical Indexing

Embedded MATLAB Function blocks now support logical indexing when variable sizing is enabled. Embedded MATLAB supports variable-size data by default for MEX and C/C++ code generation.

For more information about logical indexing, see [Using Logicals in Array Indexing](#) in the MATLAB documentation.

Support for Variable-Size Matrices in Buses

Embedded MATLAB Function blocks now support Simulink buses containing variable-size matrices as inputs and outputs.

Support for Tunable Structure Parameters

Embedded MATLAB Function blocks now support tunable structure parameters. See [Working with Structure Parameters in MATLAB Function Blocks](#).

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

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

Simulink Data Management

New Function Finds Variables Used by Models and Blocks

The new `Simulink.findVars` function returns information about workspace variables and their usage. For example, you can use `Simulink.findVars`, sometimes in conjunction with other Simulink functions, to:

- Identify all workspace variables used by a model or block
- Identify any workspace variables unused by a model or block
- Search a model for all places where a specified variable is referenced
- Subdivide a model, including only necessary variables with each model

See `Simulink.findVars` and the other Simulink functions referenced on that page for more information.

MATLAB Structures as Tunable Structure Parameters

You can create a MATLAB structure that groups base workspace variables into a hierarchy, and dereference the structure fields to provide values in Simulink block parameter expressions. This technique reduces base workspace clutter and allows related workspace variables to be conveniently grouped. However, in previous releases you could not use a MATLAB structure as a masked subsystem or a model reference argument, and no value given by a MATLAB structure field could be tuned. These restrictions limited the usefulness of MATLAB structures for grouping variables used in block parameter expressions.

In R2010a, these restrictions no longer apply to MATLAB structures that contain only numeric data. You can use a numeric structure, or any substructure within it, as a masked subsystem or a model reference argument, thereby passing all values in the structure with a single argument. You can also control MATLAB structure tunability using the same techniques that control MATLAB variable tunability. In R2010a, all values in a given structure must be either tunable or nontunable. See [Using Structure Parameters](#) for more information.

Simulink.saveVars Documentation Added

The `Simulink.saveVars` function was added in R2009b but was incompletely documented. See [New Function Exports Workspace Variables and Values](#) for more information.

Custom Floating-Point Types No Longer Supported

Custom floating-point types, `float(TotalBits, ExpBits)`, are no longer supported.

Version History

If you have code that uses custom floating-point types, modify this code using one of these methods:

- Replace calls to `float(TotalBits, ExpBits)` with calls to `fixdt('double')` or `fixdt('single')` as appropriate.

- Create your own custom float replacement function.

Write a MATLAB function `custom_float_user_replacement` and place the file on your MATLAB path. This function must take `TotalBits` and `ExpBits` as input arguments and return a supported numeric type object, such as `fixdt('double')` or `fixdt('single')`.

For example,

```
function DataType = custom_float_user_replacement(TotalBits,ExpBits)

if (TotalBits <= 32) && (ExpBits <= 8)
    DataType = numerictype('single');
else
    DataType = numerictype('double');
end
```

In R2010a and future releases, if the file `custom_float_user_replacement.m` is on your MATLAB path, calls to `float(TotalBits, ExpBits)` automatically call `custom_float_user_replacement(TotalBits, ExpBits)`.

Data Store Logging

You can log the values of a local or global data store data variable for all the steps in a simulation. Data store logging is useful for:

- Model debugging - view the order of all data store writes
- Confirming a model modification - use the logged data to establish a baseline for comparing results to identify the impact of a model modification

To log a local data store that you create with a Data Store Memory block:

- Use the new **Logging** pane of the Block Parameters dialog box for the Data Store Memory block.
- Enable data store logging with the new **Configuration Parameters > Data Import/Export > Data stores** parameter.

To log a data store defined by a `Simulink.Signal` object, from the MATLAB command line, set `DataLogging` (which is a property of the `LoggingInfo` property of `Simulink.Signal`) to 1.

For details, see Logging Data Stores. To see an example of logging a global data store, run the `sldemo_mdref_dsm` demo.

Models with No States Now Return Empty Variables

Simulink creates empty variables for state logging (`xout`) or final state logging (`xfinal`), if both of these conditions apply:

- A model has no states.
- In the **Configuration Parameters > Data Import/Export** pane, you enable the **States, Final States**, or both parameters (the default is off).

Version History

If you configure your model to return empty variables when it has no states, then a possible result is that Simulink creates more variables than it did in previous releases.

Using model variants, running different models in batch mode, tuning models, or reconfiguring models can produce unexpected results based on the state values. For example, if you simulate a model that produces a state value, and then run a model variant that produces no state, Simulink overwrites the state value with an empty variable. If your model depends on the first state value not being overwritten if no state is returned in a subsequent simulation (which was the case in previous releases), then you get unexpected results.

To File Block Enhancements

The To File block now supports:

- Saving very large data sets that may be too large to fit in RAM
- Saving logged data up until the point of a premature ending of simulation processing. Previously, if the simulation processing did not complete, then To File did not store any logged data for that simulation.
- A new **Save format** parameter to control whether the block uses `Timeseries` or array format for data.
 - Use `Timeseries` format for writing multidimensional, real, or complex inputs, with different data types, (for example, built-in data types, including `Boolean`; enumerated (`enum`) data and fixed-point data with a word length of up to 32 bits.
 - Use `Array` format only for one-dimensional, double, noncomplex inputs. Time values are saved in the first row. Additional rows correspond to input elements.

Version History

For data saved using MAT file versions prior to 7.3, the From File block can only load two-dimensional arrays consisting of one-dimensional, double, noncomplex samples. To load data of any other type, complexity, or dimension, use a `Timeseries` object and save the file using MAT file version 7.3 or later. For example, use `'save file_name -v7.3 timeseries_object'`:

```
save file_name -v7.3 timeseries_object
```

From File Block Enhancements

The From File block now supports:

- Incremental loading of very large data sets that may be too large to fit in RAM
- Built-in data types, including `Boolean`
- Fixed-point data with a word length of up to 32 bits
- Complex data
- Multidimensional data

Root Inport Support for Fixed-Point Data Contained in a Structure

You can now use a root (top-level) Inport block to supply fixed-point data that is contained in a structure.

In releases before R2010a, you had to use a Simulink.Timeseries object instead of a structure.

Simulink Signal Management

Enhanced Support for Proper Use of Bus Signals

To improve model reliability and robustness, avoid mixing Mux blocks and bus signals. To help you use Mux blocks and bus signals properly, R2010a adds these enhancements:

- When Simulink detects Mux block and bus signal mixtures, the Mux blocks used to create bus signals diagnostic now generates:
 - A warning when all the following conditions apply:
 - You load a model created in a release before R2010a.
 - The diagnostic is set to 'None'.
 - Simulink detects improper Mux block usage.
 - An error for new models
- Two new diagnostics in the **Configuration Parameters > Diagnostics > Connectivity** pane:
 - The Non-bus signals treated as bus signals diagnostic detects when Simulink implicitly converts a non-bus signal to a bus signal to support connecting the signal to a Bus Assignment or Bus Selector block.
 - The Repair bus selections diagnostic repairs broken selections in the Bus Selector and Bus Assignment block parameters dialog boxes that are due to upstream bus hierarchy changes.

Version History

In R2010a, if you load a model created in a prior release, you might get warning messages that you did not get before. To avoid getting Mux block-related warnings for existing models that you want to load in R2010a, use the `slreplace_mux` function to substitute Bus Creator blocks for any Mux blocks used to create buses signals.

Bus Initialization

In releases before R2010a:

- For *virtual* buses, you could specify a non-zero scalar or vector initial condition (IC) value that applies to all elements of the bus. You could use a vector value only if all bus elements use the same data type.
- For *nonvirtual* buses, the only value you could specify was zero.

In R2010a, you can create a MATLAB structure for an IC. You can:

- Specify ICs for all or a subset of the bus elements.
- Use the new `Simulink.Bus.createMATLABStruct` helper method to create a full IC structure.
- Use the new Model Advisor Simulink check, **Check for partial structure parameter usage with bus signals**, to detect when structure parameters are not consistent in shape with the associated bus signal.

Using IC structures helps you to:

- Specify nonzero initial conditions
- Specify initial conditions for mixed-dimension signals
- Apply a different IC for each signal in the bus
- Specify ICs for a subset of signals in a bus without specifying ICs for all the signals
- Use the same ICs for multiple blocks, signals, or models

For information about creating and using initial condition structures, see [Specifying Initial Conditions for Bus Signals](#).

S-Functions for Working with Buses

The following S-functions provide a programmatic interface for working with buses:

S-function	Description
<code>ssGetBusElementComplexSignal</code>	Get the signal complexity for a bus element.
<code>ssGetBusElementDataType</code>	Get the data type identifier for a bus element.
<code>ssGetBusElementDimensions</code>	Get the dimensions of a bus element.
<code>ssGetBusElementName</code>	Get the name of a bus element.
<code>ssGetBusElementNumDimensions</code>	Get the number of dimensions for a bus element.
<code>ssGetBusElementOffset</code>	Get the offset from the start of the bus data type to a bus element.
<code>ssGetNumBusElements</code>	Get the number of elements in a bus signal.
<code>ssGetSFcnParamName</code>	Get the value of a block parameter for an S-function block.
<code>ssIsDataTypeABus</code>	Determine whether a data type identifier represents a bus signal.
<code>ssRegisterTypeFromParameter</code>	Register a data type that a parameter in the Simulink data type table specifies.
<code>ssSetBusInputAsStruct</code>	Specify whether to convert the input bus signal for an S-function from virtual to nonvirtual.
<code>ssSetBusOutputAsStruct</code>	Specify whether the output bus signal from an S-function must be virtual or nonvirtual.
<code>ssSetBusOutputObjectName</code>	Specify the name of the bus object that defines the structure and type of the output bus signal.

Command Line API for Accessing Information About Bus Signals

You can use two new signal property parameters to get information about the type and hierarchy of a signal programmatically:

- `CompiledBusType`
 - Returns information about whether the signal connected to a port is a bus, and if so, whether it is a virtual or nonvirtual bus
- `SignalHierarchy`

- Returns the signal name of the signal. If the signal is a bus, the parameter also returns the hierarchy and names of the bus signal.

See Model Parameters and View Information about Buses.

Signal Name Propagation for Bus Selector Block




The new `SignalNameFromLabel` port parameter supports signal name propagation for Bus Creator block input signals whenever you change the name of an input signal programmatically. You can set this parameter with the `set_param` command, specifying either a port or line handle and the signal name to propagate.

See Model Parameters.

Block Enhancements

New Square Root Block

You can use the new Sqrt block to perform square-root calculations. This block includes the following functions:

Function	Icon
sqrt	
signedSqrt	
rSqrt	

Version History

The `sqrt` and `1/sqrt` functions no longer appear in the Math Function block. For backward compatibility, models with a Math Function block that uses one of these two functions continue to work. However, consider running the `slupdate` function on your model. `slupdate` replaces any Math Function block that uses `sqrt` or `1/sqrt` with an equivalent Sqrt block that ensures the same behavior.

New Second-Order Integrator Block

You can use the new Second-Order Integrator block to model second-order systems that have bounds on their states. This block is useful for modeling physical systems, for example, systems that use Newton's Second Law and have constraints on their motion.

Benefits of using this block include:

- Highly accurate results
- Efficient detection of zero crossings
- Prevention of direct feedthrough and algebraic loops

New Find Nonzero Elements Block

You can use the new Find block to locate all nonzero elements of an input signal. This block outputs the indices of nonzero elements in linear indexing or subscript form and provides these benefits:

When you use the block to...	You can...
Convert logical indexing to linear indexing	Use the linear indices you get from processing a logical indexing signal as the input to a Selector or Assignment block

When you use the block to...	You can...
Extract subscripts of nonzero values	Use the subscript of matrices for 2-D or higher-dimensional signal arrays to aid with image processing
Represent sparse signals	Use indices and values as a compact representation of sparse signals

PauseFcn and ContinueFcn Callback Support for Blocks and Block Diagrams

The new `PauseFcn` and `ContinueFcn` callbacks detect clicking of the **Pause** and **Continue** buttons during simulation. You can set these callbacks using the `set_param` command or the **Callbacks** tab of the Model Properties dialog box. Both the `PauseFcn` and `ContinueFcn` callbacks support Normal and Accelerator simulation modes.

Gain Block Can Inherit Parameter Data Type from Gain Value

The Gain block now supports the **Parameter data type** setting of `Inherit: Inherit from 'Gain'`. This enhancement provides the benefit of inheriting the parameter data type directly from the **Gain** parameter. For example:

If you set Gain to...	The parameter data type inherits...
2	double
single(2)	single
int8(2)	int8

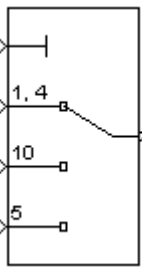
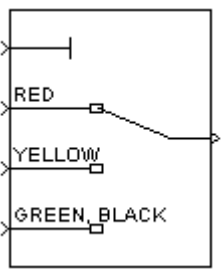
Direct Lookup Table (n-D) Block Enhancements

The Direct Lookup Table (n-D) block now supports:

- Multidimensional signals for the table input port
- Fixed-point data types for the table input port
- Explicit specification of the table data type in the block dialog box

Multiport Switch Block Allows Explicit Specification of Data Port Indices

The icon for the Multiport Switch block now shows the values of indices on data port labels. This enhancement helps you identify the data inputs without having to open the block dialog box:

Block Parameter Settings	Block Icon
Data port order: Specify indices Data port indices (e.g. {1,[2,3]}): {[1,4],10,5}	
Data port order: Specify indices Data port indices (e.g. {1,[2,3]}): {myColors.RED, myColors.YELLOW, [myColors.GREEN, myColors.BLACK]}	

When you load existing models that contain the Multiport Switch block, the following parameter mapping occurs:

Block Parameter Settings of a Model from R2009b or Earlier	Block Parameter Settings When You Load the Model in R2010a
Number of inputs: 3 <input type="checkbox"/> Use zero-based indexing	Data port order: One-based contiguous Number of data ports: 3
Number of inputs: 3 <input checked="" type="checkbox"/> Use zero-based indexing	Data port order: Zero-based contiguous Number of data ports: 3

The following command-line parameter mapping applies:

Old Prompt on Block Dialog Box	New Prompt on Block Dialog Box	Old Command-Line Parameter	New Command-Line Parameter
Number of inputs	Number of data ports	Inputs	Same
Use zero-based indexing	Data port order	zeroidx	DataPortOrder

The parameter mapping in R2010a ensures that you get the same block behavior as in previous releases.

Version History

In R2010a, a warning appears at compile time when your model contains a Multiport Switch block with the following configuration:

- The control port uses an enumerated data type.
- The data port order is contiguous.

During edit time, the block icon cannot show the mapping of each data port to an enumerated value. This configuration can also lead to unused ports during simulation and unused code during Real-Time Workshop code generation.

Run the `sUpdate` function on your model to replace each Multiport Switch block of this configuration with a block that explicitly specifies data port indices. Otherwise, your model might not work in a future release.

In R2010a, the following Multiport Switch block configuration also produces a warning at compile time:

- The control port uses a fixed-point or built-in data type.
- The data port order is contiguous.
- At least one of the contiguous data port indices is not representable with the data type of the control port.

The warning alerts you to unused ports during simulation and unused code during Real-Time Workshop code generation.

Trigonometric Function Block Supports CORDIC Algorithm and Fixed-Point Data Types

When you select `sin`, `cos`, or `sincos` for the Trigonometric Function block, additional parameters are available.

New Block Parameter	Purpose	Benefit
Approximation method	Specify the type of approximation the block uses to compute output: <code>None</code> or <code>CORDIC</code> .	Enables you to use a faster method of computing block output for fixed-point and HDL applications.
Number of iterations	For the <code>CORDIC</code> algorithm, specify how many iterations to use for computing block output.	Enables you to adjust the precision of your block output.

This block now supports fixed-point data types when you select `sin`, `cos`, or `sincos` and set **Approximation method** to `CORDIC`.

Enhanced Block Support for Enumerated Data Types

The following Simulink blocks now support enumerated data types:

- Data Type Conversion Inherited

- Data Type Duplicate
- Interval Test
- Interval Test Dynamic
- Probe (input only)
- Relay (output only)
- Unit Delay Enabled
- Unit Delay Enabled Resettable
- Unit Delay Resettable
- Unit Delay With Preview Enabled
- Unit Delay With Preview Enabled Resettable
- Unit Delay With Preview Enabled Resettable External RV
- Unit Delay With Preview Resettable
- Unit Delay With Preview Resettable External RV

For more information, see Enumerations and Modeling in the Simulink User's Guide.

Lookup Table Dynamic Block Supports Direct Selection of Built-In Data Types for Outputs

In R2010a, you can select the following data types directly for the **Output data type** parameter of the Lookup Table Dynamic block:

- double
- single
- int8
- uint8
- int16
- uint16
- int32
- uint32
- boolean

Previously, you had to enter an expression for **Output data type** to specify a built-in data type.

Compare To Zero and Wrap To Zero Blocks Now Support Parameter Overflow Diagnostic

If the input data type to a Compare To Zero or Wrap To Zero block cannot represent zero, detection of this parameter overflow occurs. In the **Diagnostics > Data Validity** pane of the Configuration Parameters dialog box, set **Parameters > Detect overflow** to warning or error.

Data Type Duplicate Block Enhancement

The Data Type Duplicate block is now a built-in block. Previously, this block was a masked S-Function. The read-only **BlockType** parameter has changed from S-Function to `DataTypeDuplicate`.

Version History

In R2010a, signal propagation might behave differently from previous releases. As a result, your model might not compile under these conditions:

- Your model contains a Data Type Duplicate block in a source loop.
- Your model has underspecified signal data types.

If your model does not compile, set data types for signals that are not fully specified.

Lookup Table and Lookup Table (2-D) Blocks To Be Deprecated in a Future Release

In a future release, the Lookup Table and Lookup Table (2-D) blocks will no longer appear in the Simulink Library Browser. Consider replacing instances of those two blocks by using 1-D and 2-D versions of the Lookup Table (n-D) block. Among other enhancements, the Lookup Table (n-D) block supports the following features that the other two blocks do not:

- Specification of parameter data types different from input or output signal types
- Reduced memory use and faster code execution for evenly spaced breakpoints that are nontunable
- Fixed-point data types with word lengths up to 128 bits
- Specification of index search method
- Specification of action for out-of-range inputs

To upgrade your model:

Step	Description	Reason
1	Run the Simulink Model Advisor check for Check model, local libraries, and referenced models for known upgrade issues.	Identify blocks that do not have compatible settings with the Lookup Table (n-D) block.
2	For each block that does not have compatible settings with the Lookup Table (n-D) block: <ul style="list-style-type: none"> • Decide how to address each warning. • Adjust block parameters as needed. 	Modify each Lookup Table or Lookup Table (2-D) block to make them compatible.
3	Repeat steps 1 and 2 until you are satisfied with the results of the Model Advisor check.	Ensure that block replacement works for the entire model.
4	Run the <code>supdate</code> function on your model.	Perform block replacement with the Lookup Table (n-D) block.

Version History

The Model Advisor check groups all Lookup Table and Lookup Table (2-D) blocks into three categories:

- Blocks that have compatible settings with the Lookup Table (n-D) block
- Blocks that have incompatible settings with the Lookup Table (n-D) block
- Blocks that have repeated breakpoints

Blocks with Compatible Settings

When a block has compatible parameter settings with the Lookup Table (n-D) block, automatic block replacement can occur without backward incompatibilities.

Lookup Method in the Lookup Table or Lookup Table (2-D) Block	Parameter Settings in the Lookup Table (n-D) Block After Block Replacement	
	Interpolation	Extrapolation
Interpolation-Extrapolation	Linear	Linear
Interpolation-Use End Values	Linear	None-Clip
Use Input Below	None-Flat	Not applicable

Depending on breakpoint characteristics, the Lookup Table (n-D) block uses one of two index search methods.

Breakpoint Characteristics in the Lookup Table or Lookup Table (2-D) Block	Index Search Method in the Lookup Table (n-D) Block After Block Replacement
Not evenly spaced	Binary search
Evenly spaced and tunable	A prompt appears, asking you to select Binary search or Evenly spaced points.
Evenly spaced and nontunable	

The Lookup Table (n-D) block also adopts other parameter settings from the Lookup Table or Lookup Table (2-D) block. For parameters that exist only in the Lookup Table (n-D) block, the following default settings apply after block replacement:

Lookup Table (n-D) Block Parameter	Default Setting After Block Replacement
Breakpoint data type	Inherit: Same as corresponding input
Action for out-of-range input	None

Blocks with Incompatible Settings

When a block has incompatible parameter settings with the Lookup Table (n-D) block, the Model Advisor shows a warning and a recommended action, if applicable.

- If you perform the recommended action, you can avoid incompatibility during block replacement.
- If you use automatic block replacement without performing the recommended action, you might see numerical differences in your results.

Incompatibility Warning	Recommended Action	What Happens for Automatic Block Replacement
The Lookup Method is Use Input Nearest or Use Input Above. The Lookup Table (n-D) block does not support these lookup methods.	Change the lookup method to one of the following: <ul style="list-style-type: none"> Interpolation - Extrapolation Interpolation - Use End Values Use Input Below 	The Lookup Method changes to Interpolation - Use End Values. In the Lookup Table (n-D) block, this setting corresponds to: <ul style="list-style-type: none"> Interpolation set to Linear Extrapolation set to None-Clip
The Lookup Method is Interpolation - Extrapolation, but the input and output are not the same floating-point type. The Lookup Table (n-D) block supports linear extrapolation only when all inputs and outputs are the same floating-point type.	Change the extrapolation method or the port data types of the block.	You also see a message that explains possible numerical differences.
The block uses small fixed-point word lengths, so that interpolation uses only one rounding operation. The Lookup Table (n-D) block uses two rounding operations for interpolation.	None	You see a message that explains possible numerical differences.

Blocks with Repeated Breakpoints

When a block has repeated breakpoints, the Model Advisor recommends that you change the breakpoint data and rerun the check. You cannot perform automatic block replacement for blocks with repeated breakpoints.

Elementary Math Block Now Obsolete

The Elementary Math block is now obsolete. You can replace any instance of this obsolete block in your model by using one of these blocks in the Math Operations library:

- Math Function
- Rounding Function
- Trigonometric Function

Version History

If you open a model that contains an Elementary Math block, a warning message appears. This message suggests running `slupdate` on your model to replace each instance of the obsolete block with an appropriate substitute.

If you try to start simulation or generate code for a model that contains this obsolete block, an error message appears.

DocBlock Block RTF File Compression

In R2010a, when you add or modify a DocBlock block that uses Microsoft RTF format and you save the model, Simulink compresses the RTF file. The saved RTF files with images are much smaller than in previous releases.

Version History

In R2010a, if you use `s_lupdate` or save a model that includes a DocBlock block that uses RTF format, you cannot run the model in an earlier version of Simulink.

To run a model that has a compressed RTF file in an earlier version of Simulink, use **Save As** to save the model in the format of the earlier release.

Simulink Extras PID Controller Blocks Deprecated

In R2010a, the PID Controller (with Approximate Derivative) and PID Controller blocks of the Simulink Extras library no longer appear in the Simulink Library Browser. For models created using R2009b or earlier, consider using the `s_lupdate` function to replace these blocks with the new PID Controller block of the Simulink/Continuous or Simulink/Discrete library. Among other enhancements, the new PID Controller block supports:

- Continuous-time and discrete-time modeling
- Ideal and Parallel controller forms
- Automatic PID tuning (requires a Simulink Control Design™ license)

For more information, see the PID Controller and PID Controller (2 DOF) block reference pages.

Version History

For backward compatibility, simulation and code generation of models that contain the deprecated PID Controller (with Approximate Derivative) or PID Controller block continue to work.

User Interface Enhancements

Model Explorer Column Views

The Model Explorer now supports column views, which specify sets of property columns to display in the **Contents** pane. The Model Explorer displays only the properties that are defined for the current column view. The Model Explorer does not add new properties dynamically as you add objects to the **Contents** pane. Using a defined subset of properties to display streamlines the task of exploring and editing model object properties and increases the density of the data displayed.

Model Explorer provides several standard column views with common property sets. You can:

- Select the column view based on the task you are performing
- Customize the standard column views
- Create your own column views
- Export and import column views saved in MAT-files, which you can share with other users

See The Model Explorer: Controlling Contents Using Views.

Version History

Column views replace the **Customize Contents** option provided in previous releases.

In R2010a, the Model Explorer provides a different interface for performing some of the tasks that you previously performed using **View** menu items. The following table summarizes differences between R2009b and R2010a.

R2009b View Menu Item	R2010a Model Explorer Interface Change
Dialog View	Replaced by Show Dialog Pane
Customize Contents	Replaced by Column View > Show Details
Show Properties	Eliminated; select Column View > Show Details to specify properties to display
Mark Nonexistent Properties	Replaced by Show Nonexistent Properties as ' - '
Library Browser	Eliminated (you can access the Library Browser from the Simulink Editor View menu)
List View Options	Replaced by Row Filter

Model Explorer Display of Masked Subsystems and Linked Library Subsystems

The Model Explorer now contains global options for specifying whether the Model Explorer displays the contents of library links and masked subsystems. These options also control whether the **Model Hierarchy** pane displays linked or masked subsystems. See Displaying Masked Subsystems and Displaying Linked Library Subsystems.

Version History

In R2010a, when you select a masked subsystem node in the **Model Hierarchy** pane, the **Contents** pane displays the objects of the subsystem, reflecting the global setting to display masked subsystems. In prior releases, if you selected a masked subsystem node, you needed to right-click the node and select **Look Under Mask** to view the subsystem objects in the **Contents** pane.

In R2010a, the search results reflect the **Show Library Links** and **Show Masked Subsystems** settings. In previous releases, you specified the **Look Inside Masked Subsystems** and **Look Inside Linked Subsystems** options as part of the search options. R2010a does not include those search options.

Model Explorer Object Count

The top-right section of the **Contents** pane displays a count of objects found for the currently selected nodes in the **Model Hierarchy** pane. The count indicates the number of objects displayed in the **Contents** pane, compared to the total number of objects in the currently selected nodes. The number of displayed objects is less than the total number of objects in scope when you filter some objects by using **View > Row Filter** options. See Object Count.

Model Explorer Search Option for Variable Usage

You can use the new `for Variable Usage` search type to search for blocks that use a variable that is defined in the base or model workspaces. See Search Bar Controls.

Model Explorer Display of Signal Logging and Storage Class Properties

The Model Explorer **Contents** pane displays the following additional properties for signal lines:

- Signal logging-related properties (such as `DataLogging`)
- Storage class properties, including properties associated with custom storage classes for signals

Displaying these properties in the **Contents** pane enables batch editing. Prior to R2010a, you could edit these properties only in the Signal Properties dialog box.

Model Explorer Column Insertion Options

In R2010a, right-clicking on a column heading in the Contents pane provides two new column insertion options:

- **Insert Path** - adds the `Path` property column to the right of the selected column.
- **Insert Recently Hidden Columns** - selects a property from a list of columns you recently hid, to add that property column to the right of the selected column

See Adding Property Columns.

Diagnostics for Data Store Memory Blocks

The Model Advisor 'By Task' folder now contains a Data Store Memory Blocks subfolder. This subfolder contains checks relating to Data Store Memory blocks that examine your model for:

- Multitasking, strong typing, and shadowing issues
- An enabled status of the read/write diagnostics
- Read/write issues

New Command-Line Option for RSim Targets

A new `-h` command-line option allows you to print a summary of the available options for RSim executable targets.

Simulink.SimulationOutput.get Method for Obtaining Simulation Results

The `Simulink.SimulationOutput` class now has a `get` method. After simulating your model, you can use this method to access simulation results from the `Simulink.SimulationOutput` object.

Simulink.SimState.ModelSimState Class has New `snapshotTime` Property

The `Simulink.SimState.ModelSimState` class has a new `snapshotTime` property. You can use this property to access the exact time at which Simulink took a “snapshot” of the simulation state (`SimState`) of your model.

Simulink.ConfigSet.saveAs to Save Configuration Sets

The `saveAs` method is added to the `Simulink.ConfigSet` class to allow you to easily save the settings of configuration sets as MATLAB functions or scripts. Using the MATLAB function or script, you can share and archive model configuration sets. You can also compare the settings in different configuration sets by comparing the MATLAB functions or scripts of the configuration sets.

For details, see *Save a Configuration Set* in the *Simulink User's Guide*.

S-Functions

Building C MEX-Files from Ada and an Example Ada Wrapper

In an R2008b release note, MathWorks announced that support for Ada S-functions in Simulink would be removed in a future release and a migration strategy would be forthcoming.

New S-Function API Checks for Branched Function-Calls

A new S-function API, `ssGetCallSystemNumFcnCallDestinations`, allows you to determine the number of function-call blocks that your S-function calls. Based on this returned number, you can then deduce whether or not your S-function calls a branched function-call.

You can call this SimStruct function from `mdlSetWorkWidths` or later in your S-function.

New C MEX S-Function API and M-File S-Function Flag for Compliance with For Each Subsystem

To allow a C MEX S-function to reside inside of a For Each Subsystem block, you must call the new `ssSupportsMultipleExecInstances` API and set the flag to true in the `mdlSetWorkWidths` method.

As for M-file S-functions, you must set the new flag `block.SupportsMultipleExecInstances` to true in the Setup section.

Legacy Code Tool Enhanced to Support Enumerated Data Types and Structured Tunable Parameters

The Legacy Code Tool has been enhanced to support

- Enumerated data types for input, output, parameters, and work vectors
- Structured tunable parameters

For more information about data types that the Legacy Code Tool supports, see [Supported Data Types](#). For more information about the Legacy Code Tool, see

- [Integrating Existing C Functions into Simulink Models with the Legacy Code Tool in the Writing S-Functions documentation](#)
- [legacy_code](#) function reference page

Version History

For enumerated data type support:

- If you upgrade from R2008b or later release, you can continue to compile the S-function source code and continue to use the compiled output from an earlier release without recompiling the code.
- If you upgrade from R2008a or earlier release, you cannot use enumerated types; the Simulink engine will display an error during simulation.

You cannot use tunable structured parameters with Legacy Code Tool in a release prior to R2010a.

Documentation Improvements

Modeling Guidelines for High-Integrity Systems

MathWorks intends the Modeling Guidelines for High-Integrity Systems document to be for engineers developing models and generating code for high-integrity systems using Model-Based Design with MathWorks products. This document describes creating Simulink models that are complete, unambiguous, statistically deterministic, robust, and verifiable. The document focus is on model settings, block usage, and block parameters that impact simulation behavior or code generated by the Real-Time Workshop Embedded Coder product.

These guidelines do not assume that you use a particular safety or certification standard. The guidelines reference some safety standards where applicable, including IEC 61508 and MISRA C .

You can use the Model Advisor to support adhering to these guidelines. Each guideline lists the checks that are applicable to that guideline.

For more information, see Modeling Guidelines for High-Integrity Systems in the Simulink documentation.

MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow Included in Help

MathWorks Automotive Advisory Board (MAAB) involves major automotive original equipment manufacturers (OEMs) and suppliers in the process of evolving MathWorks controls, simulation, and code generation products, including the Simulink, Stateflow, and Real-Time Workshop products. An important result of the MAAB has been the “MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow.” Help for the Simulink product now includes these guidelines. The MAAB guidelines link to relevant Model Advisor MAAB check help and MAAB check help links to relevant MAAB guidelines.

For more information, see MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow in the Simulink documentation.

R2009bSP1

Version: 7.4.1

Bug Fixes

R2009b

Version: 7.4

New Features

Bug Fixes

Version History

Simulation Performance

Single-Output `sim` Syntax

An enhanced `sim` command provides for greater ease of use and for greater compatibility with `parfor` loops. Since the command now saves all simulation results to a single object, the management of output variables is straightforward for all cases, including parallel computing.

Expanded Support by Rapid Accelerator

Simulink Rapid Accelerator mode now supports root inputs of enumerated data type and fixed-point parameters of any word length.

SimState Support in Accelerator Mode

Simulink Accelerator mode now supports the SimState feature. You can therefore save the simulation state and later resume the simulation from the exact save time.

Integer Arithmetic Applied to Sample Hit Computations

For fixed-step simulations, Simulink now computes sample time hits using integer arithmetic. This modification improves the timing resolution of sample hits of multirate models.

Version History

Previously, if an S-function had two rates, and if `ssIsSampleHit(S, idx1) == true && ssIsSampleHit(S, idx2) == true`, then Simulink would adjust the task times to be evaluated as `ssGetTaskTime(S, idx1) == ssGetTaskTime(S, idx2)`. Simulink no longer forces this equality; instead, Simulink now leaves the individual task times to be integer multiples of their corresponding periods. Consequently, existing code with logic that relies upon the equality of the task times needs to be updated.

In addition, the behavior of the command `get_param(model, 'SimulationTime')` is now different. Instead of returning the time of the next known sample hit at the bottom of the current step, this command now returns the current time.

Improved Accuracy of Variable-Step Discrete Solver

For variable-step discrete simulation of purely discrete models, where the fundamental step size is the same as the fastest discrete rate, Simulink now uses the specified start and stop times.

Version History

Previously, if the fundamental step size was equal to the fastest discrete rate, the Simulink simulation did not uniformly honor the user-specified start and stop times. Specifically, if the start and stop times were not exact multiples of the fundamental step size, then the start time was adjusted to the time of the first sample time hit and the simulation stopped at the sample time hit just before the specified stop time. However, if the simulation was required to hit certain time points (either by specifying

TSPAN in the `sim` command such as `'sim('Model_A',[0 10])'`, or via the `OutputTimes` parameter), then the start and stop times were not adjusted

Now Simulink variable-step simulation of purely discrete models consistently honors the user-specified start and stop times, irrespective of whether the fastest discrete sample time is the GCD of all of the other sample times

Component-Based Modeling

Enhanced Library Link Management

In R2009b, improved library link management (Links Tool) facilitates visualizing and restoring edited library links. See *Working with Library Links* for more information.

Enhanced Mask Editor Provides Tabs and Signal Attributes

You can use the R2009b Mask Editor to create a mask that has tabbed panes, and define the same signal attribute specifications in a mask that built-in Simulink blocks provide. See *Working with Block Masks*, *Simulink Mask Editor* and *Mask Icon Drawing Commands* for more information.

Model Reference Variants

Model reference variants allow you to configure any Model block to select its referenced model from a set of candidate models. The selection occurs when you compile the model that contains the Model block, and depends on the values of one or more MATLAB variables or Simulink parameters in the base workspace. To configure a Model block to select the model that it references, you:

- Provide a set of Boolean expressions that reference base workspace values.
- Associate each expression with one of the models that the block could reference.

When you compile the model, Simulink evaluates all the expressions. Each Model block that uses model reference variants then selects the candidate model whose associated expression is `true`, and ignores all the other models. Compilation then proceeds exactly as if you had entered the name of the selected model literally in the Model block's **Model name** field.

You can nest Model blocks that use variants to any level, allowing you to define any number of arbitrarily complex customized models within a single framework. No matter how many simulation environments you define, selecting one requires only setting variable or parameter values appropriately in the base workspace. See *Setting Up Model Variants* for more information.

Protected Referenced Models

A *protected model* is a referenced model from which all block and line information has been eliminated. Protecting a model does not use encryption technology. A protected model can be distributed without revealing the intellectual property that it embodies. The model is said to run in Protected mode, and gives the same results that its source model does when run in Accelerator mode.

You can use a protected model much as you could any referenced model that executes in Accelerator mode. Simulink tools work with protected models to the extent possible given that the model's contents are obscured. For example, the Model Explorer and the Model Dependency Viewer show the hierarchy under an ordinary referenced model, but not under a protected model. Signals in a protected model cannot be logged, because the log could reveal information about the protected model's contents.

When a referenced model requires object definitions or tunable parameters that are defined in the MATLAB base workspace, the protected version of the model may need some or all of those same definitions when it executes as part of a third-party model. Simulink provides techniques for

identifying and obtaining the needed data. You can use the Simulink Manifest Tools or other techniques to package the model and any data for delivery.

Protecting a model requires a Real-Time Workshop license, which makes code generation capabilities available for use internally when creating the protected version of the model. The receiver of a protected model does not need a Real-Time Workshop license to use the model, and cannot use Real-Time Workshop to generate code for the model or any model that references it.

To accommodate protected models, the Model block now accepts a suffix in the **Model name** field. This suffix can be `.mdl` for an unprotected model or `.mdlp` for a protected model. If the suffix is omitted, Model block first searches the MATLAB path for a block with the specified name and the suffix `.mdl`. If that search fails, the block searches the path for a model with the suffix `.mdlp`.

The Model block now has a field named `ProtectedModel`, a boolean that indicates whether the referenced model is protected, and three fields for representing the name of the referenced model in different formats: `ModelNameDialog`, `ModelName`, and `ModelFile`. See the Model block parameters in Ports & Subsystems Library Block Parameters for information about these parameters. For more information about protecting models, see Protecting Referenced Models.

Simulink Manifest Tools

Enhanced Simulink Manifest Tools now discover and analyze model variants, protected models, and Simscape files.

New manifest analysis options for controlling whether to report file dependency locations for user files, all files, or no files. For example, you may not want to view the file locations of all the dependencies on MathWorks products. This is typical if your main use of Simulink Manifest Tools is to discover and package all the required files for your model. By not analyzing file locations, you speed up report creation, and the report is smaller and easier to navigate. If you need to trace all dependencies to understand why a particular file or toolbox is required by a model, you can always regenerate the full report of all files.

The manifest report is enhanced with sortable columns, and now MATLAB Programs as well as P-files are reported in the manifest if both exist.

For more information, see Model Dependencies.

S-Function Builder

The S-Function Builder has been enhanced to support bus signals for managing complex signal interfaces. See Developing S-Functions for more information.

Embedded MATLAB Function Blocks

Support for Variable-Size Arrays and Matrices

Embedded MATLAB Function blocks now support variable-size arrays and matrices with known upper bounds. With this feature, you can define inputs, outputs, and local variables to represent data that varies in size at runtime.

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

The **Lock output scaling against changes by the autoscaling tool** check box is now **Lock data type setting against changes by the fixed-point tools**. Previously, this check box was visible only if you entered an expression or a fixed-point data type, such as `fixdt(1,16,0)`. This check box is now visible for any data type specification. This enhancement enables you to lock the current data type settings on the dialog box against changes that the Fixed-Point Advisor or Fixed-Point Tool chooses.

New Compilation Report for Embedded MATLAB Function Blocks

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

Version History

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

New Options for Controlling Run-time Checks for Faster Performance

In simulation, the code generated for Embedded MATLAB Function blocks includes various run-time checks. To reduce the size of the generated code, and potentially improve simulation times, you can use new **Simulation Target** configuration parameters to control whether or not your generated code performs:

- Integrity checks to detect violations of memory integrity in the generated code. For more information, see *Ensure memory integrity* in the Simulink Graphical User Interface.
- Responsiveness checks to periodically check for Ctrl+C breaks and refresh graphics. For more information, see *Ensure responsiveness* in the Simulink Graphical User Interface.

Embedded MATLAB Function Blocks Improve Size Propagation Behavior

Heuristics for size propagation have improved for underspecified models. During size propagation, Embedded MATLAB Function blocks no longer provide default sizes. Instead, for underspecified models, Simulink gets defaults from other blocks that have more size information.

Version History

Certain underspecified models that previously ran without error may now generate size mismatch errors. Examples of underspecified models include:

- Models that contain a cycle in which no block specifies output size
- Models that do not specify the size of input ports

To eliminate size mismatch errors:

- Specify sizes for the input ports of your subsystem or model.
- Specify sizes of all ports on at least one block in any loop in your model.

Simulink Data Management

New Function Exports Workspace Variables and Values

The new `Simulink.saveVars` function can save workspace variables and their values into a MATLAB file. The file containing the data is human-readable and can be manually edited. If Simulink cannot generate MATLAB code for a workspace variable, `Simulink.saveVars` saves that variable into a companion MAT-file rather than a MATLAB file. Executing the MATLAB file (which also loads any companion MAT file) restores the saved variables and their values to the workspace. See `Simulink.saveVars` for more information.

New Enumerated Constant Block Outputs Enumerated Data

Although the Constant block can output enumerated values, it provides many block parameters that do not apply to enumerated types, such as **Output minimum** and **Output maximum**. In R2009b, the **Sources** library includes the Enumerated Constant block. When you need a block that outputs constant enumerated values, use Enumerated Constant rather than Constant to avoid seeing irrelevant block parameters.

Enhanced Switch Case Block Supports Enumerated Data

The Switch Case block now supports enumerated data types for the input signal and case conditions. For more information, see Enumerations and Modeling and the Switch Case block documentation.

Code for Multiport Switch Block Shows Enumerated Values

In previous releases, generated code for a Multiport Switch block that uses enumerated data contains the underlying integer for each enumerated value rather than its name. In R2009b, the code contains the name of each enumerated value rather than its underlying integer. This change adds readability and facilitates comparing the code with the model, but has no effect on the behavior of the code. For more information, see Enumerations and Modeling and Multiport Switch.

Data Class Infrastructure Partially Deprecated

Some classes and properties in the Simulink data class infrastructure have been deprecated in R2009b. See Working with Data for information about Simulink data classes.

Version History

If you use any of the deprecated constructs, Simulink posts a warning that identifies the construct and describes one or more techniques for eliminating it. The techniques differ depending on the construct. You can ignore these warnings in R2009b, but MathWorks recommends making the described changes now because the deprecated constructs may be removed from future releases, upgrading the warnings to errors.

Saving Simulation Results to a Single Object

Enhanced `sim` command that saves all simulation results to a single object for easier management of simulation results.

Simulation Restart in R2009b

In order to restart an R2009a simulation in R2009b, you should first regenerate the initial SimState in R2009b.

Version History

The SimState that Simulink saves from a R2009a simulation might be incompatible with the internal representation of the same model in R2009b. Simulink detects this incompatibility when the R2009a SimState is used to restart a R2009b simulation. If the mismatch resides in the model interface only, then Simulink issues a warning. (You can use the Simulink diagnostic 'SimState interface checksum mismatch' to turn off such warnings or to direct Simulink to report an error.) However, if the mismatch resides in the structural representation of the model, then Simulink reports an error. To avoid these errors and warnings, you need to regenerate the initial SimState in R2009b.

Removing Support for Custom Floating-Point Types in Future Release

Support for custom floating-point types, `float(TotalBits, ExpBits)`, will be removed in a future release.

In R2009b, Simulink continues to process these types.

For more information, see `float`.

Simulink File Management

Removal of Functions

The following functions are no longer available:

- adams.m
- euler.m
- gear.m
- linsim.m
- rk23.m
- rk45.m

Deprecation of SaveAs to R12 and R13

In R2009b, you will no longer be able to use the `SaveAs` feature to save a model to releases R12 or R13. You will, however, be able to save models to R12 and R13 using the command-line. In R2010a, the command-line capability will also be removed.

Improved Behavior of `Save_System`

When you use the `save_system` function to save a model to an earlier release, you will no longer receive a dialog box that indicates that the save was successful.

Simulink Signal Management

Variable-Size Signals

New capability that allows signal sizes to change during execution facilitates modeling of systems with varying environments, resources, and constraints. For Simulink models that demonstrate using variable-size signals, see [Working with Variable-Size Signals](#)

Simulink Support

- Referenced Model
- Simulink Accelerator and Rapid Accelerator
- Bus Signals
- C-mex S-function
- Level-2 M-file S-function
- Simulink Debugger
- Signal Logging and Loading
- Block Run-Time Object

Simulink Block Support

Support for variable-size signal inputs and outputs in over 40 Simulink blocks including many blocks from the Math Operations library. For a list of Simulink blocks, see [Simulink Block Support for Variable-Size Signals](#)

Block Enhancements

New Turnkey PID Controller Blocks for Convenient Controller Simulation and Tuning

You can implement a continuous- or discrete-time PID controller with just one block by using one of the new PID Controller and PID Controller (2DOF) blocks. With the new blocks, you can:

- Configure your controller in any common controller configuration, including PID, PI, PD, P, and I.
- Tune PID controller gains either manually in the block or automatically in the new PID Tuner. (PID Tuner requires a Simulink Control Design license.)
- Generate code to implement your controller using any Simulink data type, including fixed-point data types (requires a Real-Time Workshop license).

You can set many options in the PID Controller and PID Controller (2DOF) blocks, including:

- Ideal or parallel controller configurations
- Optional output saturation limit with anti-windup circuitry
- Optional signal-tracking mode for bumpless control transfer and multiloop controllers
- Setpoint weighting in the PID Controller (2DOF) block

The blocks are available in the Continuous and Discrete libraries. For more information on using the blocks, see the PID Controller and PID Controller (2DOF) reference pages. For more information on tuning the PID blocks, see Automatic PID Tuning in the Simulink Control Design reference pages.

New Enumerated Constant Block Outputs Enumerated Data

Although the Constant block can output enumerated values, it provides many block parameters that do not apply to enumerated types, such as **Output minimum** and **Output maximum**. In R2009b, the **Sources** library includes the Enumerated Constant block. When you need a block that outputs constant enumerated values, use Enumerated Constant rather than Constant to avoid seeing irrelevant block parameters.

Enhanced Switch Case Block Supports Enumerated Data

The Switch Case block now supports enumerated data types for the input signal and case conditions. For more information, see Enumerations and Modeling and the Switch Case block documentation.

Code for Multiport Switch Block Shows Enumerated Values

In previous releases, generated code for a Multiport Switch block that uses enumerated data contains the underlying integer for each enumerated value rather than its name. In R2009b, the code contains the name of each enumerated value rather than its underlying integer. This change adds readability and facilitates comparing the code with the model, but has no effect on the behavior of the code. For more information, see Enumerations and Modeling and Multiport Switch.

Discrete Transfer Fcn Block Has Performance, Data Type, Dimension, and Complexity Enhancements

The following enhancements apply to the Discrete Transfer Fcn block:

- Improved numerics and run-time performance of outputs and states by reducing the number of divide operations in the filter to one
- Support for signed fixed-point and signed integer data types
- Support for vector and matrix inputs
- Support for input and coefficients with mixed complexity
- A new **Initial states** parameter for entering nonzero initial states
- A new **Optimize by skipping divide by leading denominator coefficient (a0)** parameter that provides more efficient implementation by eliminating all divides when the leading denominator coefficient is one. This enhancement provides optimized block performance.

Version History

Due to these enhancements, you might encounter the following compatibility issues:

- **Realization parameter removed**

The Real-Time Workshop software `realization` parameter has been removed from this block. You can no longer use the `set_param` and `get_param` functions on this block parameter. The generated code for this block has been improved to be similar to the former 'sparse' realization when the **Optimize by skipping divide by leading denominator coefficient (a0)** parameter is selected, while maintaining tunability as in the former 'general' realization when the parameter is not selected.

- **State changes**

Due to the reduction in the number of divide operations that the block performs, you might notice that your logged states have changed when the leading denominator coefficient is not one.

Lookup Table (n-D) Block Supports Parameter Data Types Different from Signal Data Types

The Lookup Table (n-D) block supports breakpoint data types that differ from input data types. This enhancement provides these benefits:

- Lower memory requirement for storing breakpoint data that uses a smaller type than the input signal
- Sharing of prescaled breakpoint data between two Lookup Table (n-D) blocks with different input data types
- Sharing of custom storage breakpoint data in generated code for blocks with different input data types

The Lookup Table (n-D) block supports table data types that differ from output data types. This enhancement provides these benefits:

- Lower memory requirement for storing table data that uses a smaller type than the output signal

- Sharing of prescaled table data between two Lookup Table (n-D) blocks with different output data types
- Sharing of custom storage table data in generated code for blocks with different output data types

The Lookup Table (n-D) block also supports separate data type specification for intermediate results. This enhancement enables use of a higher precision for internal computations than for table data or output data.

For consistency with other lookup table blocks, the **Process out-of-range input** parameter prompt is now **Action for out-of-range input**. Similarly, the command-line parameter is now `ActionForOutOfRangeInput`. For backward compatibility, the old command-line parameter `ProcessOutOfRangeInput` continues to work. The parameter settings also remain the same: `None`, `Warning`, or `Error`.

Reduced Memory Use and More Efficient Code for Evenly Spaced Breakpoints in Prelookup and Lookup Table (n-D) Blocks

For the Prelookup and Lookup Table (n-D) blocks, the generated code now stores only the first breakpoint, spacing, and number of breakpoints when:

- The breakpoint data is nontunable.
- The index search method is `Evenly spaced points`.

This enhancement reduces memory use and provides faster code execution. Previously, the code stored all breakpoint values in a set, regardless of the tunability or spacing of the breakpoints.

The following enhancements also provide more efficient code for the two blocks:

Block	Enhancement for Code Efficiency
Lookup Table (n-D)	Removal of unnecessary bit shifts for calculating the fraction
Prelookup and Lookup Table (n-D)	Use of simple division instead of computation-expensive function calls for calculating the index and fraction

Math Function Block Computes Reciprocal of Square Root

The Math Function block now supports a new function for computing the reciprocal of a square root: `1/sqrt`. You can use one block instead of two separate blocks for this computation, resulting in smaller block diagrams.

You can select one of two methods for computing the reciprocal of a square root: `Exact` or `Newton-Raphson`. Both methods support real input and output signals. When you use the `Newton-Raphson` method, you can also specify the number of iterations to perform the algorithm.

Math Function Block Enhancements for Real-Time Workshop Code Generation

The Math Function block now supports Real-Time Workshop code generation in these cases:

- Complex input and output signals for the `pow` function, for use with floating-point data types
- Fixed-point data types with fractional slope and nonzero bias for the `magnitude^2`, `square`, and `reciprocal` functions

Relational Operator Block Detects Signals That Are Infinite, NaN, or Finite

The Relational Operator block now includes `isInf`, `isNaN`, and `isFinite` functions to detect signals that are infinite, NaN, or finite. These new functions support real and complex input signals. If you select one of these functions, the block changes automatically to one-input mode.

Changes in Text and Visibility of Dialog Box Prompts for Easier Use with Fixed-Point Advisor and Fixed-Point Tool

The **Lock output scaling against changes by the autoscaling tool** check box is now **Lock output data type setting against changes by the fixed-point tools**. Previously, this check box was visible only if you entered an expression or a fixed-point data type for the output, such as `fixdt(1,16,0)`. This check box is now visible for any output data type specification. This enhancement helps you lock the current data type settings on a dialog box against changes that the Fixed-Point Advisor or Fixed-Point Tool chooses.

This enhancement applies to the following blocks:

- Abs
- Constant
- Data Store Memory
- Data Type Conversion
- Difference
- Discrete Derivative
- Discrete-Time Integrator
- Divide
- Dot Product
- Fixed-Point State-Space
- Gain
- Inport
- Lookup Table
- Lookup Table (2-D)
- Lookup Table Dynamic
- Math Function
- MinMax
- Multipoint Switch
- Outport
- Prelookup
- Product

- Product of Elements
- Relay
- Repeating Sequence Interpolated
- Repeating Sequence Stair
- Saturation
- Saturation Dynamic
- Signal Specification
- Switch

The **Lock scaling against changes by the autoscaling tool** check box is now **Lock data type settings against changes by the fixed-point tools**. Previously, this check box was visible only if you entered an expression or a fixed-point data type, such as `fixdt(1,16,0)`. This check box is now visible for any data type specification. This enhancement helps you lock the current data type settings on a dialog box against changes that the Fixed-Point Advisor or Fixed-Point Tool chooses.

This enhancement applies to the following blocks:

- Discrete FIR Filter
- Interpolation Using Prelookup
- Lookup Table (n-D)
- Sum
- Sum of Elements

Direct Lookup Table (n-D) Block Enhancements

The Direct Lookup Table (n-D) block now supports:

- Direct entry of **Number of table dimensions**
- Entry of **Table data** using the Lookup Table Editor

Previously, entering an integer greater than 4 for the **Number of table dimensions** required editing **Explicit number of table dimensions**. This extra parameter no longer appears on the block dialog box. For backward compatibility, scripts that contain `explicitNumDims` continue to work.

The other parameters for the block have changed as follows. For backward compatibility, the old command-line parameters continue to work.

Prompt on Block Dialog Box	Old Command-Line Parameter	New Command-Line Parameter
Number of table dimensions	<code>maskTabDims</code>	<code>NumberOfTableDimensions</code>
Inputs select this object from table	<code>outDims</code>	<code>InputsSelectThisObjectFromTable</code>
Make table an input	<code>tabIsInput</code>	<code>TableIsInput</code>
Table data	<code>mXTable</code>	<code>Table</code>
Action for out-of-range input	<code>clipFlag</code>	<code>ActionForOutOfRangeInput</code>
Sample time	<code>samptime</code>	<code>SampleTime</code>

The read-only **BlockType** parameter has also changed from S-Function to LookupNDDirect.

Version History

In R2009b, signal dimension propagation can behave differently from previous releases. Your model might not compile under these conditions:

- A Direct Lookup Table (n-D) block is in a source loop.
- Underspecified signal dimensions exist.

If your model does not compile, set dimensions explicitly for underspecified signals.

Unary Minus Block Enhancements

Conversion of the Unary Minus block from a masked S-Function to a core block enables more efficient simulation of the block.

You can now specify sample time for the block. The **Saturate to max or min when overflows occur** check box is now **Saturate on integer overflow**, and the command-line parameter is now `SaturateOnIntegerOverflow`. For backward compatibility, the old command-line parameter `DoSatur` continues to work.

The read-only **BlockType** parameter has also changed from S-Function to UnaryMinus.

Weighted Sample Time Block Enhancements

Conversions of the Weighted Sample Time and Weighted Sample Time Math blocks from masked S-Functions to core blocks enable more efficient simulation of the blocks.

The following parameter changes apply to both blocks. For backward compatibility, the old command-line parameters continue to work.

Old Prompt on Block Dialog Box	New Prompt on Block Dialog Box	Old Command-Line Parameter	New Command-Line Parameter
Output data type mode	Output data type	OutputDataType ScalingMode	OutDataTypeStr
Saturate to max or min when overflows occur	Saturate on integer overflow	DoSatur	SaturateOnIntegerOverflow

The read-only **BlockType** parameter has also changed from S-Function to SampleTimeMath.

Switch Case Block Parameter Change

For the Switch Case block, the command-line parameter for the **Show default case** check box is now `ShowDefaultCase`. For backward compatibility, the old command-line parameter `CaseShowDefault` continues to work.

Signal Conversion Block Parameter Change

For the Signal Conversion block, the parameter prompt for the **Override optimizations and always copy signal** check box is now **Exclude this block from 'Block reduction' optimization**.

Compare To Constant and Compare To Zero Blocks Use New Default Setting for Zero-Crossing Detection

The **Enable zero-crossing detection** parameter is now on by default for the Compare To Constant and Compare To Zero blocks. This change provides consistency with other blocks that support zero-crossing detection.

Signal Builder Block Change

You can no longer see the system under the Signal Builder block mask. In previous releases, you could right-click this block and select **Look Under Mask**.

In the Model Explorer, the Signal Builder block no longer appears in the Model Hierarchy view. In previous releases, this view was visible.

User Interface Enhancements

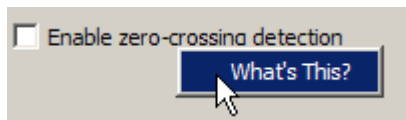
Context-Sensitive Help for Simulink Blocks in the Continuous Library

R2009b introduces context-sensitive help for parameters that appear in Simulink blocks of the Continuous library. This feature provides quick access to a detailed description of the block parameters.

To use the context-sensitive help:

- 1 Place your pointer over the label of a parameter and right-click.
- 2 A **What's This?** context menu appears.

For example, the following figure shows the **What's This?** context menu that appears after right-clicking the **Enable zero-crossing detection** parameter for the PID Controller block.



- 3 Click **What's This?** A window appears showing a description of the parameter.

Adding Blocks from a Most Frequently Used Blocks List

If you are using the same block repeatedly in a model, then you can save time by using the:

- **Most Frequently Used Blocks** tab in the Library Browser
- **Most Frequently Used Blocks** context menu option in the Model Editor

These features provide quick access to blocks you have added to models frequently. For details, see Adding Frequently Used Blocks.

Highlighting for Duplicate Inport Blocks

The **Highlight to Destination** option for a signal provides more information now for duplicate inport blocks. Applying this option to a signal of an inport block that has duplicate blocks highlights:

- The signal and destination block for that signal
- The signals and destination blocks of the duplicate blocks at the currently opened level in the model

Using the Model Explorer to Add a Simulink.NumericType Object

You can add a Simulink.NumericType object to the model workspace using the Model Explorer, provided you do not enable the **Is alias** option.

An example of when you might use this feature is when you:

- Want to define user-defined data types together in the model
- Do not need to preserve the data type name in the model or in the generated code

Block Output Display Dialog Has OK and Cancel Buttons

The **Block Output Display** dialog now includes **OK** and **Cancel** buttons to specify whether or not to apply your option settings.

Improved Definition of Hybrid Sample Time

Historically, you could not use the hybrid sample time to effectively identify a multirate subsystem or block. A subsystem was marked as “hybrid” and colored in yellow whether it contained two discrete sample times or one discrete sample time and one or more blocks with constant sample time [inf, 0]. Now, in R2009b, the check for the hybrid attribute no longer includes constant sample times, thereby improving the usefulness of the hybrid sample time color in identifying subsystems (and blocks) that are truly multirate.

Find Option in the Model Advisor

In R2009b, the Model Advisor includes a **Find** option to help you find checks. The find option, accessible through the **Edit** menu, allows you to find checks and folders more easily by searching names and analysis descriptions.

For more information, see Overview of the Model Advisor Window.

R2009a

Version: 7.3

New Features

Bug Fixes

Version History

Simulation Performance

Saving and Restoring the Complete SimState

Use the new SimState feature to save the complete simulation state. Unlike the final states stored in earlier versions of Simulink, the SimState contains the complete simulation state of the model (including block states that are logged). You can then restore the state at a later time and continue simulation from the exact instant at which you stopped the simulation.

Save Simulink Profiler Results

Save the results of the Simulink Profiler and later regenerate reports for review or for comparison.

Component-Based Modeling

Port Value Displays in Referenced Models

In R2009a, port value displays can appear for blocks in a Normal mode referenced model. To control port value displays, choose **View > Port Values** in the model window. For complete information about port value displays, see [Displaying Port Values](#).

Parallel Builds Enable Faster Diagram Updates for Large Model Reference Hierarchies In Accelerator Mode

R2009a provides potentially faster diagram updates for models containing large model reference hierarchies by building referenced models that are configured in Accelerator mode in parallel whenever possible. For example, updating of each model block can be distributed across the cores of a multicore host computer.

To take advantage of this feature, Parallel Computing Toolbox software must be licensed and installed in your development environment. If Parallel Computing Toolbox software is available, updating a model diagram rebuilds referenced models configured in Accelerator mode in parallel whenever possible.

For example, to use parallel building for updating a large model reference hierarchy on a desktop machine with four cores, you could perform the following steps:

- 1 Issue the MATLAB command `matlabpool 4` to set up a pool of four MATLAB workers, one for each core, in the Parallel Computing Toolbox environment.
- 2 Open your model and make sure that the referenced models are configured in Accelerator mode.
- 3 Optionally, inspect the model reference hierarchy. For example, you can use the Model Dependency Viewer from the **Tools** menu of Model Explorer to determine, based on model dependencies, which models will be built in parallel.
- 4 Update your model. Messages in the MATLAB command window record when each parallel or serial build starts and finishes.

The performance gain realized by using parallel builds for updating referenced models depends on several factors, including how many models can be built in parallel for a given model referencing hierarchy, the size of the referenced models, and host machine attributes such as amount of RAM and number of cores.

The following notes apply to using parallel builds for updating model reference hierarchies:

- Parallel builds of referenced models support only local MATLAB workers. They do not support remote workers in MATLAB Distributed Computing Server™ configurations.
- The host machine should have an appropriate amount of RAM available for supporting the number of local workers (MATLAB sessions) that you plan to use. For example, setting `matlabpool` to 4 results in five MATLAB sessions on your machine, each using approximately 120 MB of memory at startup.
- The same MATLAB environment must be set up in each MATLAB worker session as in the MATLAB client session — for example, the same base workspace variables, MATLAB path settings, and so forth. You can do this using the `PreLoadFcn` callback of the top model. Since the top

model is loaded with each MATLAB worker session, its preload function can be used for any MATLAB worker session setup.

Embedded MATLAB Function Blocks

Support for Enumerated Types

Embedded MATLAB Function blocks now support Simulink enumerated types and generate C code for enumerated data. See [Using Enumerated Data in MATLAB Function Blocks](#).

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

Embedded MATLAB Function blocks now use BLAS libraries to speed up low-level matrix operations during simulation. See [Speeding Up Simulation with the Basic Linear Algebra Subprograms \(BLAS\) Library](#).

Data Management

Signal Can Resolve to at Most One Signal Object

You can resolve a named signal to a signal object. The object can then specify or validate properties of the signal. For more information, see `Simulink.Signal`, Using Signal Objects to Initialize Signals and Discrete States, and Using Signal Objects to Tune Initial Values.

In previous releases, you could associate a signal with multiple signal objects, provided that the multiple objects specified compatible signal attributes. In R2009a, a signal can be associated with at most one signal object. The signal can reference the object more than once, but every reference must resolve to exactly the same object. A different signal object that has exactly the same properties will not meet the requirement. See Multiple Signal Objects for more information.

Version History

A compile-time error occurs in R2009a if a model associates more than one signal object with any signal. To prevent the error, decide which object the signal will use, and delete or reconfigure all references to any other signal objects so that all remaining references resolve to the chosen signal object. See Displaying Signal Sources and Destinations for a description of techniques that you can use to trace the full extent of a signal.

“Signed” Renamed to “Signedness” in the `Simulink.NumericType` class

In previous releases, the Property dialog of a `Simulink.NumericType` object whose **Data type mode** was any Fixed-point mode showed a property named **Signed**, which was a checkbox. Selecting the checkbox specified a signed type; clearing it specified an unsigned type. The API equivalent of **Signed** was `Signed`, a Boolean whose values could be 1 (signed) or 0 (unsigned).

In R2009a, a property named **Signedness** replaces **Signed** in the Property dialog of a `Simulink.NumericType` object. You can set **Signedness** to Signed (the default), Unsigned, or Auto, which specifies that the object inherits its **Signedness**. The API equivalent of **Signedness** is `Signedness`, which can be 1 (signed), 0 (unsigned), or Auto.

For compatibility with existing models, the property `Signed` remains available in R2009a. Setting `Signed` in R2009a sets `Signedness` accordingly. Accessing `Signed` in R2009a returns the value of `Signedness` if that value is 0 or 1, or generates an error if the value of `Signedness` is Auto, because that is not a legal value for `Signed`.

Do not use the `Signed` with `Simulink.NumericType` in new models; use `Signedness` instead. See `Simulink.NumericType` for more information.

“Sign” Renamed to “Signedness” in the Data Type Assistant

For blocks and classes that support fixed-point data types, the property **Sign** previously appeared in the Data Type Assistant when the **Mode** was Fixed point. In R2009a, this property appears in the Data Type Assistant as **Signedness**. Only the GUI label of the property differs; its behavior and API are unchanged in all contexts.

Tab Completion for Enumerated Data Types

Tab completion now works for enumerated data types in the same way that it does for other MATLAB classes. See [Instantiating Enumerations in MATLAB](#) for details.

Simulink File Management

Model Dependencies Tools

Enhanced file dependency analysis has the following new features:

- Files in the Simulink manifest are now recorded relative to a project root folder making manifests easier to share, compare and read. See [Generate Manifests](#) and [Edit Manifests](#).
- Command-line dependency analysis can now report toolbox dependencies, and when discovering file dependencies you can optionally generate a manifest file. See [Command-Line Dependency Analysis](#)

Block Enhancements

Prelookup and Interpolation Using Prelookup Blocks Support Parameter Data Types Different from Signal Data Types

The Prelookup block supports breakpoint data types that differ from input data types. This enhancement provides these benefits:

- Enables lower memory requirement for storing breakpoint data that uses a smaller type than the input signal
- Enables sharing of prescaled breakpoint data between two Prelookup blocks with different input data types
- Enables sharing of custom storage breakpoint data in generated code for blocks with different input data types

The Interpolation Using Prelookup block supports table data types that differ from output data types. This enhancement provides these benefits:

- Enables lower memory requirement for storing table data that uses a smaller type than the output signal
- Enables sharing of prescaled table data between two Interpolation Using Prelookup blocks with different output data types
- Enables sharing of custom storage table data in generated code for blocks with different output data types

The Interpolation Using Prelookup block also supports separate data type specification for intermediate results. This enhancement enables use of a greater precision for internal computations than for table data or output data.

Lookup Table (n-D) and Interpolation Using Prelookup Blocks Perform Efficient Fixed-Point Interpolations

Whenever possible, Lookup Table (n-D) and Interpolation Using Prelookup blocks use a faster overflow-free subtraction algorithm for fixed-point interpolation. To achieve this efficiency, the blocks use a data type of larger container size to perform the overflow-free subtraction, instead of using control-flow branches as in previous releases. Also, the generated code for fixed-point interpolation is now smaller.

Version History

Due to the change in the overflow-free subtraction algorithm, fixed-point interpolation in Lookup Table (n-D) and Interpolation Using Prelookup blocks might, in a few cases, introduce different rounding results from previous releases. Both simulation and code generation use the new overflow-free algorithm, so they have the same rounding behavior and provide bit-true consistency.

Expanded Support for Simplest Rounding Mode to Maximize Block Efficiency

In R2009a, support for the Simplest rounding mode has been expanded to enable more blocks to handle mixed floating-point and fixed-point data types:

- Abs
- Data Type Conversion Inherited
- Difference
- Discrete Derivative
- Discrete FIR Filter
- Discrete-Time Integrator
- Dot Product
- Fixed-Point State-Space
- Gain
- Index Vector
- Lookup Table (n-D)
- Math Function (for the `magnitude^2`, `reciprocal`, `square`, and `sqrt` functions)
- MinMax
- Multiport Switch
- Saturation
- Saturation Dynamic
- Sum
- Switch
- Transfer Fcn Direct Form II
- Transfer Fcn Direct Form II Time Varying
- Transfer Fcn First Order
- Transfer Fcn Lead or Lag
- Transfer Fcn Real Zero
- Weighted Sample Time
- Weighted Sample Time Math

For more information, see [Rounding Mode: Simplest](#).

New Rounding Modes Added to Multiple Blocks

For the following Simulink blocks, the dialog box now displays `Convergent` and `Round` as possible rounding modes. These modes enable numerical agreement with embedded hardware and MATLAB results.

- Abs
- Data Type Conversion
- Data Type Conversion Inherited

- Difference
- Discrete Derivative
- Discrete FIR Filter
- Discrete-Time Integrator
- Divide
- Dot Product
- Fixed-Point State-Space
- Gain
- Index Vector
- Interpolation Using Prelookup
- Lookup Table
- Lookup Table (2-D)
- Lookup Table (n-D)
- Lookup Table Dynamic
- Math Function (for the magnitude^2 , reciprocal , square , and sqrt functions)
- MinMax
- Multiport Switch
- Prelookup
- Product
- Product of Elements
- Saturation
- Saturation Dynamic
- Sum
- Switch
- Transfer Fcn Direct Form II
- Transfer Fcn Direct Form II Time Varying
- Transfer Fcn First Order
- Transfer Fcn Lead or Lag
- Transfer Fcn Real Zero
- Weighted Sample Time
- Weighted Sample Time Math

In the dialog box for these blocks, the field **Round integer calculations toward** has been renamed **Integer rounding mode**. The command-line parameter remains the same.

For more information, see [Rounding Mode: Convergent](#) and [Rounding Mode: Round](#) in the Fixed-Point Toolbox™ documentation.

Version History

If you use an earlier version of Simulink software to open a model that uses the **Convergent** or **Round** rounding mode, the mode changes automatically to **Nearest**.

Lookup Table (n-D) Block Performs Faster Calculation of Index and Fraction for Power of 2 Evenly-Spaced Breakpoint Data

For power of 2 evenly-spaced breakpoint data, the Lookup Table (n-D) block uses bit shifts to calculate the index and fraction, instead of division. This enhancement provides these benefits:

- Faster calculation of index and fraction for power of 2 evenly-spaced breakpoint data
- Smaller size of generated code for the Lookup Table (n-D) block

Discrete FIR Filter Block Supports More Filter Structures

The following filter structures have been added to the Discrete FIR Filter block:

- Direct form symmetric
- Direct form antisymmetric
- Direct form transposed
- Lattice MA

Running a model with these filter structures requires a Signal Processing Blockset license.

Discrete Filter Block Performance, Data Type, Dimension, and Complexity Enhancements

The following enhancements have been made to the Discrete Filter block:

- Improved numerics and run-time performance of outputs and states by reducing the number of divide operations in the filter to at most one
- Support for signed fixed-point and integer data types
- Support for vector and matrix inputs
- Support for complex inputs and filter coefficients, where inputs and coefficients can each be real or complex, independently of the other
- A new **Initial states** parameter allows you to enter non-zero initial states
- A new **Leading denominator coefficient equals 1** parameter provides a more efficient implementation by eliminating all divides when the leading denominator coefficient is one

Version History

Due to these enhancements, you might encounter the compatibility issues in the following sections.

Realization parameter removed. The Real-Time Workshop software `realization` parameter has been removed from this block. You can no longer use the `set_param` and `get_param` functions on this block parameter. The generated code for this block has been improved to be similar to the former 'sparse' realization, while maintaining tunability as in the former 'general' realization.

State changes. Due to the reduction in the number of divide operations performed by the block, you might notice that your logged states have changed when the leading denominator coefficient is not one.

MinMax Block Performs More Efficient and Accurate Comparison Operations

For multiple inputs with mixed floating-point and fixed-point data types, the MinMax block selects an appropriate data type for performing comparison operations, instead of using the output data type for all comparisons, as in previous releases. This enhancement provides these benefits:

- Faster comparison operations, with fewer fixed-point overflows
- Smaller size of generated code for the MinMax block

Logical Operator Block Supports NXOR Boolean Operator

In R2009a, the Logical Operator block has been enhanced with a new NXOR Boolean operator. When you select this operator, the block returns TRUE when an even number of inputs are TRUE. Similarly, the block returns FALSE when an even number of inputs are FALSE.

Use NXOR to replace serial XOR and NOT operations in a model.

Discrete-Time Integrator Block Uses Efficient Integration-Limiting Algorithm for Forward Euler Method

When you select the **Limit output** check box for the Forward Euler method, the Discrete-Time Integrator block uses only one saturation when a second saturation is unnecessary. This change in the integration-limiting algorithm provides these benefits:

- Faster integration
- Smaller size of generated code for the Discrete-Time Integrator block

Dot Product Block Converted from S-Function to Core Block

Conversion of the Dot Product block from a masked S-Function to a core block enables more efficient simulation and better handling of the block in Simulink models.

Due to this conversion, you can specify sample time and values for the output minimum and maximum for the Dot Product block. The read-only **BlockType** parameter has also changed from S-Function to DotProduct.

Version History

In R2009a, signal dimension propagation might behave differently from previous releases. As a result, your model might not compile under these conditions:

- Your model contains a Dot Product block in a source loop.
- Your model has underspecified signal dimensions.

If your model does not compile, set dimensions for signals that are not fully specified.

For example, your model might not compile in this case:

- Your model contains a Transfer Fcn Direct Form II Time Varying block, which is a masked S-Function with a Dot Product block in a source loop.

- The second and third input ports of the Transfer Fcn Direct Form II Time Varying block are unconnected, which results in underspecified signal dimensions.

To ensure that your model compiles in this case, connect Constant blocks to the second and third input ports of the Transfer Fcn Direct Form II Time Varying block and specify the signal dimensions for both ports explicitly.

Pulse Generator Block Uses New Default Values for Period and Pulse Width

For the Pulse Generator block, the default **Period** value has changed from 2 to 10, and the default **Pulse Width** value has changed from 50 to 5. These changes enable easier transitions between time-based and sample-based mode for the pulse type.

Random Number, Uniform Random Number, and Unit Delay Blocks Use New Default Values for Sample Time

The default **Sample time** values for the Random Number, Uniform Random Number, and Unit Delay blocks have changed:

- The default **Sample time** value for the Random Number and Uniform Random Number blocks has changed from 0 to 0.1.
- The default **Sample time** value for the Unit Delay block has changed from 1 to -1.

Trigonometric Function Block Provides Better Support of Accelerator Mode

The Trigonometric Function block now supports Accelerator mode for all cases with real inputs and Normal mode support. For more information about simulation modes, see Accelerating Models in the Simulink User's Guide.

Reshape Block Enhanced with New Input Port

The Reshape block **Output dimensionality** parameter has a new option, **Derive from reference input port**. This option creates a second input port, **Ref**, on the block and derives the dimensions of the output signal from the dimensions of the signal input to the **Ref** input port. Similarly, the Reshape block command-line parameter, **OutputDimensionality**, has the new option, **Derive from reference input port**.

Multidimensional Signals in Simulink Blocks

The following blocks were updated to support multidimensional signals. For more information, see Signal Dimensions in the Simulink User's Guide.

- Assertion
- Extract Bits
- Check Discrete Gradient
- Check Dynamic Gap

- Check Dynamic Lower Bound
- Check Dynamic Range
- Check Dynamic Upper Bound
- Check Input Resolution
- Check Static Gap
- Check Static Lower Bound
- Check Static Range
- Check Static Upper Bound
- Data Type Scaling Strip
- Wrap to Zero

Subsystem Blocks Enhanced with Read-Only Property That Indicates Virtual Status

The following subsystem blocks now have the property, `IsSubsystemVirtual`. This read-only property returns a Boolean value, `on` or `off`, to indicate if a subsystem is virtual.

- Atomic Subsystem
- Code Reuse Subsystem
- Configurable Subsystem
- Enabled and Triggered Subsystem
- Enabled Subsystem
- For Iterator Subsystem
- Function-Call Subsystem
- If Action Subsystem
- Subsystem
- Switch Case Action Subsystem
- Triggered Subsystem
- While Iterator Subsystem

User Interface Enhancements

Port Value Displays in Referenced Models

In R2009a, port value displays can appear for blocks in a Normal mode referenced model. To control port value displays, choose **View > Port Values** in the model window. For complete information about port value displays, see [Displaying Port Values](#).

Print Sample Time Legend

Print the Sample Time Legend either as an option of the block diagram print dialog box or directly from the legend. In either case, the legend will print on a separate sheet of paper. For more information, see [Print Sample Time Legend](#).

M-API for Access to Compiled Sample Time Information

New MATLAB API provides access to the compiled sample time data, color, and annotations for a specific block or the entire block diagram directly from M code.

Model Advisor Report Enhancements

In R2009a, the Model Advisor report is enhanced with:

- The ability to save the report to a location that you specify.
- Improved readability, including the ability to:
 - Filter the report to view results according to the result status. For example, you can now filter the report to show errors and warnings only.
 - Collapse and expand the folder view in the report.
 - View a summary of results for each folder in the report.

See [Consulting the Model Advisor in the Simulink User's Guide](#).

Counterclockwise Block Rotation

This release lets you rotate blocks counterclockwise as well as clockwise (see [How to Rotate a Block](#) for more information).

Physical Port Rotation for Masked Blocks

This release lets you specify that the ports of a masked block not be repositioned after a clockwise rotation to maintain a left-to-right and top-to-bottom numbering of the ports. This enhancement facilitates use of masked blocks in mechanical systems, hydraulic systems, and other modeling applications where block diagrams do not have a preferred orientation (see [Port Rotation Type](#) for more information.)

Smart Guides

In R2009a, when you drag a block, Simulink draws lines, called smart guides, that indicate when the block's ports, center, and edges align with the ports, centers, and edges of other blocks in the same diagram. This helps you create well-laid-out diagrams (see Smart Guides for more information).

Customizing the Library Browser's User Interface

Release 2009a lets you customize the Library Browser's user interface. You can change the order in which libraries appear in the Library Browser, disable or hide libraries, sublibraries, and blocks, and add, disable, or hide items on the Library Browser's menus. See Customizing the Library Browser for more information.

Subsystem Creation Command

This release adds a command, `Simulink.BlockDiagram.createSubSystem`, that creates a subsystem from a specified group of blocks.

Removal of Lookup Table Designer from the Lookup Table Editor

In R2009a, the Lookup Table Designer is no longer available in the Lookup Table Editor.

Version History

Previously, you could select **Edit > Design Table** in the Lookup Table Editor to launch the Lookup Table Designer. In R2009a, this menu item is no longer available.

S-Functions

Level-1 Fortran S-Functions

In this release, if you attempt to compile or simulate a model with a Level-1 Fortran S-function, you will receive an error due to the use of the newly deprecated function 'MXCREATEFULL' within the Fortran S-function wrapper 'simulink.F'. If your S-function does not explicitly use 'MXCREATEFULL', simply recompile the S-function. If your S-function uses 'MXCREATEFULL', replace each instance with 'MXCREATEDOUBLEMATRIX' and recompile the S-function.

R2008b

Version: 7.2

New Features

Bug Fixes

Version History

Simulation Performance

Parallel Simulations in Rapid Accelerator Mode

Simulink now has the capability to run parallel simulations in Rapid Accelerator mode using `parfor` on prebuilt Simulink models.

You can now run parallel simulations in Rapid Accelerator mode with different external inputs and tunable parameters. The `sim` command can be called from a `parfor` loop if the model does not require a rebuild.

For more information, see [Running a Simulation Programmatically](#).

Improved Rebuild Mechanism in Rapid Accelerator Mode

Simulink now has enhanced tuning of the solver and logging parameters in Rapid Accelerator mode without requiring a rebuild.

An improved rebuild mechanism ensures that the model does not rebuild when you change block diagram parameters (e.g., stop time, solver tolerances, etc.). This enhancement significantly decreases the time for simulation in Rapid Accelerator mode.

Data Type Size Limit on Accelerated Simulation Removed

In previous releases, accelerated simulation was not supported for models that use integer or fixed-point data types greater than 32 bits in length. In this release, the acceleration limit on integer and fixed-point data type size has increased to 128 bits, the same as the limit for normal-mode, i.e., unaccelerated simulation.

New Initialization Behavior in Conditional, Action, and Iterator Subsystems

For releases prior to 2008b, at the simulation start time, Simulink initializes all blocks unconditionally and subsystems cannot reset the states. Release 2008b introduces behavior that mirrors the behavior of Real-Time Workshop. For normal simulation mode, the Simulink block initialization method (`mdlInitializeConditions`) can be called more than once at the start time if:

- The block is contained within a Conditional, Action, or Iterator subsystem.
- The subsystem is configured to reset states when enabled (or triggered); and the subsystem is enabled (or triggered) at the start time.

This new initialization behavior has the following effect on S-functions:

- If you need to ensure that the initialization code in the `mdlInitializeConditions` function runs only once, then move this initialization code into the `mdlStart` method. MathWorks recommends this code change as a best practice.
- The change to the block initialization method, as described above, exposed a bug in the S-function macro `ssIsFirstInitCond` for applications involving an S-function within a Conditional, Action or Iterator subsystem. This bug has been fixed in R2008b.

To determine if you consequently need to update your Simulink S-functions for compatibility, compare the simulation results from R2007b or an earlier release with those of R2008b. If they differ at the start time, `ssIsFirstInitCond` is running more than once and you must regenerate and recompile the appropriate Simulink S-functions.

For Real-Time Workshop, you must regenerate and recompile all S-function targets and any Real-Time Workshop target for which the absolute time is turned on. (If a third-party vendor developed your S-functions, have the vendor regenerate and recompile them for you. The vendor can use the `SLDiagnostics` feature to identify all S-functions in a model.)

Component-Based Modeling

Processor-in-the-Loop Mode in Model Block

In R2008b, Simulink has a new Model block simulation mode for processor-in-the-loop (PIL) verification of generated code. This feature requires Real-Time Workshop Embedded Coder software. The feature lets you test the automatically generated and cross-compiled object code on your embedded processor by easily switching between Normal, Accelerator, and PIL simulation modes in your original model. You can reuse test suites, resulting in faster iteration between model development and generated code verification. For more information, see Referenced Model Simulation Modes.

Conditionally Executed Subsystem Initial Conditions

R2008b of Simulink includes enhanced handling of initial conditions for conditionally executed subsystems, Merge blocks, and Discrete-Time Integrator blocks, improving consistency of simulation results.

This feature allows you to select simplified initialization mode for conditionally executed subsystems, Merge blocks, subsystem elapsed time, and Discrete-Time Integrator blocks. The simplified initialization improves the consistency of simulation results, especially for models that do not specify initial conditions for conditional subsystem output ports, and for models that have conditionally executed subsystem output ports connected to S-functions.

Note To use the new simplified initialization mode, you must activate this feature.

Activating This Feature for New Models

For new models, you can activate this feature as follows:

- 1 In the model window, select **Simulation > Configuration Parameters**.
The Configuration Parameters dialog box opens.
- 2 Select **Diagnostics > Data Validity**.
The Data Validity Diagnostics pane opens.
- 3 In the Model Initialization section, set **Underspecified initialization detection** to **Simplified**.
- 4 Select **Diagnostics > Connectivity**.
The Connectivity Diagnostics pane opens.
- 5 Set **Mux blocks used to create bus signals** to error.
- 6 Set **Bus signal treated as vector** to error.
- 7 Click **OK**.

For more information, see Underspecified initialization detection.

Migrating Existing Models

For existing models, MathWorks recommends using the Model Advisor to migrate your model to the new simplified initialization mode settings.

To migrate an existing model:

- 1** In the model window, select **Simulation > Configuration Parameters**.
The Configuration Parameters dialog box opens.
- 2** Select **Diagnostics > Data Validity**.
The Data Validity Diagnostics pane opens.
- 3** In the Merge Block section, set **Detect multiple driving blocks executing at the same time step** to error.
- 4** Click **OK**.
- 5** Simulate the model and ensure that it runs without errors.
- 6** Select **Tools > Model Advisor**.
The Model Advisor opens.
- 7** In the Model Advisor Task Manager, select **By Product > Simulink**.
- 8** Run **Check bus usage** in the Model Advisor.
- 9** Run **Check consistency of initialization parameters for Outport and Merge blocks** in the Model Advisor.
- 10** After you have resolved any errors identified by this check, click **Proceed** to migrate your model to simplified initialization mode.

For information on using the Model Advisor, see Consulting the Model Advisor in the Simulink User's Guide.

For information on the Model Advisor checks, see Check consistency of initialization parameters for Outport and Merge blocks.

Version History

Activating this feature can cause differences in simulation results, when compared to previous versions. Since you must opt-in to this feature before any changes are made, there are no issues for existing models. However, MathWorks recommends that you backup existing models before you migrate them, in case you want to return to the original behavior.

Model Block Input Enhancement

Model block inputs can now be local and reusable. This capability reduces global data usage and data copying when interfacing with code from a referenced model, which can reduce memory usage during simulation and increase the efficiency of generated code. This enhancement is always relevant, so no configuration parameter is necessary or provided to control it.

One Parameter Controls Accelerator Mode Build Verbosity

In previous releases, the `ModelReferenceSimTargetVerbose` parameter controlled verbosity when a referenced model was built for execution in Accelerator mode, as specified by the Model

block's Simulation mode parameter. The `ModelReferenceSimTargetVerbose` had no GUI equivalent. See Referenced Model Simulation Modes and the Model block documentation for more information.

A different parameter, `AccelVerboseBuild`, controls the verbosity when a model is built in Simulink Accelerator mode or Rapid Accelerator mode, as specified in the **Simulation** menu. See Accelerating Models for more information. The GUI equivalent of the `AccelVerboseBuild` parameter is **Configuration Parameters > Optimization > Verbose accelerator builds**. See Verbose accelerator builds for more information.

All types of accelerated simulation entail code generation (though the code is not visible to the user) and the two verbosity parameters control whether a detailed account of the code generation process appears in the MATLAB Command Window. However, providing separate verbosity parameters for the two cases was unnecessary.

In R2008b, the `ModelReferenceSimTargetVerbose` parameter is deprecated and has no effect. The `AccelVerboseBuild` parameter (**Configuration Parameters > Optimization > Verbose accelerator builds**) now controls the verbosity for Simulink Accelerator mode, referenced model Accelerator mode, and Rapid Accelerator mode.

Another parameter, `RTWVerbose` (**Configuration Parameters > Real-Time Workshop > Debug > Verbose build**) controls the verbosity of Real-Time Workshop code generation. This parameter is unaffected by the changes to `ModelReferenceSimTargetVerbose` and `AccelVerboseBuild`.

Version History

In R2008b, trying to set `ModelReferenceSimTargetVerbose` generates a warning message and has no effect on verbosity. The warning says to use `AccelVerboseBuild` instead. The default for `AccelVerboseBuild` is 'off'.

A model saved in R2008b will not include the `ModelReferenceSimTargetVerbose` parameter. An R2008b model saved to an earlier Simulink version that supports `ModelReferenceSimTargetVerbose` will include that parameter, giving it the same value that `AccelVerboseBuild` has in the R2008b version.

The effect of loading a model from an earlier Simulink version into R2008b depends on the source version:

- **Prior to R14:** Neither parameter exists, so no compatibility consideration arises.
- **R14 - R2006b:** Only `ModelReferenceSimTargetVerbose` exists. Copy its value to `AccelVerboseBuild`.
- **R2007a:** Both parameters exist but neither has a GUI equivalent. Ignore the value of `ModelReferenceSimTargetVerbose` and post no warning.
- **R2007b - R2008a:** Both parameters exist and `AccelVerboseBuild` has a GUI equivalent. If `ModelReferenceSimTargetVerbose` is 'on', post a warning to use `AccelVerboseBuild` instead.

Embedded MATLAB Function Blocks

Support for Fixed-Point Word Lengths Up to 128 Bits

Embedded MATLAB Function blocks now support up to 128 bits of fixed-point precision. This increase in maximum precision from 32 to 128 bits supports generating efficient code for targets with non-standard word sizes and allows Embedded MATLAB Function blocks to work with large fixed-point signals.

Enhanced Simulation and Code Generation Options for Embedded MATLAB Function Blocks

You can now specify embeddable code generation options from the Embedded MATLAB Editor using a new menu item: **Tools > Open RTW Target**. Simulation options continue to be available from **Tools > Open Simulation Target**.

In addition, simulation and embeddable code generation options now appear in a single dialog box. For details, see “Unified Simulation and Embeddable Code Generation Options” on page 32-17.

Data Type Override Now Works Consistently on Outputs

When you enable data type override for Embedded MATLAB Function blocks, outputs with explicit and inherited types are converted to the override type. For example, if you set data type override to `true singles`, the Embedded MATLAB Function block converts all outputs to `single` type and propagates the override type to downstream blocks.

In previous releases, Embedded MATLAB Function blocks did not apply data type override to outputs with inherited types. Instead, the inherited type was preserved even if it did not match the override type, sometimes causing errors during simulation.

Version History

Applying data type override rules to outputs with inherited types may introduce the following compatibility issues:

- Downstream Embedded MATLAB Function blocks must be able to accept the propagated override type. Therefore, you must allow data type override for downstream blocks for which you set output type explicitly. Otherwise, you may not be able to simulate your model.
- You might get unexpected simulation results if the propagated type uses less precision than the original type.

Improperly-Scaled Fixed-Point Relational Operators Now Match MATLAB Results

When evaluating relational operators, Embedded MATLAB Function blocks compute a common type that encompasses both input operands. In previous releases, if the common type required more than 32 bits, Embedded MATLAB Function blocks may have given different answers from MATLAB. Now, Embedded MATLAB Function blocks give the same answers as MATLAB.

Version History

Some relational operators generate multi-word code even if one of the fixed-point operands is not a multi-word value. To work around this issue, cast both operands to the same Fixed-Point Toolbox type (using the same scaling method and properties).

Data Management

Support for Enumerated Data Types

Simulink models now support enumerated data types. For details, see:

- Enumerations and Modeling
- Using Enumerated Data in Stateflow Charts in the Stateflow documentation
- Enumerations in the Real-Time Workshop documentation

Simulink Bus Editor Enhancements

The Simulink Bus Editor can now filter displayed bus objects by either name or relationship. See [Filtering Displayed Bus Objects](#) for details.

You can now fully customize the export and import capabilities of the Simulink Bus Editor. See [Customizing Bus Object Import and Export](#) for details.

New Model Advisor Check for Data Store Memory Usage

A new Model Advisor check posts advice and warnings about the use of Data Store Memory, Data Store Read, and Data Store Write blocks. See [Check Data Store Memory blocks for multitasking, strong typing, and shadowing issues](#) for details.

Simulink File Management

Model Dependencies Tools

Enhanced file dependency analysis can now:

- Find system target files
- Analyze STF_make_rtw_hook functions
- Analyze all configuration sets, not just the active set.

See Scope of Dependency Analysis in the Simulink User's Guide.

Block Enhancements

Trigonometric Function Block

R2008b provides an enhanced Trigonometric Function block to:

- Support sincos
- Provide greater floating-point consistency

Math Function Block

In Simulink 2008b, an enhanced Math Function block provides greater floating-point consistency.

Merge Block

R2008b provides enhanced handling of initial conditions for the Merge block and thus improves the consistency of simulation results.

For more information, see “Conditionally Executed Subsystem Initial Conditions” on page 32-4.

Discrete-Time Integrator Block

R2008b provides an enhanced handling of initial conditions for the Discrete-Time Integrator block and thereby improves the consistency of simulation results.

For more information, see “Conditionally Executed Subsystem Initial Conditions” on page 32-4.

Modifying a Link to a Library Block in a Callback Function Can Cause Illegal Modification Errors

In this release, Simulink software can signal an error if a block callback function, e.g., `CopyFcn`, modifies a link to a library block. For example, an error occurs if you attempt to copy a library link to a self-modifying masked subsystem whose `CopyFcn` deletes a block contained by the subsystem. This change means that you cannot use block callback functions to create self-modifying library blocks. Mask initialization code for a library block is the only code allowed to modify the block.

Version History

Previous releases allowed use of block callback functions to create self-modifying library blocks. Opening, editing, or running models that contain links to such blocks can cause illegal modification errors in the current release. As a temporary work around, you can break any links in your model to a library block that uses callback functions to modify itself. The best long-term solution is to move the self-modification code to the block's mask initialization section.

Random Number Block

In the dialog box for the Random Number block, the field **Initial Seed** has been renamed **Seed**. The command-line parameter remains the same.

Signal Generator Block

The Signal Generator block now supports multidimensional signals. For a list of blocks that support multidimensional signals, see Signal Dimensions in the Simulink User's Guide.

Sum Block

The accumulator of the Sum block now applies for all input signals of any data type (for example, double, single, integer, and fixed-point). In previous releases, the accumulator of this block was limited to inputs and outputs of only integer or fixed-point data types.

Switch Block

The Switch block now supports the immediate back propagation of a known output data type to the first and third input ports. This occurs when you set the **Output data type** parameter to `Inherit: Inherit via internal rule` and select the **Require all data port inputs to have the same data type** check box. In previous releases, this back propagation did not occur immediately.

Uniform Random Number Block

In the dialog box for the Uniform Random Number block, the field **Initial Seed** has been renamed **Seed**. The command-line parameter remains the same.

User Interface Enhancements

Sample Time

The display of sample time information has been expanded to include:

- Signal lines labeling with new color-independent **Annotations**
- A new **Sample Time Legend** maps the sample time **Colors** and **Annotations** to sample times.
- A distinct color for indicating that a block and signal are asynchronous.

The section “Modeling and Simulation of Discrete Systems” has been renamed “Working with Sample Times” and has been significantly expanded to provide a comprehensive review of sample times and a discussion on the new Sample Time Legend and Sample Time Display features. For more information, see Working with Sample Times.

Model Advisor

In R2008b, the Model Advisor is enhanced with:

- A model and data restore point that provides you with the ability to revert changes made in response to advice from the Model Advisor
- Context-sensitive help available for Model Advisor checks
- Tristate check boxes that visually indicate selected and cleared checks in folders
- A system selector for choosing the system level that the Model Advisor checks

See Consulting the Model Advisor in the Simulink User's Guide.

“What’s This?” Context-Sensitive Help for Commonly Used Blocks

R2008b introduces context-sensitive help for parameters that appear in the following commonly used blocks in Simulink:

Bus Creator
 Bus Selector
 Constant
 Data Type Conversion
 Demux
 Discrete-Time Integrator
 Gain
 Inport
 Integrator
 Logical Operator
 Mux
 Outport
 Product
 Relational Operator
 Saturation
 Subsystem
 Sum
 Switch

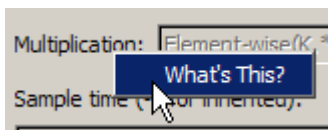
Terminator
Unit Delay

This feature provides quick access to a detailed description of the parameters, saving you the time it would take to find the information in the Help browser.

To use the "What's This?" help, do the following:

- 1 Place your cursor over the label of a parameter.
- 2 Right-click. A **What's This?** context menu appears.

For example, the following figure shows the **What's This?** context menu appearing after right-clicking the **Multiplication** parameter for the Gain block.



- 3 Click **What's This?** A context-sensitive help window appears showing a description of the parameter.

Compact Icon Option Displays More Blocks in Library Browser

This release introduces a compact icon option that maximizes the number of blocks and libraries visible in the Library Browser's **Library** pane without scrolling (see Library Pane).

Signal Logging and Test Points Are Controlled Independently

In previous releases, a signal could be logged only if it was also a test point. Therefore, selecting **Log signal data** in the Signal Properties dialog box automatically selected **Test point**, and disabled it so that it could not be cleared. However, a signal can be a test point without being logged, so clearing **Log signal data** did not automatically clear **Test point**. The same asymmetric behavior occurred programmatically with the underlying `DataLogging` and `TestPoint` parameters.

In R2008b, no connection exists between enabling logging for a signal and making the signal a test point. Either, both, or neither capability can be enabled for any signal. Selecting and clearing **Log signal data** therefore has no effect on the setting of **Test point**, and similarly for the underlying parameters. See [Exporting Signal Data Using Signal Logging](#) and [Working with Test Points](#) for more information.

To reflect the independence of logging and test points, the command **Test Point Indicators** in the Simulink **Format > Port/Signal Displays** menu has been renamed **Testpoint/Logging Indicators**. The effect of the command, the graphical indicators displayed, and the meaning of the underlying parameter `ShowTestPointIcons`, are all unchanged.

Version History

Scripts and practices that relied on **Log signal data** to automatically set a test point must be changed to set the test point explicitly. The relevant `set_param` commands are:

```
set_param(PortHandle(n), 'DataLogging', 'on')
set_param(PortHandle(n), 'TestPoint', 'on')
```


To disable either capability, set the relevant parameter to 'off'. See Enabling Logging for a Signal for an example.

Signal Logging Consistently Retains Duplicate Signal Regions

A *virtual signal* is a signal that graphically represents other signals or parts of other signals. Virtual signals are purely graphical entities; they have no functional or mathematical significance. The nonvirtual components of a virtual signal are called *regions*. For example, if Mux block (which is a virtual block) inputs two nonvirtual signals, the block outputs a virtual signal that has two regions. See Virtual Signals and Mux Signals for more information.

In previous releases, when a virtual signal contains duplicate regions, signal logging excluded all but one of the duplicates in some contexts, but included all of the duplicates in other contexts, giving inconsistent results. For example, if the same nonvirtual signal is connected to two input ports of a Mux block, that one signal is the source of two regions in the Mux block output. Previously, if that output was being logged in Normal mode simulation, the log object would contain data for only one of the regions, because the other was eliminated as a duplicate.

In R2008a, Simulink no longer eliminates duplicate regions when logging the output of virtual blocks like Mux or Selector blocks. Simulink now logs all regions, which appear in a `Simulink.TsArray` object. The duplicate regions have unique names as follows:

```
<signal_name>_reg<#counter>
```

This change affects signal logs and all capabilities that depend on signal logging, such as scopes and signal viewers.

Version History

In cases where signal logging previously omitted duplicate regions, signal logs will now be larger, and scopes and signal viewers will now show more data. This change could give the impression that the results of simulation have changed, but actually only the logging of those results has changed. No action is needed unless:

- A dependency exists on the exact size of a log or the details of its contents.
- The size and details have changed due to the inclusion of previously omitted signals.

In such a case, make changes as needed to accept the changed logging behavior. See Exporting Signal Data Using Signal Logging for more information.

Simulink Configuration Parameters

In R2008b, the following Simulink configuration parameters are updated:

Note The command-line parameter name is not changing for these parameters.

Location	Previous Parameter	New Parameter
Solver	States shape preservation / ShapePreserveControl	Shape preservation / ShapePreserveControl

Location	Previous Parameter	New Parameter
Solver	Consecutive min step size violations / MaxConsecutiveMinStep	Number of consecutive min steps / MaxConsecutiveMinStep
Solver	Consecutive zero crossings relative tolerance / ConsecutiveZCsStepRelTol	Time tolerance / ConsecutiveZCsStepRelTol
Solver	Zero crossing location algorithm / ZeroCrosAlgorithm	Algorithm / ZeroCrosAlgorithm
Solver	Zero crossing location threshold / ZCThreshold	Signal threshold/ ZCThreshold
Solver	Number of consecutive zero crossings allowed / MaxConsecutiveZCs	Number of consecutive zero crossings / MaxConsecutiveZCs
Optimization	Eliminate superfluous temporary variables (Expression folding) / ExpressionFolding	Eliminate superfluous local variables (Expression folding) / ExpressionFolding
Optimization	Remove internal state zero initialization / ZeroInternalMemoryAtStartup	Remove internal data zero initialization / ZeroInternalMemoryAtStartup

In R2008b, the following Simulink configuration parameters have moved:

Note The command-line parameter name is not changing for these parameters.

Parameter	Old Location	New Location
Check undefined subsystem initial output	Diagnostics > Compatibility	Diagnostics > Data Validity
Check preactivation output of execution context	Diagnostics > Compatibility	Diagnostics > Data Validity
Check runtime output of execution context	Diagnostics > Compatibility	Diagnostics > Data Validity

In R2008b, the **Optimization > Minimize array reads using temporary variables** parameter has been obsolete.

Model Help Menu Update

The Simulink model **Help** menu now includes links to block support tables for the following products, if they are installed.

- Simulink
- Communications Blockset
- Signal Processing Blockset
- Video and Image Processing Blockset

To obtain the block support tables for all of these products that are installed, select **Help > Block Support Table > All Tables**.

In previous releases, **Help > Block Support Table** provided such tables only for the main Simulink library.

Unified Simulation and Embeddable Code Generation Options

You can now specify both simulation and embeddable code generation options in the Configuration Parameters dialog box. The simulation options apply only to Embedded MATLAB Function blocks, Stateflow charts, and Truth Table blocks.

The following table summarizes changes that apply for Embedded MATLAB Function blocks:

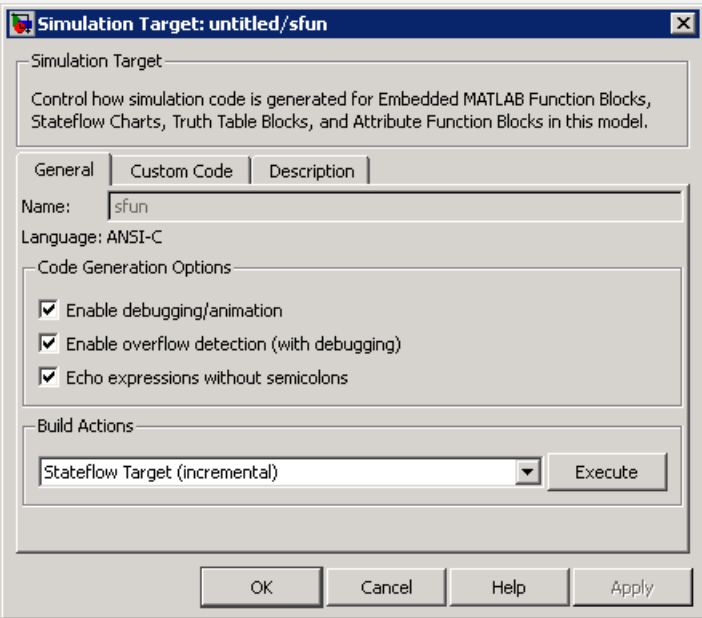
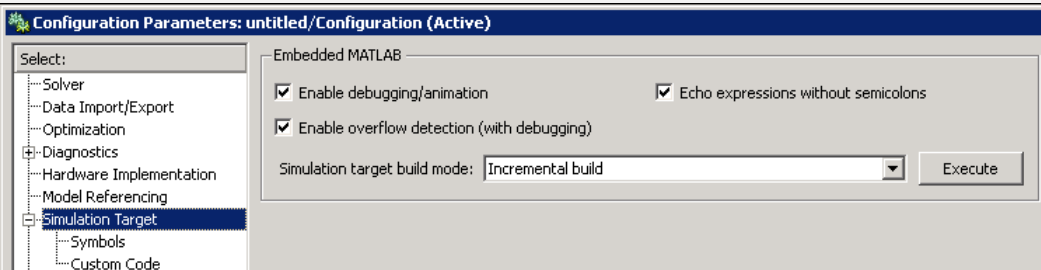
Type of Model	Simulation Options	Embeddable Code Generation Options
Nonlibrary	<p>Migrated from the Simulation Target dialog box to the Configuration Parameters dialog box.</p> <p>See:</p> <ul style="list-style-type: none"> • “Nonlibrary Models: Changes for the General Pane of the Simulation Target Dialog Box” on page 32-18 • “Nonlibrary Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box” on page 32-19 • “Nonlibrary Models: Changes for the Description Pane of the Simulation Target Dialog Box” on page 32-20 	<p>New menu item in the Embedded MATLAB Editor for specifying code generation options for nonlibrary models: Tools > Open RTW Target</p> <p>New options in the Real-Time Workshop pane of the Configuration Parameters dialog box.</p> <p>See:</p> <ul style="list-style-type: none"> • “Nonlibrary Models: Enhancement for the Real-Time Workshop: Symbols Pane of the Configuration Parameters Dialog Box” on page 32-25 • “Nonlibrary Models: Enhancement for the Real-Time Workshop: Custom Code Pane of the Configuration Parameters Dialog Box” on page 32-26
Library	<p>Migrated from the Simulation Target dialog box to the Configuration Parameters dialog box.</p> <p>See:</p> <ul style="list-style-type: none"> • “Library Models: Changes for the General Pane of the Simulation Target Dialog Box” on page 32-22 • “Library Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box” on page 32-23 • “Library Models: Changes for the Description Pane of the Simulation Target Dialog Box” on page 32-23 	<p>New menu item in Embedded MATLAB Editor for specifying custom code generation options for library models: Tools > Open RTW Target</p> <p>For a description of these options, see “Library Models: Support for Specifying Custom Code Options in the Real-Time Workshop Pane of the Configuration Parameters Dialog Box” on page 32-26.</p>

For details about the new options, see Configuration Parameters Dialog Box in the Simulink Graphical User Interface documentation. For compatibility information, see “Version History” on page 32-30.

For changes specific to Stateflow, see Unified Simulation and Embeddable Code Generation Options for Stateflow Charts and Truth Table Blocks.

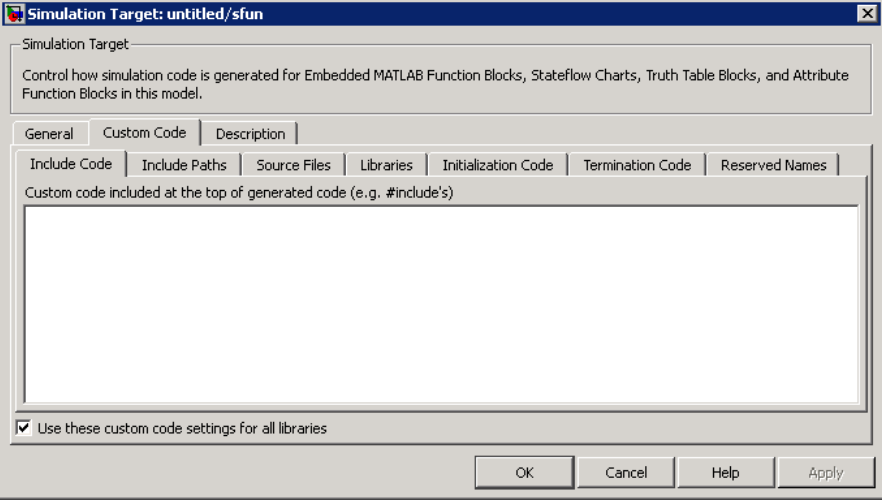
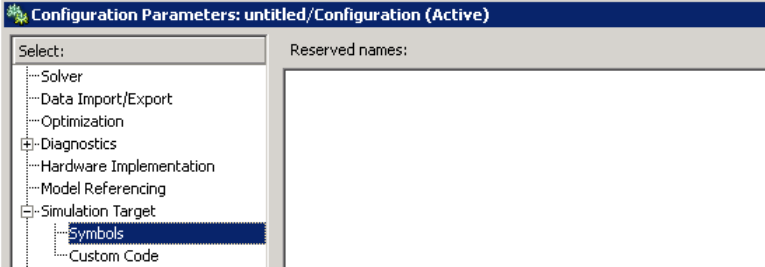
Nonlibrary Models: Changes for the General Pane of the Simulation Target Dialog Box

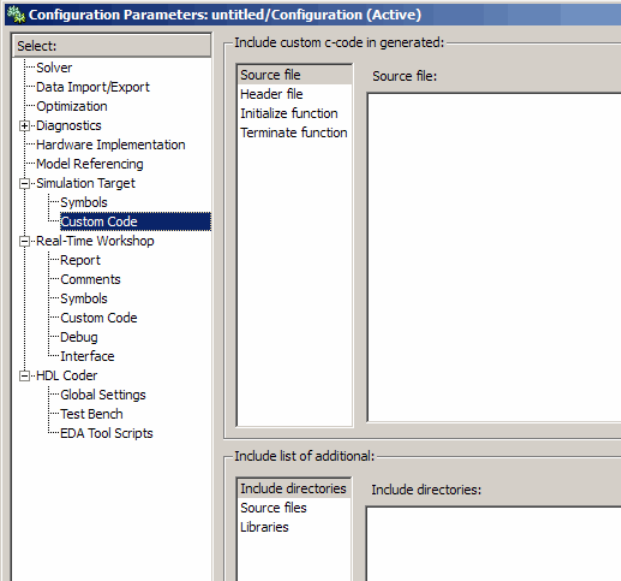
The following sections describe changes in the panes of the Simulation Target dialog box for nonlibrary models.

Release	Appearance
Previous	<p>General pane of the Simulation Target dialog box</p> 
New	<p>Simulation Target pane of the Configuration Parameters dialog box</p> 

For details, see “Nonlibrary Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box” on page 32-21.

Nonlibrary Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box

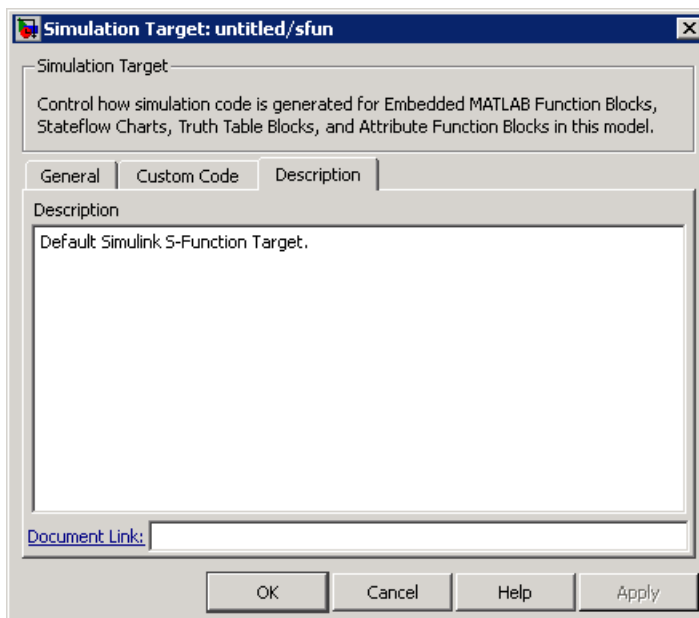
Release	Appearance
Previous	<p>Custom Code pane of the Simulation Target dialog box</p> 
New	<p>Simulation Target > Symbols pane of the Configuration Parameters dialog box</p> 

Release	Appearance
New	<p>Simulation Target > Custom Code pane of the Configuration Parameters dialog box</p> 

For details, see “Nonlibrary Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box” on page 32-21.

Nonlibrary Models: Changes for the Description Pane of the Simulation Target Dialog Box

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



In R2008b, these options are no longer available. For older models where the **Description** pane contained information, the text is now accessible only in the Model Explorer. When you select

Simulink Root > Configuration Preferences in the **Model Hierarchy** pane, the text appears in the **Description** field for that model.

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

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

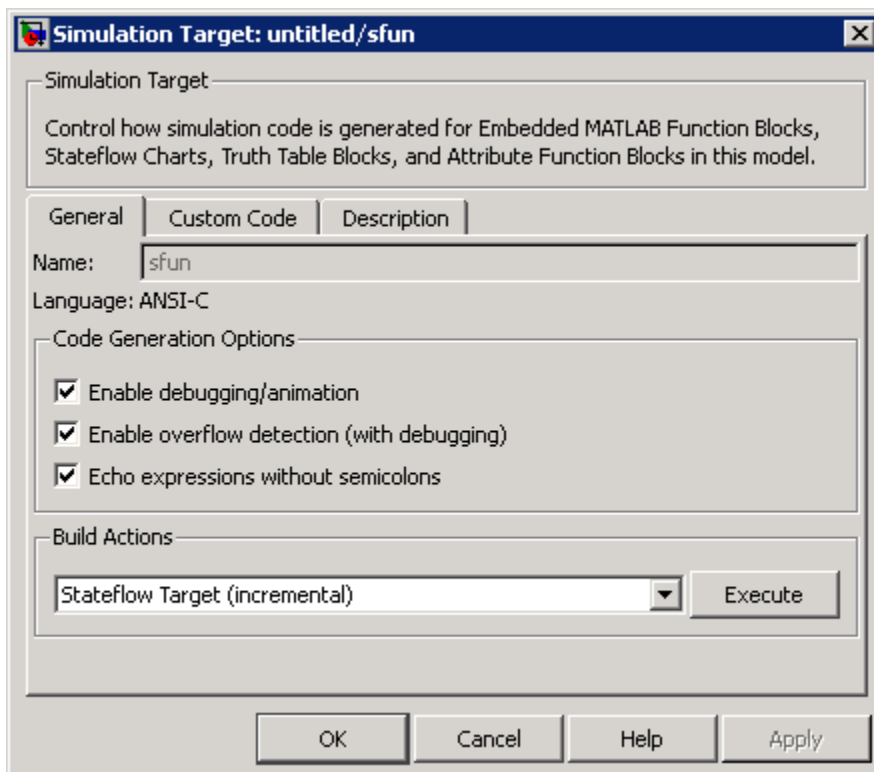
Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
General > Enable debugging / animation	Simulation Target > Enable debugging / animation	on
General > Enable overflow detection (with debugging)	Simulation Target > Enable overflow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

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

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

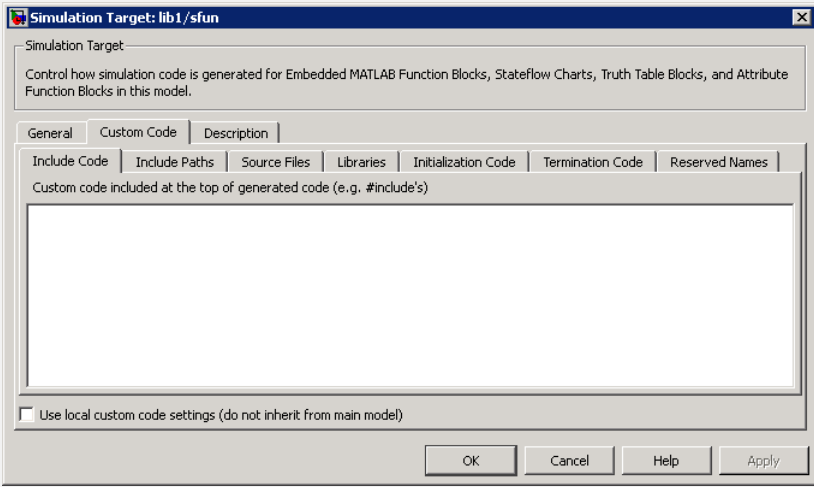
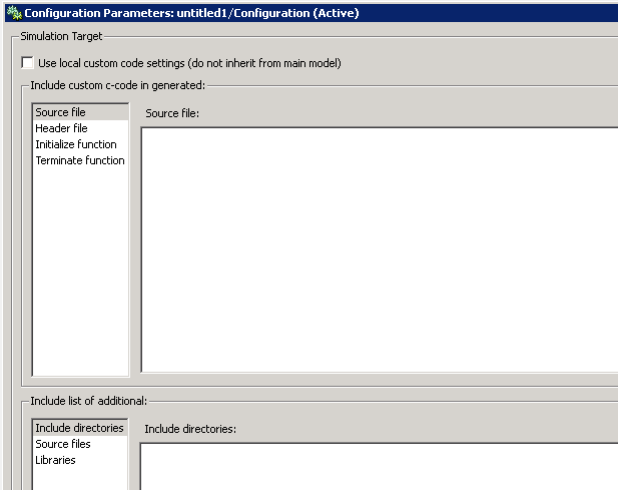
Library Models: Changes for the General Pane of the Simulation Target Dialog Box

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



In R2008b, these options are no longer available. All library models inherit these option settings from the main model to which the libraries are linked.

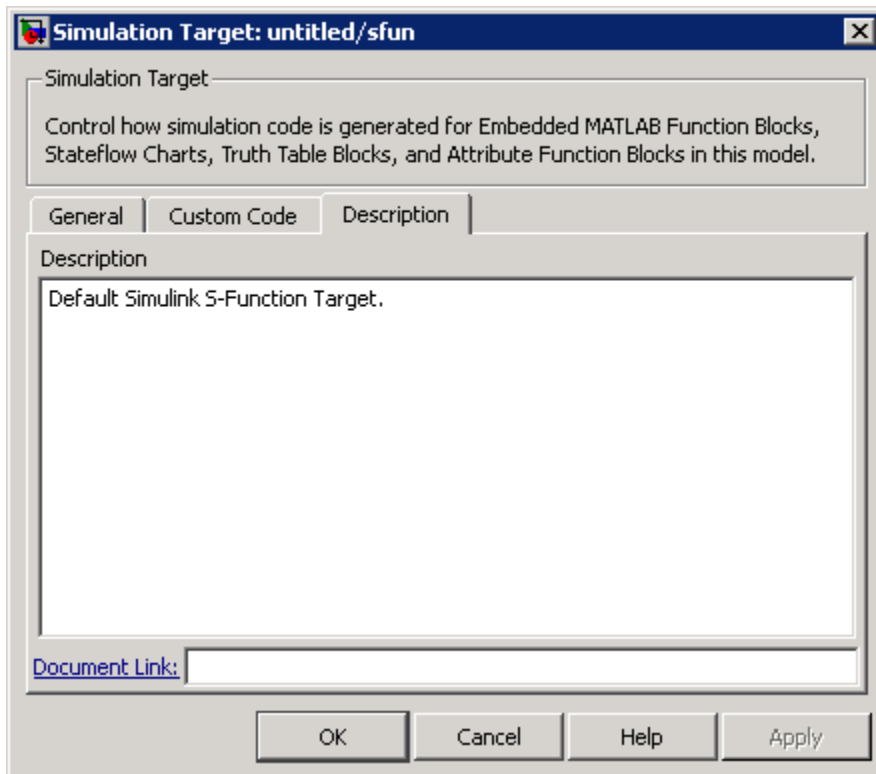
Library Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box

Release	Appearance
Previous	<p>Custom Code pane of the Simulation Target dialog box</p> 
New	<p>Simulation Target pane of the Configuration Parameters dialog box</p> 

For details, see “Library Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box” on page 32-24.

Library Models: Changes for the Description Pane of the Simulation Target Dialog Box

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



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

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

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

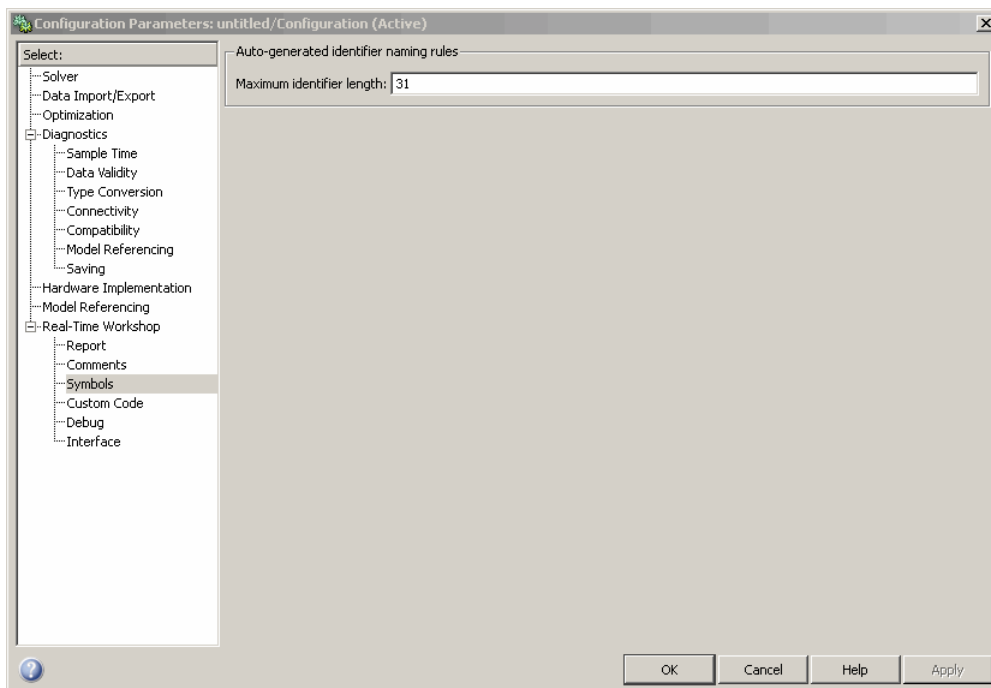
Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
General > Enable debugging / animation	None	Not applicable
General > Enable overflow detection (with debugging)	None	Not applicable
General > Echo expressions without semicolons	None	Not applicable
General > Build Actions	None	Not applicable
None	Simulation Target > Source file	' '
Custom Code > Include Code	Simulation Target > Header file	' '
Custom Code > Include Paths	Simulation Target > Include directories	' '
Custom Code > Source Files	Simulation Target > Source files	' '

Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
Custom Code > Libraries	Simulation Target > Libraries	' '
Custom Code > Initialization Code	Simulation Target > Initialize function	' '
Custom Code > Termination Code	Simulation Target > Terminate function	' '
Custom Code > Reserved Names	None	Not applicable
Custom Code > Use local custom code settings (do not inherit from main model)	Simulation Target > Use local custom code settings (do not inherit from main model)	off
Description > Description	None	Not applicable
Description > Document Link	None	Not applicable

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

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

In previous releases, the **Real-Time Workshop > Symbols** pane of the Configuration Parameters dialog box appeared as follows.



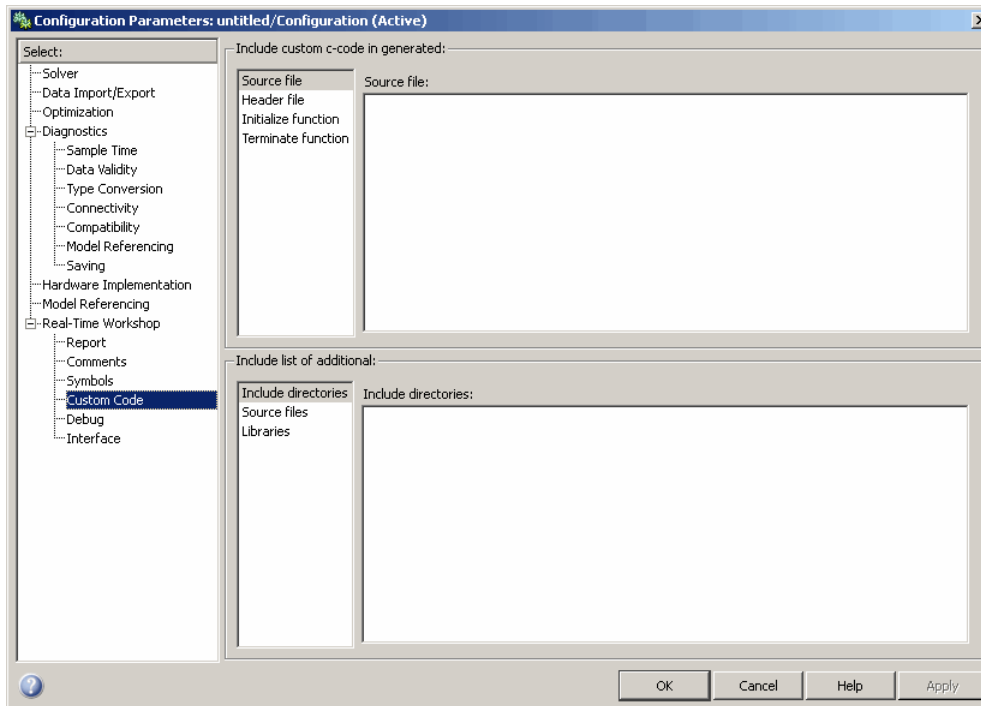
In R2008b, a new option is available in this pane: **Reserved names**. You can use this option to specify a set of keywords that the Real-Time Workshop build process should not use. This action

prevents naming conflicts between functions and variables from external environments and identifiers in the generated code.

You can also choose to use the reserved names specified in the **Simulation Target > Symbols** pane to avoid entering the same information twice for the nonlibrary model. Select the option **Use the same reserved names as Simulation Target**.

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

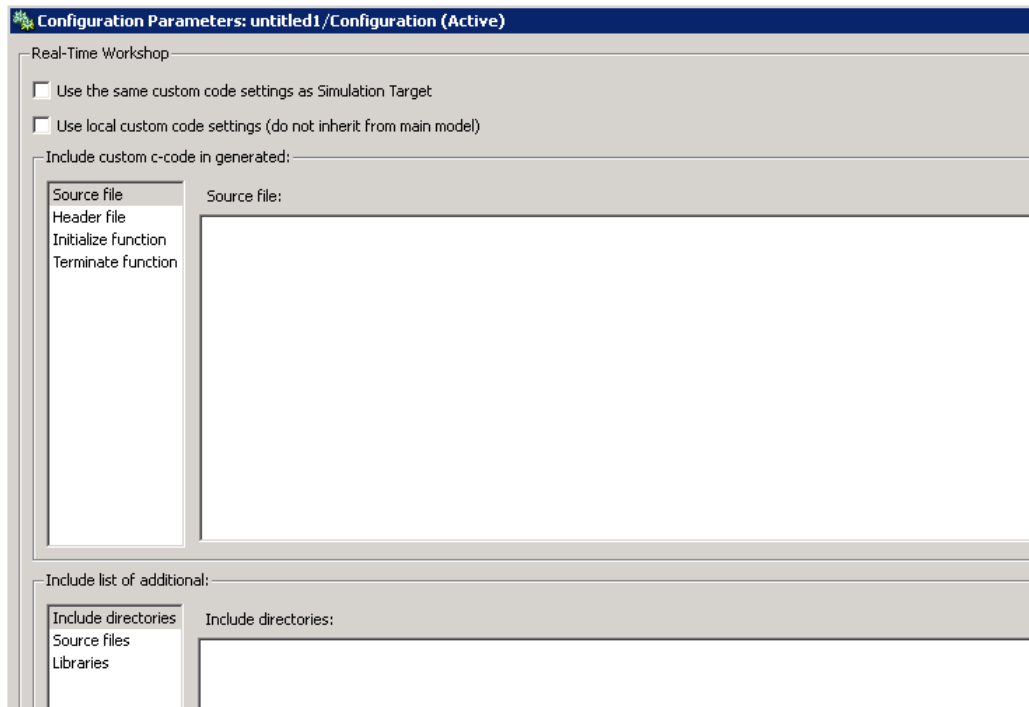
In previous releases, the **Real-Time Workshop > Custom Code** pane of the Configuration Parameters dialog box appeared as follows.



In R2008b, a new option is available in this pane: **Use the same custom code settings as Simulation Target**. You can use this option to copy the custom code settings from the **Simulation Target > Custom Code** pane to avoid entering the same information twice for the nonlibrary model.

Library Models: Support for Specifying Custom Code Options in the Real-Time Workshop Pane of the Configuration Parameters Dialog Box

In R2008b, you can specify custom code options in the Configuration Parameters dialog box, as shown:



For more information, see Code Generation Pane: Custom Code in the Real-Time Workshop documentation.

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

Previously, you could programmatically set options for simulation and embeddable code generation of models containing Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks by accessing the API properties of Target objects `sfun` and `rtw`, respectively. In R2008b, the API properties of Target objects `sfun` and `rtw` are replaced by parameters that you configure using the commands `get_param` and `set_param`.

For compatibility details, see “Version History” on page 32-30.

Mapping of Object Properties to Simulation Parameters for Nonlibrary Models

The following table maps API properties of the Target object `sfun` for nonlibrary models to the equivalent parameters in R2008b. Object properties are listed in alphabetical order; those not listed in the table do not have equivalent parameters in R2008b.

Old <code>sfun</code> Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
<code>CodeFlagsInfo</code> (<code>'debug'</code>)	General > Enable debugging / animation	<code>SFSimEnableDebug</code> string - off, on	Simulation Target > Enable debugging / animation

Old sfun Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CodeFlagsInfo ('overflow')	General > Enable overflow detection (with debugging)	SFSimOverflowDetection string - off, on	Simulation Target > Enable overflow detection (with debugging)
CodeFlagsInfo ('echo')	General > Echo expressions without semicolons	SFSimEcho string - off, on	Simulation Target > Echo expressions without semicolons
CustomCode	Custom Code > Include Code	SimCustomHeaderCode string - ''	Simulation Target > Custom Code > Header file
CustomInitializer	Custom Code > Initialization Code	SimCustomInitializer string - ''	Simulation Target > Custom Code > Initialize function
CustomTerminator	Custom Code > Termination Code	SimCustomTerminator string - ''	Simulation Target > Custom Code > Terminate function
ReservedNames	Custom Code > Reserved Names	SimReservedNameArray string array - {}	Simulation Target > Symbols > Reserved names
UserIncludeDirs	Custom Code > Include Paths	SimUserIncludeDirs string - ''	Simulation Target > Custom Code > Include directories
UserLibraries	Custom Code > Libraries	SimUserLibraries string - ''	Simulation Target > Custom Code > Libraries
UserSources	Custom Code > Source Files	SimUserSources string - ''	Simulation Target > Custom Code > Source files

Mapping of Object Properties to Simulation Parameters for Library Models

The following table maps API properties of the Target object sfun for library models to the equivalent parameters in R2008b. Object properties are listed in alphabetical order; those not listed in the table do not have equivalent parameters in R2008b.

Old sfun Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CustomCode	Custom Code > Include Code	SimCustomHeaderCode string - ''	Simulation Target > Header file

Old sfun Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CustomInitializer	Custom Code > Initialization Code	SimCustomInitializer <i>string - ''</i>	Simulation Target > Initialize function
CustomTerminator	Custom Code > Termination Code	SimCustomTerminator <i>string - ''</i>	Simulation Target > Terminate function
UseLocalCustomCodeSettings	Custom Code > Use local custom code settings (do not inherit from main model)	SimUseLocalCustomCode <i>string - off, on</i>	Simulation Target > Use local custom code settings (do not inherit from main model)
UserIncludeDirs	Custom Code > Include Paths	SimUserIncludeDirs <i>string - ''</i>	Simulation Target > Include directories
UserLibraries	Custom Code > Libraries	SimUserLibraries <i>string - ''</i>	Simulation Target > Libraries
UserSources	Custom Code > Source Files	SimUserSources <i>string - ''</i>	Simulation Target > Source files

Mapping of Object Properties to Code Generation Parameters for Library Models

The following table maps API properties of the Target object rtw for library models to the equivalent parameters in R2008b. Object properties are listed in alphabetical order; those not listed in the table do not have equivalent parameters in R2008b.

Old rtw Object Property	Old Option in the RTW Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CustomCode	Custom Code > Include Code	CustomHeaderCode <i>string - ''</i>	Real-Time Workshop > Header file
CustomInitializer	Custom Code > Initialization Code	CustomInitializer <i>string - ''</i>	Real-Time Workshop > Initialize function
CustomTerminator	Custom Code > Termination Code	CustomTerminator <i>string - ''</i>	Real-Time Workshop > Terminate function

Old rtw Object Property	Old Option in the RTW Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
UseLocalCustomCodeSettings	Custom Code > Use local custom code settings (do not inherit from main model)	RTWUseLocalCustomCode string - off, on	Real-Time Workshop > Use local custom code settings (do not inherit from main model)
UserIncludeDirs	Custom Code > Include Paths	CustomInclude string - ''	Real-Time Workshop > Include directories
UserLibraries	Custom Code > Libraries	CustomLibrary string - ''	Real-Time Workshop > Libraries
UserSources	Custom Code > Source Files	CustomSource string - ''	Real-Time Workshop > Source files

Version History

When you load and save older models in R2008b, not all target property settings are preserved.

What Happens When You Load an Older Model in R2008b

When you use R2008b to load a model created in an earlier version, dialog box options and the equivalent object properties for simulation and embeddable code generation targets migrate automatically to the Configuration Parameters dialog box, except in the cases that follow.

For the simulation target (sfun) of a nonlibrary model, these options and properties do not migrate to the Configuration Parameters dialog box.

Option in the Simulation Target Dialog Box of a Nonlibrary Model	Equivalent Object Property
Custom Code > Use these custom code settings for all libraries	ApplyToAllLibs
Description > Description	Description Note If you load an older model that contained user-specified text in the Description field, that text now appears in the Model Explorer. When you select Simulink Root > Configuration Preferences in the Model Hierarchy pane, the text appears in the Description field for that model.
Description > Document Link	Document

For the simulation target (sfun) of a library model, these options and properties do not migrate to the Configuration Parameters dialog box.

Option in the Simulation Target Dialog Box of a Library Model	Equivalent Object Property
General > Enable debugging / animation	CodeFlagsInfo('debug')
General > Enable overflow detection (with debugging)	CodeFlagsInfo('overflow')
General > Echo expressions without semicolons	CodeFlagsInfo('echo')
General > Build Actions	None
Custom Code > Reserved Names	ReservedNames
Description > Description	Description
Description > Document Link	Document

For the embeddable code generation target (rtw) of a library model, these options and properties do not migrate to the Configuration Parameters dialog box.

Option in the RTW Target Dialog Box of a Library Model	Equivalent Object Property
General > Comments in generated code	CodeFlagsInfo('comments')
General > Use bitsets for storing state configuration	CodeFlagsInfo('statebitsets')
General > Use bitsets for storing boolean data	CodeFlagsInfo('databitsets')
General > Compact nested if-else using logical AND/OR operators	CodeFlagsInfo('emitlogicalops')
General > Recognize if-elseif-else in nested if-else statements	CodeFlagsInfo('elseifdetection')
General > Replace constant expressions by a single constant	CodeFlagsInfo('constantfolding')
General > Minimize array reads using temporary variables	CodeFlagsInfo('redundantloadelimination')
General > Preserve symbol names	CodeFlagsInfo('preservenames')
General > Append symbol names with parent names	CodeFlagsInfo('preservenameswithparent')
General > Use chart names with no mangling	CodeFlagsInfo('exportcharts')
General > Build Actions	None
Custom Code > Reserved Names	ReservedNames
Description > Description	Description
Description > Document Link	Document

What Happens When You Save an Older Model in R2008b

When you use R2008b to save a model created in an earlier version, parameters for simulation and embeddable code generation from the Configuration Parameters dialog box are saved. However, properties of API Target objects `sfun` and `rtw` are not saved if those properties do not have an equivalent parameter in the Configuration Parameters dialog box. In R2008b, this behavior applies even if you choose to save the model as an older version (for example, R2007a).

New Parameters in the Configuration Parameters Dialog Box for Simulation and Embeddable Code Generation

In R2008b, new parameters are added to the Configuration Parameters dialog box for simulation and embeddable code generation of models that contain Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks.

New Simulation Parameters for Nonlibrary Models

The following table lists the new simulation parameters that apply to nonlibrary models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
SimBuildMode <i>string</i> - <code>sf_incremental_build</code> , <code>sf_nonincremental_build</code> , <code>sf_make</code> , <code>sf_make_clean</code> , <code>sf_make_clean_objects</code>	Simulation Target > Simulation target build mode	Specifies how you build the simulation target for a model.
SimCustomSourceCode <i>string</i> - ''	Simulation Target > Custom Code > Source file	Enter code lines to appear near the top of a generated source code file.

New Simulation Parameter for Library Models

The following table lists the new simulation parameter that applies to library models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
SimCustomSourceCode <i>string</i> - ''	Simulation Target > Source file	Enter code lines to appear near the top of a generated source code file.

New Code Generation Parameters for Nonlibrary Models

The following table lists the new code generation parameters that apply to nonlibrary models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
ReservedNameArray <i>string array</i> - {}	Real-Time Workshop > Symbols > Reserved names	Enter the names of variables or functions in the generated code that match the names of variables or functions specified in custom code.
RTWUseSimCustomCode <i>string</i> - <code>off</code> , <code>on</code>	Real-Time Workshop > Custom Code > Use the same custom code settings as Simulation Target	Specify whether to use the same custom code settings as those specified for simulation.
UseSimReservedNames <i>string</i> - <code>off</code> , <code>on</code>	Real-Time Workshop > Symbols > Use the same reserved names as Simulation Target	Specify whether to use the same reserved names as those specified for simulation.

New Code Generation Parameters for Library Models

The following table lists the new code generation parameters that apply to library models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
CustomSourceCode <i>string</i> - ''	Real-Time Workshop > Source file	Enter code lines to appear near the top of a generated source code file.
RTWUseSimCustomCode <i>string</i> - off , on	Real-Time Workshop > Use the same custom code settings as Simulation Target	Specify whether to use the same custom code settings as those specified for simulation.

S-Functions

Ada S-Functions

In future releases, Simulink will not have a built-in Ada S-function capability. As a mitigation strategy, call Ada code from Simulink using standard Ada 95 language features and the Simulink C-MEX S-function API. For details of this process, please contact Technical Support at MathWorks.

Legacy Code Tool Enhancement

The Legacy Code Tool data structure has been enhanced with a new S-function options field, `singleCPPMexFile`, which when set to `true`

- Requires you to generate and manage an inlined S-function as only one file (.cpp) instead of two (.c and .tlc)
- Maintains model code style—level of parentheses usage and preservation of operand order in expressions and condition expressions in `if` statements—as specified by model configuration parameters.

When you choose not to use this option, code generated by the Legacy Code Tool does not reflect code style configuration settings and requires you to manage C-MEX and TLC files.

For more information, see:

- Integrating Existing C Functions into Simulink Models with the Legacy Code Tool in the Writing S-Functions documentation
- Integrate External Code Using Legacy Code Tool in the Real-Time Workshop documentation
- `legacy_code` function reference page

Version History

- If you upgrade from an earlier release, you can continue to use S-functions generated from the Legacy Code Tool available in earlier releases. You can continue to compile the S-function source code and you can continue to use the compiled output from an earlier release without recompiling the code.
- If you set the new `singleCPPMexFile` options field to `true`, when creating an S-function, you cannot use that S-function, in source or compiled form, with versions of Simulink earlier than Version 7.2 (R2008b).

MATLAB Changes Affecting Simulink

Changes to MATLAB Startup Options

The `matlab` command line arguments `-memmgr` and `-check_malloc` are deprecated and will be removed in a future release.

For more information, see [Changes to matlab Memory Manager Startup Options](#) in the MATLAB Release Notes.

MATLAB Graphics Tools Not Supported Under `-nojvm` Startup Option

If you start MATLAB using the command `matlab -nojvm` (which disables Java), you will receive warnings when using many graphical tools, for example, when you create figures, print Simulink models, or view Simulink scopes.

R2008a

Version: 7.1

New Features

Bug Fixes

Version History

Simulation Performance

Rapid Accelerator

Improved Rapid Accelerator sim-command performance when running long simulations of small models on Microsoft Windows platforms.

Long Rapid Accelerator mode simulations of small models invoked by the `sim` command under the Microsoft Windows operating system now run faster.

Additional Zero Crossing Algorithm

A second zero crossing algorithm that is especially useful in systems exhibiting strong chattering behavior has been added for use with variable step solvers.

The new algorithm is selected by choosing `Adaptive` from the **Zero crossing location algorithm** option in the Solver pane of the Configuration Parameter dialog. The default algorithm is `Non-Adaptive`, which is the algorithm used prior to this release.

For more information, see [Zero-Crossing Algorithms](#).

Component-Based Modeling

Efficient Parent Model Rebuilds

In previous releases, changing a referenced model that executed in Accelerator mode or was used for code generation triggered rebuilding every model that directly or indirectly referenced the changed model. The rebuilding occurred even if the change to the referenced model had no effect on its interface to its parent(s).

In R2008a, changing a referenced model that executes in Accelerator mode or is used for code generation triggers rebuilding a parent model only when the change directly affects the referenced model's interface to the parent model. This behavior eliminates unnecessary code regeneration, which can significantly reduce the time needed to update a diagram.

The faster diagram update has no effect on simulation behavior or performance, but may change the messages that appear in the MATLAB Command Window. See Referencing a Model for information about model referencing.

Scalar Root Inputs Passed Only by Reference

The **Configuration Parameters > Model Referencing > Pass scalar root inputs by value** option is `Off` by default, indicating that scalar root inputs are passed by reference. In previous releases, setting the option to `On` affected both simulation and generated code, and caused scalar root inputs to be passed by value. In R2008a, the option has no effect on simulation: scalar root inputs are now always passed by reference, regardless of the setting of **Pass scalar root inputs by value**. The effect of the option on code generation is the same as in previous releases. See Pass fixed-size scalar root inputs by value for code generation for more information.

Unlimited Referenced Models

In previous releases, Microsoft Windows imposed a limit on the number of models that could be referenced in Accelerator mode in a model hierarchy. This limitation is removed in R2008a. Under Microsoft Windows, as on all other platforms, the number of referenced models that can appear in a model hierarchy is effectively unlimited. See Referencing a Model for information about model referencing.

Embedded MATLAB Function Blocks

Nontunable Structure Parameters

Embedded MATLAB Function blocks now support nontunable MATLAB structure parameters. For more information, see [Working with Structure Parameters in MATLAB Function Blocks](#).

Bidirectional Traceability

You can navigate between a line of generated code and its corresponding line of source code in Embedded MATLAB Function blocks. For more information, see [Using Traceability in MATLAB Function Blocks](#).

Specify Scaling Explicitly for Fixed-Point Data

When you define data of fixed-point type in Embedded MATLAB Function blocks, you must specify the scaling explicitly in the General pane of the Data properties dialog box. For example, you cannot enter an incomplete specification such as `fixdt(1,16)` in the Type field. If you do not specify scaling explicitly, you will see an error message when you try to simulate your model.

To ensure that the data type definition is valid for fixed-point data, perform one of these steps in the General pane of the Data properties dialog box:

- Use a predefined option in the Type drop-down menu.
- Use the Data Type Assistant to specify the Mode as fixed-point.

Version History

Previously, you could omit scaling in data type definitions for fixed-point data. Such data types were treated as integers with the specified sign and word length. This behavior has changed. Embedded MATLAB Function blocks created in earlier versions may now generate errors if they contain fixed-point data with no scaling specified.

Data Management

Array Format Cannot Be Used to Export Multiple Matrix Signals

When you export signals to a workspace in Array format from more than one output, none of the signals can be a matrix signal. In previous releases, violating this rule did not always cause an error, but the matrix data was not exported correctly. In R2008a, violating the rule always causes an error, and no data export occurs.

When exporting data to a workspace in Array format from multiple outputs, use a Reshape block to convert any matrix signal to a one-dimensional (1-D) array. This restriction applies only to Array format. If you specify either Structure or Structure with time format, you can export matrix signals to a workspace from multiple outputs without first converting the signals to vectors.

Version History

The more stringent error checking in R2008a can cause models that export data in Array format from multiple outputs to generate errors rather than silently exporting matrix data incorrectly. To eliminate such errors, use a Reshape block to convert any matrix signal to a vector, or switch to Structure or Structure with time format. See Exporting Simulation Data for information about data export.

Bus Editor Upgraded

The Simulink Bus Editor has been reimplemented to provide a GUI interface similar to that of the Model Explorer, and to provide several new capabilities, including importing/exporting data from MAT-files and M-files, defining bus objects and elements with the Data Type Assistant, and creating and viewing bus hierarchies (nested bus objects). See Using the Bus Editor for details.

Changing Nontunable Values Does Not Affect the Current Simulation

In previous releases, changing the value of any variable or parameter during simulation took effect immediately. In R2008a, only changes to tunable variables and parameters take effect immediately. Other changes have no effect until the next simulation begins. This modification causes simulation behavior to match generated code behavior when the values of nontunable variables and parameters change, and it improves efficiency by eliminating unnecessary re-evaluation. For information about parameter tuning, see Tunable Parameters and Using Tunable Parameters.

Version History

In R2008a, simulation behavior will differ if the behavior in a previous release depended on changing any nontunable variable or parameter during simulation. To regain the previous behavior, define as tunable any nontunable variable or parameter that you want to change during simulation for the purpose of affecting simulation immediately.

Detection of Illegal Rate Transitions

Illegal rate transitions between a block and a triggered subsystems or function call subsystems are now detected when the block is connected to a Unit Delay or Zero Hold block inside a triggered subsystem.

Version History

In this release, Simulink detects illegal rate transition errors when the block sample time is different from the triggered subsystem sample time in those models where the block is connected to a Unit Delay or Zero Hold block inside the triggered subsystem.

Explicit Scaling Required for Fixed-Point Data

In R2008a, when you define a fixed-point data type in a Simulink model, you must explicitly specify the scaling unless the block supports either integer scaling mode or best-precision scaling mode. If a block supports neither of these modes, you cannot define an incomplete fixed-point data type like `fixdt(1,16)`, which specifies no scaling. See [Specifying a Fixed-Point Data Type and Showing Fixed-Point Details](#) for information about defining fixed-point data types.

Version History

In previous releases, you could define a fixed-point data type that specified no scaling in a block that supported neither integer scaling mode nor best-precision scaling mode. The Simulink software posted no warning, and treated fixed-point data type as an integer data type with the specified word length. For example, `fixdt(1,16)` was treated as `fixdt(1,16,0)`.

In R2008a, a fixed-point data type definition that specifies no scaling generates an error unless the block supports either integer scaling mode or best-precision scaling mode. If such an error occurs when you compile a model from an earlier Simulink version, redefine the incomplete fixed-point data type to be an integer type if nothing more is needed, or to be scaled appropriately for its value range.

Fixed-Point Details Display Available

The Data Type Assistant can now display the status and details of fixed-point data types. See [Specifying a Fixed-Point Data Type and Showing Fixed-Point Details](#) for more information.

More than 2GB of Simulation Data Can be Logged on 64-Bit Platforms

When you log time, states, final states, and signals on a 64-bit platform, you can now save more simulation data in the MATLAB base workspace than was previously possible.

- When you log data using the `Structure`, `Structure with time`, or `Timeseries` format, you can now save up to $2^{48}-1$ bytes in each field that contains logged data.
- When you log data using `Array` format, you can now save up to $2^{48}-1$ bytes in each array that contains logged data.

Previously the limit was $2^{31}-1$ bytes in each field or array containing logged data. See [Exporting Signal Data Using Signal Logging and Data Import/Export Pane](#) for information about logging data.

Order of Simulink and MPT Parameter and Signal Fields Changed

The order of the fields in the `Simulink.Parameter` and `Simulink.Signal` classes, and in their respective subclasses `mpt.Parameter` and `mpt.Signal`, has changed in R2008a.

The order for `Simulink.Parameter` (and `mpt.Parameter`) is now:

```

Simulink.Parameter (handle)
  Value: []
  RTWInfo: [1x1 Simulink.ParamRTWInfo]
  Description: ''
  DataType: 'auto'
  Min: -Inf
  Max: Inf
  DocUnits: ''
  Complexity: 'real'
  Dimensions: [0 0]

```

The order for `Simulink.Signal` (and `mpt.Signal`) is now:

```

Simulink.Signal (handle)
  RTWInfo: [1x1 Simulink.SignalRTWInfo]
  Description: ''
  DataType: 'auto'
  Min: -Inf
  Max: Inf
  DocUnits: ''
  Dimensions: -1
  Complexity: 'auto'
  SampleTime: -1
  SamplingMode: 'auto'
  InitialValue: ''

```

Loading a model that uses any `Simulink.Parameter` or `mpt.Parameter` objects, and was saved in a release prior to R2008a, may post an Inconsistent Data warning in the MATLAB Command Window. This message does not indicate a problem with the model, which need not be changed. Resave the model in R2008a to update it to use the new parameter class definitions. The warning will not appear when you reopen the model.

Range Checking for Complex Numbers

Previous releases did not provide range checking for complex numbers, and attempting it generated an error. In R2008a, you can specify a minimum and/or maximum value for a complex number wherever range checking is available and a complex number is a legal value.

The specified minimum and maximum values apply separately to the real part and to the imaginary part of the complex number. If the value of either part of the number is less than the minimum, or greater than the maximum, the complex number is outside the specified range.

No range checking occurs against any combination of the real and imaginary parts, such as $(\sqrt{a^2+b^2})$. See [Checking Parameter Values and Signal Ranges](#) for information about range checking.

Rate Transition Blocks Needed on Virtual Buses

In this release, Simulink never automatically inserts a Rate Transition block into a virtual bus, even if `Automatically handle rate transfer` is selected. Instead, an error is displayed indicating that a Rate Transition block must be manually inserted.

Version History

Some models that worked in previous releases, but were dependent on automatic Rate Transition block insertion, will now report an error and will no longer run. An error will be reported if all of these apply:

- The Automatically handle rate transfer option is enabled
- The model is multirate
- The model has a virtual bus, all of the elements on the bus have the same data type, and the sample time changes
- A bus selector block is not present on the virtual bus at a point after the sample time changes
- The only way to address the rate transition problem is to insert a rate transition block

Sample Times for Virtual Blocks

In models with asynchronous function calls, some virtual blocks now correctly assign generic sample times instead of triggered sample times.

Version History

The `CompiledSampleTime` parameter now reports the compiled sample time as generic sample time (that is, [-1, -inf]) rather than triggered sample time ([-1,-1]) for virtual blocks for which all of the following is true:

- The virtual block is downstream from an asynchronous source
- The virtual block is not inside a triggered subsystem
- The virtual block had a triggered ([-1,-1]) sample time in previous releases

The simulation results, code generation, and sample time colors are not affected by this change.

Signals Needing Resolution Are Graphically Indicated

In R2008a, the Simulink Editor by default graphically indicates signals that must resolve to signal objects. For any labeled signal whose **Signal name must resolve to signal object** property is enabled, a signal resolution icon appears to the left of the signal name. The icon looks like this:



A signal resolution icon indicates only that a signal's **Signal name must resolve to signal object** property is enabled. The icon does not indicate whether the signal is actually resolved, and does not appear on a signal that is implicitly resolved without its **Signal name must resolve to signal object** property being enabled. See Signal Resolution Indicators for more information.

Simulink File Management

Autosave

New Autosave option automatically creates a backup copy of models before updating or simulating. If you open or load a model which has a more recent autosave copy available, a dialog appears where you can choose to overwrite the original model file with the autosave copy.

You can set the Autosave option in the Simulink Preferences Window. See Autosave.

Old Version Notification

New option to notify when loading a model saved in a previous version of Simulink software.

You can set this option in the Simulink Preferences Window. See Simulink Preferences Window: Main Pane.

Model Dependencies Tools

Enhanced file dependency analysis now also detects:

- TLC files required by S-functions.
- .fig files created by GUIDE.
- Files referenced by common data loading functions. File names passed to `xlsread`, `importdata`, `dlmread`, `csvread`, `wk1read`, and `textread` are now identified, in addition to the existing capability for `load`, `fopen` and `imread`.

See Scope of Dependency Analysis.

Block Enhancements

New Discrete FIR Filter Block Replaces Weighted Moving Average Block

The Discrete FIR Filter block in the Discrete library is new for this release. This block independently filters each channel of the input signal with the specified digital FIR filter. The Discrete FIR Filter block replaces the Weighted Moving Average block.

Version History

You should replace Weighted Moving Average blocks in your existing models with the Discrete FIR Filter block. To do this, run the `s!update` command on your models.

Rate Transition Block Enhancements

Support for Rate Transition blocks has been enhanced in the following ways:

- Rate Transition block output port sample time now can be specified as a multiple of input port sample time, using the new Rate Transition block parameters **Output port sample time options** and **Sample time multiple (>0)**. See the Rate Transition block documentation for more information.
- In previous releases, auto-insertion of Rate Transition blocks was selected for a model using the option **Automatically handle data transfers between tasks** on the **Solver** pane of the Configuration Parameters dialog box. When selected, auto-insertion always ensured data transfer determinism for periodic tasks.

This release allows you to control the level of data transfer determinism when auto-insertion of Rate Transition blocks is selected for your model. The **Solver** pane option for selecting auto-insertion has been renamed to **Automatically handle rate transition for data transfer**. Selecting auto-insertion now enables a new option, **Deterministic data transfer**. Selecting **Never (minimum delay)** or **Whenever possible** for this option can provide reduced latency for models that do not require determinism. See the Solver Pane section in the Simulink Graphical User Interface documentation for more information.

- Auto-insertion of Rate Transition blocks is now supported for additional rate transitions, such as sample times with nonzero offset, and between non-integer-multiple sample times.

Enhanced Lookup Table (n-D) Block

The Lookup Table (n-D) block now supports all data types, complex table data, and nonscalar inputs. See the Lookup Table (n-D) block documentation for more information.

New Accumulator Parameter on Sum Block

The Sum block dialog box displays a new parameter for specifying the data type of its accumulator. See the Sum block documentation for more information.

User Interface Enhancements

Simulink Library Browser

A new version of the Simulink Library browser has the following enhancements:

- Now available on all platforms supported by Simulink software.
- Improved performance for browsing and searching of libraries, by allowing these operations to proceed without actually loading the libraries.
- Enhanced search finds all blocks and displays search results in a separate tab.
- New option to display library blocks in a compact grid layout that conserves screen space.

Simulink Preferences Window

New unified Simulink Preferences window for configuring default settings. The new Preferences window allows you to configure file change notifications, autosave options, fonts, display options, and model configuration defaults.

See Simulink Preferences Window: Main Pane.

Model Advisor

In R2008a, the Model Advisor tool is enhanced with improved GUI navigation, check analysis, and reports including:

- Reset option that reverts the status of all checks to **Not Run** while keeping the current check selection.
- Model Advisor Result Explorer to make changes to your model.
- **Input Parameters** to provide inputs to checks.
- Check results reported in the same order as the Model Advisor tree.
- The ability to generate reports for any folder.
- Timestamps in reports indicating when checks run at different times.

See Consulting the Model Advisor in the Simulink User's Guide.

Solver Controls

Enhanced controls in the Solver pane of the Configuration Parameters dialog. The Solver pane of the Configuration Parameters dialog has been changed as follows:

- The **Solver diagnostic controls** pane has been removed and two new panes have been added (**Tasking and sample time options**, and **Zero crossing options**)
- The Automatically handle data transfers between tasks control has been moved to the **Tasking and sample time options** pane, and has been renamed Automatically handle rate transition for data transfer
- The Higher priority value indicates higher task priority control has been moved to the **Tasking and sample time options** pane

- The Number of consecutive min step size violations allowed control has been moved to the **Solver options** pane, and has been renamed Consecutive min step size violations allowed
- The States shape preservation control has been added to the **Solver options** pane
- The Consecutive zero crossings relative tolerance control has been moved to the **Zero crossing options** pane
- The Number of consecutive zero crossings allowed control has been moved to the **Zero crossing options** pane
- The Zero crossing control control has been moved to the **Zero crossing options** pane
- The Zero crossing location algorithm control has been added to the **Zero crossing options** pane
- The Zero crossing location threshold control has been added to the **Zero crossing options** pane
- Options that in previous releases were only visible when enabled are now always visible. They are grayed when not enabled.

For more information on the Configuration parameters solver pane, see Solver Pane.

Version History

The Solver pane of the Configuration Parameter dialog has been restructured, and many parameters have moved or been renamed. Please refer to the list of changes above for information on specific parameters.

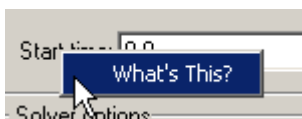
“What’s This?” Context-Sensitive Help Available for Simulink Configuration Parameters Dialog

R2008a introduces “What's This?” context-sensitive help for parameters that appear in the Simulink Configuration Parameters dialog. This feature provides quick access to a detailed description of the parameters, saving you the time it would take to find the information in the Help browser.

To use the "What's This?" help, do the following:

- 1 Place your cursor over the label of a parameter.
- 2 Right-click. A **What's This?** context menu appears.

For example, the following figure shows the **What's This?** context menu appearing after a right-click on the **Start time** parameter in the **Solver** pane.



- 3 Click **What's This?** A context-sensitive help window appears showing a description of the parameter.

S-Functions

Simplified Level-2 M-File S-Function Template

New basic version of the Level-2 M-file S-function template `msfuntmpl_basic.m` simplifies creating Level-2 M-file S-functions. See Writing Level-2 MATLAB S-Functions in *Writing S-Functions* for more information.

Version History

MATLAB V7.6 (R2008a) on Linus Torvalds' Linux® platforms is now built with a compiler that utilizes glibc version 2.3.6. To work with MATLAB V7.6 (R2008a), MEX-file S-functions compiled on a Linux platform must be rebuilt.

R2007b

Version: 7.0

New Features

Bug Fixes

Version History

Simulation Performance

Simulink Accelerator™

Simulink Accelerator™ has been incorporated into Simulink software, and a new Rapid Accelerator mode has been added for faster simulation through code generation technology. See *Accelerating Models* in Simulink User's Guide.

Note When using From File blocks in Rapid Accelerator mode, the corresponding MAT file must be in the current directory.

Version History

A license is no longer required to use the Accelerator or Rapid Accelerator modes.

Simulink Profiler

Simulink Profiler has been incorporated into Simulink software for the identification of simulation performance bottlenecks. See *Capturing Performance Data* in Simulink User's Guide.

Compiler Optimization Level

Simulink Accelerator mode, Rapid Accelerator mode, and model reference simulation targets can now specify the compiler optimization level used (choose between minimizing compilation time or simulation time). See *Customizing the Build Process* in Simulink User's Guide.

Version History

The new model configuration parameter **Compiler optimization level** defaults to **Optimizations off (faster builds)**. As a result, you might notice shorter build times, but longer execution times, compared to previous releases. However, any previously defined custom compiler optimization options using `OPT_OPTS` will be honored, and model behavior should be unchanged.

Variable-Step Discrete Solver

Simulink software has been enhanced to no longer take unnecessary time steps at multiples of the maximum step size when using a variable-step discrete solver.

Referenced Models Can Execute in Normal or Accelerator Mode

In previous releases, Simulink software executed all referenced models by generating code for them and executing the generated code. In this release, Simulink software can execute appropriately configured referenced models interpretively. Such execution is called Normal mode execution, and execution via generated code is now called Accelerator mode execution. The technique of executing a referenced model via generated code has not changed, but it did not previously need a separate name because it was the only alternative.

Many restrictions that previously applied to all referenced model execution now apply only to Accelerator mode execution, and are relaxed in Normal mode. For example, some Simulink tools that

did not work with referenced models because they are incompatible with generated code can now be used by executing the referenced model in Normal mode.

Normal mode also has some restrictions that do not apply to Accelerator mode. For example, at most, one instance of a given model in a referenced model hierarchy can execute in Normal mode. See Referencing a Model in Simulink User's Guide for information about using referenced models in Normal and Accelerator mode.

Accelerator and Model Reference Targets Now Use Standard Internal Functions

For more consistent simulation results, Simulink Accelerator mode, Rapid Accelerator mode, and the model reference simulation target now perform mathematical operations with the same internal functions that MATLAB and Simulink products use.

Component-Based Modeling

New Instance View Option for the Model Dependency Viewer

The Model Dependency viewer has a new option to display each reference to a model and indicate whether the reference is simulated in Accelerator or Normal mode. See Referencing a Model and Model Dependency Viewer in Simulink User's Guide.

Mask Editor Now Requires Java

The Mask Editor now requires that the MATLAB product start with Java enabled. See Simulink Mask Editor in Simulink User's Guide.

Version History

You can no longer use the Mask Editor if you start MATLAB with the `-nojvm` option.

Embedded MATLAB Function Blocks

Complex and Fixed-Point Parameters

Embedded MATLAB Function blocks now support complex and fixed-point parameters.

Support for Algorithms That Span Multiple M-Files

You can now generate embeddable code for external M-functions from Embedded MATLAB function blocks. This feature allows you to call external functions from multiple locations in an M-file or model and include these functions in the generated code.

Version History

In previous releases, Embedded MATLAB function blocks did not compile external M-functions, but instead dispatched them to the MATLAB product for execution (after warning). Now, the default behavior is to compile and generate code for external M-functions called from Embedded MATLAB function blocks. If you do not want Embedded MATLAB function blocks to compile external M-functions, you must explicitly declare them to be extrinsic, as described in *Calling MATLAB Functions* in the Embedded MATLAB documentation.

Loading R2007b Embedded MATLAB Function Blocks in Earlier Versions of Simulink Software

If you save Embedded MATLAB Function blocks in R2007b, you will not be able to load the corresponding model in earlier versions of Simulink software. To work around this issue, save your model in the earlier version before loading it, as follows:

- 1 In the Simulink Editor, select **File > Save As**.
- 2 In the **Save as type** field, select the version in which you want to load the model.

For example, if you want to load the model in Simulink R2007a, select **Simulink 6.6/R2007a Models (*.mdl)**.

Data Management

New Diagnostic for Continuous Sample Time on Non-Floating-Point Signals

A new diagnostic detects continuous sample time on non-floating-point signals.

New Standardized User Interface for Specifying Data Types

This release introduces a new standardized user interface, the **Data Type Assistant**, for specifying data types associated with Simulink blocks and data objects, as well as Stateflow data. See Using the Data Type Assistant in Simulink User's Guide for more information.

The **Data Type Assistant** appears on the dialogs of the following Simulink blocks:

- Abs
- Constant
- Data Store Memory
- Data Type Conversion
- Difference
- Discrete Derivative
- Discrete-Time Integrator
- Dot Product
- MATLAB Function (formally called Embedded MATLAB Function)
- Gain
- Inport
- Interpolation Using Prelookup
- Logical Operator
- Lookup Table
- Lookup Table (2-D)
- Lookup Table Dynamic
- Math Function
- MinMax
- Multiport Switch
- Outport
- Prelookup
- Product, Divide, Product of Elements
- Relational Operator
- Relay
- Repeating Sequence Interpolated
- Repeating Sequence Stair
- Saturation

- Saturation Dynamic
- Signal Specification
- Sum, Add, Subtract, Sum of Elements
- Switch
- Weighted Moving Average (obsolete — replaced by the Discrete FIR Filter block)

The **Data Type Assistant** appears on the dialogs of the following Simulink data objects:

- Simulink.BusElement
- Simulink.Parameter
- Simulink.Signal

New Block Parameters for Specifying Minimum and Maximum Values

The following new block parameters are available for specifying the minimum and maximum values of signals and other block parameters.

- **Output minimum, Minimum**
- **Output maximum, Maximum**
- **Parameter minimum**
- **Parameter maximum**

These new parameters selectively appear on the dialogs of the following Simulink blocks:

- Abs
- Constant
- Data Store Memory
- Data Type Conversion
- Difference
- Discrete Derivative
- Discrete-Time Integrator
- Gain
- Inport
- Interpolation Using Prelookup
- Lookup Table
- Lookup Table (2-D)
- Math Function
- MinMax
- Multiport Switch
- Outport
- Product, Divide, Product of Elements
- Relay
- Repeating Sequence Interpolated

- Repeating Sequence Stair
- Saturation
- Saturation Dynamic
- Signal Specification
- Sum, Add, Subtract, Sum of Elements
- Switch

New Range Checking of Block Parameters

In this release, Simulink software performs range checking of parameters associated with blocks that specify minimum and maximum values (see “New Block Parameters for Specifying Minimum and Maximum Values” on page 34-7). Simulink software alerts you when values of block parameters lie outside a range that corresponds to its specified minimum and maximum values and data type. See Checking Parameter Values in Simulink User's Guide for more information.

New Diagnostic for Checking Signal Ranges During Simulation

In the Configuration Parameters dialog, the **Diagnostics > Data Validity** pane contains a new diagnostic, **Simulation range checking**, which alerts you during simulation when blocks output signals that exceed specified minimum or maximum values (see “New Block Parameters for Specifying Minimum and Maximum Values” on page 34-7). For more information about using this diagnostic, see Signal Ranges in Simulink User's Guide.

Configuration Management

Disabled Library Link Management

The following new features help manage disabled library links and protect against accidental loss of work:

- “Disabled Link” appears in the title bar of a Model Editor window that displays a subsystem connected to a library by a disabled link.
- ToolTips for library-linked blocks include the link status as well as the destination block for the link.
- New diagnostics warn when saving a model that contains disabled or parameterized library links.
- New Model Advisor checks let you search for disabled or parameterized library links in a model.

See [Disabling Links to Library Blocks](#) in *Simulink User's Guide* for more information.

Model Dependencies Tools

The model dependencies manifest tools have these new capabilities:

- Enhanced analysis to detect file dependencies from Stateflow transitions, Embedded MATLAB functions, and requirements documents. See [Scope of Dependency Analysis](#) in *Simulink User's Guide*.
- Model dependencies tools now save user manifest edits for reuse the next time a manifest is generated. See [Edit Manifests](#) in *Simulink User's Guide*.

Embedded Software Design

Legacy Code Tool Enhancement

The Legacy Code Tool has been enhanced to allow the use of `void*` and `void**` to declare variables that represent memory allocated for specific instances of items such as file descriptors, device drivers, and memory managed externally.

For more information, see:

- Integrating Existing C Functions into Simulink Models with the Legacy Code Tool in the Developing S-Functions
- `legacy_code` function documentation

Block Enhancements

Product Block Reorders Inputs Internally

In previous releases, a Product block whose

- **Number of inputs** parameter begins with a divide character (/)
- **Multiplication** parameter specifies Element-wise (.*)

computes the reciprocal of its first input before multiplying or dividing by subsequent inputs. For example, if a Product block specifies division for its first input, $u1$, and multiplication for its second input, $u2$, previous versions of Simulink software compute

$$(1 / u1) * u2$$

In this release, the Product block internally reorders its first two inputs if particular conditions apply, such that Simulink software now computes

$$u2 / u1$$

See the Product block documentation for more information.

Block Data Tips Now Work on All Platforms

In previous releases, block data tips worked only on Microsoft Windows platforms. In this release, the data tips work on all platforms. Also, the data tip for a library link, even if disabled, now includes the name of the library block it references.

Enhanced Data Type Support for Blocks

The following blocks now allow you to specify the data type of their outputs:

- Abs
- Multipoint Switch
- Saturation
- Saturation Dynamic
- Switch

The following blocks now support single-precision floating-point inputs, outputs, and parameter values:

- Discrete Filter
- Discrete State-Space
- Discrete Transfer Fcn

New Simulink Data Class Block Object Properties

The following properties have been added to the `Simulink.BlockData` class:

- **AliasedThroughDataType**
- **AliasedThroughDataTypeID**

New Break Link Options for save_system Command

The `save_system` command's `BreakLink` option has been replaced by two options: `BreakAllLinks` and `BreakUserLinks`. The first option duplicates the behavior of the obsolete `BreakLink` option, i.e., it replaces all library links, including links to Simulink block libraries with copies of the referenced library blocks. The `BreakUserLinks` option replaces only links to user-defined libraries.

Version History

The `save_system` command continues to honor the `BreakLink` option but displays a warning message at the MATLAB command line that the option is deprecated.

Simulink Software Checks Data Type of the Initial Condition Signal of the Integrator Block

When the output port of the Constant or IC block is connected to the Initial Condition port of the Integrator block, Simulink software now compares the data type of the Initial Condition input signal of the Integrator block with the **Constant value** parameter or **Initial value** parameter of the Constant block or IC block, respectively.

Version History

If the data type for the output port of the Constant or IC blocks does not match the data type of the Initial Condition input signal for the Integrator block, Simulink software returns an error at compile time.

Usability Enhancements

Model Advisor

Model Advisor has been enhanced to navigate checks, display status, and report results. Also, this release contains a new Model Advisor Checks reference.

Alignment Commands

This release contains new block alignment, distribution, and resize commands to align groups of blocks along their edges, equalize interblock spacing, and resize blocks to be all the same size. See *Aligning, Distributing, and Resizing Groups of Blocks Automatically* in Simulink User's Guide for more information.

S-Functions

New S-Function APIs to Support Singleton Dimension Handling

The following functions have been added:

- `ssPruneNDMatrixSingletonDims`
- `ssGetInputPortDimensionSize`
- `ssGetOutputPortDimensionSize`

See S-Function SimStruct Functions — Alphabetical List in Developing S-Functions for more information.

New Level-2 M-File S-Function Example

This release includes a new Level-2 M-file S-function example in `sfundemos.mdl`. The Simulink model `msfcn_demo_varpulse.mdl` uses the S-function `msfcn_varpulse.m` to create a variable-width pulse generator.

R2007a+

Version: 6.6.1

Bug Fixes

R2007a

Version: 6.6

New Features

Bug Fixes

Version History

Multidimensional Signals

This release includes support for multidimensional signals, including:

- Sourcing of multidimensional signals
- Logging or displaying of multidimensional signals
- Large-scale modeling applications, such as those from model referencing
- Buses and nonvirtual buses
- Code generation with Real-Time Workshop software
- S-functions, including Level-2 M-File S-functions
- Stateflow charts

For further details, see:

- “Multidimensional Signals in Simulink Blocks” on page 36-2
- “Multidimensional Signals in S-Functions” on page 36-4

Simulink software supports signals with up to 32 dimensions. Do not use signals with more than 32 dimensions.

Multidimensional Signals in Simulink Blocks

The following blocks were updated to support multidimensional signals. See Signal Dimensions for further details.

- Abs
- Assignment
- Bitwise Operator
- Bus Assignment
- Bus Creator
- Bus Selector
- Compare to Constant
- Compare to Zero
- Complex to Magnitude-Angle
- Complex to Real-Imag
- Concatenate
- Constant
- Data Store Memory
- Data Store Read
- Data Store Write
- Data Type Conversion
- MATLAB Function (formally called Embedded MATLAB Function)
- Environment Controller
- From

- From Workspace
- Gain (only if the **Multiplication** parameter specifies Element-wise($K*u$))
- Goto
- Ground
- IC
- Inport
- Level-2 MATLAB S-Function
- Logical Operator
- Magnitude-Angle to Complex
- Manual Switch
- Math Function (no multidimensional signal support for the transpose and hermitian functions)
- Memory
- Merge
- MinMax
- Model
- Multiport Switch
- Outport
- Product, Product of Elements — only if the **Multiplication** parameter specifies Element-wise
- Probe
- Random Number
- Rate Transition
- Real-Imag to Complex
- Relational Operator
- Reshape
- Scope, Floating Scope
- Selector
- S-Function
- Signal Conversion
- Signal Specification
- Slider Gain
- Squeeze
- Subsystem, Atomic Subsystem, CodeReuse Subsystem
- Add, Subtract, Sum, Sum of Elements — along specified dimension
- Switch
- Terminator
- To Workspace
- Trigonometric Function
- Unary Minus
- Uniform Random Number

- Unit Delay
- Width

The Block Support Table does not list which blocks support multidimensional signals. To see if a block supports multidimensional signals, check for the entry `Multidimensionalized` in the **Characteristics** table of a block.

Multidimensional Signals in S-Functions

To use multidimensional signals in S-functions, you must use the new `SimStruct` function, `ssAllowSignalsWithMoreThan2D`.

Multidimensional Signals in Level-2 M-File S-Functions

To use multidimensional signals in Level-2 M-file S-functions, you must set the new `Simulink.MSFcnRunTimeBlock` property, `AllowSignalsWithMoreThan2D`.

New Block Parameters

This release introduces the following common block parameters.

- `PreCopyFcn`: Allows you to assign a function to call before the block is copied. See [Block Callback Parameters](#) for details.
- `PreDeleteFcn`: Allows you to assign a function to call before the block is deleted. See [Block Callback Parameters](#) for details.
- `StaticLinkStatus`: Allows you to obtain the link status of a block without updating out-of-date reference blocks. See [Checking and Setting Link Status Programmatically](#) for details.

GNU Compiler Upgrade

This release upgrades the GNU compiler to GCC 4.0.3 on Mac platforms and GCC 4.1.1 on Linux platforms. The Fortran runtime libraries for the previous GCC 3.x versions are no longer included with MATLAB.

Version History

C, C++, or Fortran MEX-files built with the previous 3.x version of the GCC compiler are not guaranteed to load in this release. Rebuild the source code for these S-functions using the new version of the GCC compiler.

Changes to Concatenate Block

This release includes the following changes to the Concatenate block:

- Its **Mode** parameter provides two settings, namely, `Vector` and `Multidimensional array`.
- Its parameter dialog box contains a new option, **Concatenate dimension**, specifying the output dimension along which to concatenate the input arrays.
- The block displays a new icon when its **Mode** parameter is set to `Multidimensional array`.

This release updates Concatenate blocks when loading models created in previous releases.

Changes to Assignment Block

This release includes the following changes to the Assignment block:

- Enter the number of dimensions in the **Number of output dimensions** parameter, then configure the input and output with the **Index Option**, **Index**, and **Output Size** parameters.
- The parameter dialog box has the following new parameters:
 - **Number of output dimensions**
 - **Index Option**
 - **Index**
 - **Output Size**
- The **Initialize output (Y)** parameter replaces **Output (Y)** and has renamed options.
- The **Action if any output element is not assigned** parameter replaces **Diagnostic if not all required dimensions populated**.
- The block displays a new icon depending on the value of **Number of input dimensions** and the **Index Option** settings.

The following parameters are obsolete:

- **Input type**
- **Use index as start value**
- **Source of element indices**
- **Elements**
- **Source of row indices**
- **Rows**
- **Source of column indices**
- **Columns**
- **Output dimensions**

This release updates Assignment blocks when loading models created in previous releases.

Changes to Selector Block

This release includes the following changes to the Selector block:

- Enter the number of dimensions in the **Number of input dimensions** parameter, then configure the input and output with the **Index Option**, **Index**, and **Output Size** parameters.
- The parameter dialog box has the following new parameters:
 - **Number of input dimensions**
 - **Index Option**
 - **Index**
 - **Output Size**
- The behavior of the **Sample time** parameter has changed. See the Selector block **Sample time** parameter for details.

- The block displays a new icon depending on the value of **Number of input dimensions** and the **Index Option** settings.

The following parameters are obsolete:

- **Input type**
- **Use index as starting value**
- **Source of row indices**
- **Rows**
- **Source of column indices**
- **Columns**
- **Output port dimensions**

This release updates Selector blocks when loading models created in previous releases.

Improved Model Advisor Navigation and Display

This release improves the Model Advisor graphical user interface (GUI) for navigating lists of checks and viewing the status of completed checks. While Model Advisor functionality and content are largely unchanged from R2006b, the Model Advisor checks display and are navigated differently than in previous versions, and the generated Model Advisor report, if requested, displays in a MATLAB web browser window that is separate from the Model Advisor GUI.

To exercise the new features, open Model Advisor for a model (for example, enter `modeladvisor('vdp')` at the MATLAB command line) and then follow the instructions in the Model Advisor window. For more information about Model Advisor navigation and display, see [Consulting the Model Advisor](#).

Change to `Simulink.ModelAdvisor.getModelAdvisor` Method

In this release, when using the `getModelAdvisor` method defined by the `Simulink.ModelAdvisor` class to change Model Advisor working scope to a different model, you must either close the previous model or invoke the `getModelAdvisor` method with `'new'` as the second argument. For example, if you previously set scope to `modelX` with

```
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelX');
```

and you want to change scope to `modelY`, you must either close `modelX` or use

```
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelY', 'new');
```

If you try to change scope between models without the `'new'` argument, an error message is displayed.

Version History

In previous releases, you could change Model Advisor working scope without closing the current session. This is no longer allowed.

If your code contains a code pattern such as the following,

```
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelX');
...
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelY');
```

you must add the 'new' argument to the second and subsequent invocations of `getModelAdvisor`. For example:

```
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelX');
...
Obj = Simulink.ModelAdvisor.getModelAdvisor('modelY', 'new');
```

Alternatively, you can close ModelX before issuing `Simulink.ModelAdvisor.getModelAdvisor('modelY')`.

New Simulink Blocks

This release introduces the following blocks:

- The Permute Dimensions block enables you to rearrange the dimensions of a multidimensional signal.
- The Squeeze block enables you to remove singleton dimensions from a multidimensional signal.

Change to Level-2 MATLAB S-Function Block

If a model includes a Level-2 MATLAB S-Function block, and an error occurs in the S-function, the Level-2 M-File S-Function block will display M-file stack trace information for the error in a dialog box. Click **OK** to remove the dialog box. In previous releases, this block did not display the stack trace information.

Model Dependency Analysis

The model dependencies manifest tools identify files required by your model. You can list required files in a 'manifest' file, package the model with required files into a ZIP file, or compare two file manifests.

See Model Dependencies for more information.

Model File Monitoring

- Warnings if a model file is changed on disk by another user or application while the model is loaded in Simulink software. (see Model File Change Notification in Managing Model Versions).
- Warning to notify the user if multiple models or libraries with the same name exist, as Simulink software may not be using the one the user expects. (see Shadowed Files).

Legacy Code Tool Enhancements

- New fields in the Legacy Code Tool data structure: `InitializeConditionsFcnSpec` and `SampleTime`. `InitializeConditionsFcnSpec` defines a function specification for a reentrant function that the S-function calls to initialize and reset states. `SampleTime` allows you to specify whether sample time is inherited from the source block, represented as a tunable parameter, or fixed.

- Support for state (persistent memory) arguments in registered function specifications.
- Support for complex numbers specified for input, output, and parameter arguments in function specifications. This support is limited to use with Simulink built-in data types.
- Support for multidimensional arrays specified for input and output arguments in function specifications. Previously, multidimensional array support applied to parameters only.
- Ability to apply the `size` function to any dimension of function input data—input, output, parameter, or state. The data type of the `size` function's return value can be any type except complex, bus, or fixed-point.
- A new Legacy Code Tool option, `'backward_compatibility'`, which you can specify with the `legacy_code` function. This option enables Legacy Code Tool syntax, as made available from MATLAB Central in releases prior to R2006b, for a given MATLAB session.
- The following new demos:
 - Specified or Inherited Sample Time with Legacy Functions
 - Persistent Memory within Legacy Functions
 - Complex Signals in Legacy Function
 - Multi-Dimensional Signals in Legacy Functions

For more information, see

- Integrating Existing C Functions into Simulink Models with the Legacy Code Tool in the Writing S-Functions documentation
- Integrate External Code Using Legacy Code Tool in the Real-Time Workshop documentation
- `legacy_code` function reference page

Version History

If you are using a version of the Legacy Code Tool that was accessible from MATLAB Central before R2006b, the syntax for using the tool differs from the syntax currently supported by Simulink software. To continue using the old style syntax, for example, `legacy_code_initialize.m`, issue the following call to `legacy_code` for a given MATLAB session:

```
legacy_code('backward_compatibility');
```

Continuous State Names

State names can now be assigned in those blocks that employ continuous states. The names are assigned with the `ContinuousStateAttributes` Block-Specific Parameters parameter, or in the Blocks Parameter dialog box.

The following blocks support continuous state names:

- Integrator
- State-Space
- Transfer Fcn
- Variable Transport Delay
- Zero-Pole

Logging of continuous states is illustrated in the `sldemo_hydrod` demo.

Changes to Embedded MATLAB Function Block

This release introduces the following changes to the Embedded MATLAB Function block:

- “New Function Checks M-Code for Compliance with Embedded MATLAB Subset” on page 36-9
- “Support for Multidimensional Arrays” on page 36-9
- “Support for Function Handles” on page 36-9
- “Enhanced Support for Frames” on page 36-9
- “New Embedded MATLAB Runtime Library Functions” on page 36-9
- “Using & and | Operators in Embedded MATLAB Function Blocks” on page 36-11
- “Calling get Function from Embedded MATLAB Function Blocks” on page 36-11
- “Documentation on Embedded MATLAB Subset has Moved” on page 36-11

New Function Checks M-Code for Compliance with Embedded MATLAB Subset

Embedded MATLAB function blocks introduce a new function, Embedded MATLAB MEX (emlmex), that checks M-code for compliance with the syntax and semantics of the Embedded MATLAB subset. You can add Embedded MATLAB-compliant code to Embedded MATLAB Function blocks and Truth Table blocks in Simulink models. For more information, see “Working with Embedded MATLAB MEX” in the Embedded MATLAB documentation.

Support for Multidimensional Arrays

Embedded MATLAB Function blocks now support multidimensional signals and parameter data, where the number of dimensions can be greater than 2. This feature is fully integrated with support for multidimensional signals in Simulink software. Supported functions in the Embedded MATLAB Run-Time Function Library have been enhanced to handle multidimensional data.

Support for Function Handles

Embedded MATLAB Function blocks now support function handles for invoking functions indirectly and parameterizing operations that you repeat frequently in your code. For more information, see the section on using function handles in About Code Generation from MATLAB Algorithms.

Enhanced Support for Frames

Embedded MATLAB Function blocks can now input and output frame-based signals directly in Simulink models. You no longer need to attach Frame Conversion blocks to inputs and outputs to achieve this functionality. See Working with Frame-Based Signals.

New Embedded MATLAB Runtime Library Functions

Embedded MATLAB Function blocks provide 31 new runtime library functions in the following categories:

- “Casting Functions” on page 36-10
- “Derivative and Integral Functions” on page 36-10
- “Discrete Math Functions” on page 36-10
- “Exponential Functions” on page 36-10
- “Filtering and Convolution Functions” on page 36-10

- “Logical Operator Functions” on page 36-10
- “Matrix and Array Functions” on page 36-10
- “Polynomial Functions” on page 36-10
- “Set Functions” on page 36-10
- “Specialized Math” on page 36-11
- “Statistical Functions” on page 36-11

See the Embedded MATLAB Run-Time Function Library for a list of all supported functions.

Casting Functions

- `typecast`

Derivative and Integral Functions

- `cumtrapz`
- `trapz`

Discrete Math Functions

- `nchoosek`

Exponential Functions

- `expm`

Filtering and Convolution Functions

- `conv2`
- `deconv`
- `detrend`
- `filter2`

Logical Operator Functions

- `xor`

Matrix and Array Functions

- `cat`
- `flipdim`
- `normest`
- `rcond`
- `sortrows`

Polynomial Functions

- `poly`

Set Functions

- `issorted`

Specialized Math

- beta
- betainc
- beta1n
- ellipke
- erf
- erfc
- erfcinv
- erfcx
- erfinv
- expint
- gamma
- gammainc
- gamma1n

Statistical Functions

- mode

Using & and | Operators in Embedded MATLAB Function Blocks

Embedded MATLAB Function blocks no longer support & and | operators in `if` and `while` conditional statements.

Version History

In prior releases, these operators compiled without error, but their short-circuiting behavior was not implemented correctly. Substitute `&&` and `||` operators instead.

Calling get Function from Embedded MATLAB Function Blocks

Embedded MATLAB Function blocks now support the Simulink Fixed Point `get` function for returning the properties of `fi` objects.

Version History

To get properties of *non-fixed-point* objects in Embedded MATLAB Function blocks, you must first declare `get` to be an extrinsic function; otherwise, your code will error. For more information refer to Calling MATLAB Functions.

Documentation on Embedded MATLAB Subset has Moved

Documentation on the Embedded MATLAB subset and its syntax, semantics, and supported functions has moved out of the Simulink Reference. See Code Generation from MATLAB User's Guide for the new Embedded MATLAB documentation.

Referenced Models Support Non-Zero Start Time

The simulation start time of all models in a model reference hierarchy was previously required to be 0. Now the simulation start time can be nonzero. The start time of all models in a model reference hierarchy must be the same. See Referencing a Model for information about these capabilities. See “Referencing Configuration Sets” on page 36-13 for information about a convenient way to give all models in a hierarchy the same configuration parameters, including simulation start time.

New Functions Copy a Model to a Subsystem or Subsystem to Model

Two new functions exist that you can use to copy contents between a block diagram and a subsystem.

`Simulink.BlockDiagram.copyContentsToSubSystem`

Copies the contents of a block diagram to an empty subsystem.

`Simulink.SubSystem.copyContentsToBlockDiagram`

Copies the contents of a subsystem to an empty block diagram.

For details, see the reference documentation for each function.

New Functions Empty a Model or Subsystem

Two new functions exist that you can use to delete the contents of a block diagram or subsystem.

`Simulink.BlockDiagram.deleteContents`

Deletes the contents of a block diagram.

`Simulink.SubSystem.deleteContents`

Deletes the contents of a subsystem.

For details, see the reference documentation for each function.

Default for Signal Resolution Parameter Has Changed

In the Configuration Parameters dialog, **Diagnostics > Data Validity** pane, the default setting for **Signal resolution** is now **Explicit only**. Previously, the default was **Explicit and warn implicit**. Equivalently, the default value of the `SignalResolutionControl` parameter is now `UseLocalSettings` (previously `TryResolveAllWithWarnings`). See [Diagnostics Pane: Data Validity](#) for more information.

Version History

Due to this change, labeling a signal is no longer enough to cause it to resolve by default to a signal object. You must also do one of the following:

- In the signal's Signal Properties dialog, select **Signal name must resolve to Simulink data object** and specify a `Simulink.Signal` object in the **Signal name** field. Simulink software then resolves that signal to that signal object.
- In the Configuration Parameters dialog, set **Diagnostics > Data Validity > Signal resolution** to **Explicit and warn implicit** (to post warnings) or **Explicit and implicit** (to suppress warnings). Simulink software then resolves all labeled signals to signal objects by matching their names, posting a warning of each resolution if so directed.

Models built in R2007a will default to **Explicit only**. Models created in previous versions will retain the **Signal resolution** value with which they were saved, and will run as they did before. New models may therefore behave differently from existing models that retain the previous default behavior. To specify the previous default behavior in a new model, change **Signal resolution** to **Explicit and warn implicit**.

Conversion Function

MathWorks discourages using implicit signal resolution except for fast prototyping, because implicit resolution slows performance, complicates model validation, and can have nondeterministic effects. Simulink software provides the `disableimplicitsignalresolution` function, which you can use to change settings throughout a model so that it does not use implicit signal resolution. See the function's reference documentation, or type:

```
help disableimplicitsignalresolution
```

in the MATLAB Command Window.

Referencing Configuration Sets

This release provides configuration references (`Simulink.ConfigSetRef` class), which you can use to link multiple models to a configuration set stored on the base workspace. All of those models then share the same configuration set, and therefore have the same configuration parameter values. Changing a parameter value in a shared configuration set changes that value for every model that uses the set. With configuration references, you can:

- Assign the same configuration set to any number of models
- Replace the configuration sets of any number of models without changing the model files
- Use different configuration sets for a referenced model in different contexts without changing the model file

See [Manage a Configuration Set](#) and [Manage a Configuration Reference](#) for more information.

Version History

You cannot change configuration parameter values by operating directly on a configuration reference as you can a configuration set. Instead, you must use the configuration reference to retrieve the configuration set and operate on the set. If you reconfigure a model to access configuration parameters using a configuration reference, you must update any scripts that change parameter values to incorporate the extra step of obtaining the configuration set from the reference before changing the values. See [Create a Freestanding Configuration Set at the Command Line](#) for details.

New Block, Model Advisor Check, and Utility Function for Bus to Vector Conversion

When the diagnostic **Configuration Parameters > Connectivity > Buses > Bus signal treated as vector** is disabled or **none**, you can input a homogeneous virtual bus to many blocks that accept vectors but are not formally defined as accepting buses. Simulink software transparently converts the bus to a vector, allowing the block to accept the bus.

However, MathWorks discourages intermixing buses and vectors, because such mixtures cause ambiguities that preclude strong type checking. The practice may become unsupported at some future time, and should not be used in new applications.

Simulink software provides diagnostics that report cases where buses are mixed with vectors, and includes capabilities that you can use to upgrade a model to eliminate such mixtures, as described in the following sections of the Simulink documentation:

- Using Composite Signals — A new chapter in R2007a that describes the specification and use of composite signals.
- Avoiding Mux/Bus Mixtures — Ambiguities that arise when composite signal types are intermixed, and the tools available for eliminating such mixtures.
- Using Diagnostics for Mux/Bus Mixtures — Two diagnostic options for detecting mixed composite signals: Mux blocks used to create bus signals and Bus signal treated as vector.
- Using the Model Advisor for Mux/Bus Mixtures — Model Advisor checks that detect mixed composite signals and recommend alternatives.
- Bus to Vector — A block that you can insert into a bus used implicitly as a vector to explicitly convert the bus to a vector.
- Simulink.BlockDiagram.addBusToVector — A function that creates a report of every bus used implicitly as a vector, and optionally inserts a Bus to Vector block into every such bus, replacing the implicit use with an explicit conversion.

Enhanced Support for Tunable Parameters in Expressions

Expressions that index into tunable parameters, such as $P(1)+P(2)/P(i)$, retain their tunability in generated code, including simulation code that is generated for a referenced model. Both the indexed parameter and the index itself can be tuned.

Parameter expressions of the form $P(i)$ retain their tunability if all of the following are true:

- The index i is a constant or variable of `double` datatype
- P is a 1D array, or a 2D array with one row or one column, of `double` datatype
- P does not resolve to a mask parameter, but to a variable in the model or the base workspace

New Loss of Tunability Diagnostic

Previously, any loss of tunability generated a warning. In R2007a, you can use the **Loss of Tunability** diagnostic to control whether loss of tunability is ignored or generates a warning or error. See Detect loss of tunability for details.

Port Parameter Evaluation Has Changed

Previously, resolution of port parameters of a masked subsystem began within the subsystem, which could violate the integrity of the mask. For example, if a subsystem mask defines parameter A , and a port of the subsystem uses A to set some port attribute, resolving A by starting within the masked block makes A externally visible, though it should be visible only within the mask.

To fix this problem, in R2007a masked subsystem port parameter resolution starts in the containing system rather than within the masked subsystem, then proceeds hierarchically upward as it did

before. This change preserves the integrity of the masked subsystem, but can change model behavior if any subsystem port previously depended for resolution on a variable defined within the mask.

Version History

A model whose ports did not reference variables defined within a mask are unaffected. A model that resolved any port parameter by accessing a variable within a masked block may behave differently or become vulnerable to future changes in behavior, as follows:

- If the port parameter's value cannot be evaluated, because the evaluation would require access to a variable defined only within the mask, an error occurs.
- If an appropriate variable exists outside the mask but has a different value than the corresponding variable within the mask, no error occurs, but model behavior may change.
- If an appropriate variable exists and has the same value inside and outside the mask, no behavioral change occurs, but later changes to the variable outside the mask may have unexpected effects.

To ensure correct results, change the model as needed so that any port parameter that previously depended on any variables defined within a mask give the intended results using the new resolution search path.

Data Type Objects Can Be Passed Via Mask Parameters

Previously, if a masked subsystem contained a block that needed to specify a data type using a data type object, the block could access the object only in the base workspace. The data type object could *not* be passed into the subsystem through a mask parameter. Parameterizing data types used by blocks under a mask was therefore not possible.

To support parameterized data types inside masked subsystems, you can now use a mask parameter to pass a data type object into a subsystem. Blocks in the subsystem can then use the object to specify data types under the mask.

Expanded Options for Displaying Subsystem Port Labels

This release provides an expanded set of options for displaying port labels on a subsystem block. The options include displaying:

- The label on the corresponding port block
- The name of the corresponding port block
- The name of the signal connected to the corresponding block

See the documentation for the **Show Port Labels** option on the Subsystem block's parameter dialog box for more information.

Model Explorer Customization Option Displays Properties of Selected Object

This release introduces a **Selection Properties** node to the Model Explorer's **Customize Contents** pane. The node allows you to customize the Model Explorer's **Contents** pane to display

only the properties of the currently selected object. See [The Model Explorer: Overview](#) for more information.

Change to PaperPositionMode Parameter

In this release, when exporting a diagram as a graphic with the `PaperPositionMode` model parameter set to `auto`, Simulink software sizes the exported graphic to be the same size as the diagram's image on the screen when viewed at normal size. When `PaperPositionMode` is set to `manual`, Simulink software sizes the exported image to have the height and width specified by the model's `PaperPosition` parameter.

Version History

In previous releases, a model's `PaperPosition` parameter determined the size of the exported graphic regardless of the setting of the model's `PaperPositionMode` parameter. To reproduce the behavior of previous releases, set the `PaperPositionMode` parameter to `manual`.

New Simulink.Bus.objectToCell Function

A new function, `Simulink.Bus.objectToCell`, is available for converting bus objects to a cell array that contains bus information. For details, see the description of `Simulink.Bus.objectToCell`.

Simulink.Bus.save Function Enhanced To Allow Suppression of Bus Object Creation

The `Simulink.Bus.save` function has been enhanced such that when using the `'cell'` format you have the option of suppressing the creation of bus objects when the saved M-file executes. To suppress bus object creation, specify the optional argument `'false'` when you execute the saved M-file.

For more detail, see the description of `Simulink.Bus.save`.

Change in Version 6.5 (R2006b) Introduced Incompatibility

A change introduced in Version 6.5 (R2006b) introduces an incompatibility between this release and releases preceding Version 6.5 (R2006b). See [“Attempting to Reference a Symbol in an Uninitialized Mask Workspace Generates an Error”](#) on page 37-5 for more information.

Nonverbose Output During Code Generation

Simulink Accelerator now defaults to nonverbose output when generating code. A new parameter, `AccelVerboseBuild`, has been added to control how much information is displayed. See [Customizing the Build Process](#) for more information.

SimulationMode Removed From Configuration Set

Previously, the `SimulationMode` property was attached to the configuration set for a model. In R2007a, the property has been removed from the configuration set. Now you set the simulation mode

for the model using the **Simulation** menu in the model window or the `set_param` function with the `SimulationMode` model parameter.

Version History

Using `Simulink.ConfigSet.SimulationMode` is not recommended. Use `set_param(modelName, 'SimulationMode', value)` instead.

R2006b

Version: 6.5

New Features

Bug Fixes

Version History

Model Dependency Viewer

The Model Dependency Viewer displays a dependency view of a model that shows models and block libraries directly or indirectly referenced by the model. The dependency view allows you to quickly determine your model's dependencies on referenced models and block libraries. See Model Dependencies for more information.

Enhanced Lookup Table Blocks

This release replaces the PreLookup Index Search and Interpolation (n-D) Using PreLookup blocks with two new blocks: Prelookup and Interpolation Using Prelookup. The new blocks provide fixed-point arithmetic, consistency checking, more efficient code generation, and other enhancements over the blocks they replace.

Version History

MathWorks plans on obsoleting the PreLookup Index Search and Interpolation (n-D) Using PreLookup blocks in a future release. In the meantime, MathWorks will continue to support and enhance these blocks. For example, this release improves the precision with which the PreLookup Index Search block computes its fraction value if its **Index search method** parameter specifies `Evenly Spaced Points`.

We recommend that you use the Prelookup and Interpolation Using Prelookup blocks for all new model development.

Legacy Code Tool

The Legacy Code Tool generates an S-function from existing C code and specifications that you supply. It enables you to transform your C functions into C MEX S-functions for inclusion in a Simulink model. See Integrating Existing C Functions into Simulink Models with the Legacy Code Tool in Developing S-Functions for more information.

Simulink Software Now Uses Internal MATLAB Functions for Math Operations

In previous releases, Simulink software used the host compiler's C++ Math Library functions to perform most mathematical operations on floating-point data. Some of those functions produced results that were slightly inconsistent with MATLAB results. In this release, Simulink software calls the same internal routines that MATLAB calls for most trigonometric, exponential, and rounding and remainder operations involving floating-point data. This ensures that when Simulink and MATLAB products operate on the same platform, they produce the same numerical results.

In particular, Simulink software now performs mathematical operations with the same internal functions that MATLAB uses to implement the following M-functions:

- `sin`, `cos`, `tan`
- `asin`, `acos`, `atan`, `atan2`
- `sinh`, `cosh`, `tanh`
- `asinh`, `acosh`, `atanh`

- `log`, `log2`, `log10`
- `mod`, `rem`
- `power`

Note By default, in this release Real-Time Workshop software continues to use C Math Library functions in the code that it generates from a Simulink model.

Enhanced Integer Support in Math Function Block

The `sqrt` operation in the Math Function block now supports built-in integer data types.

Configuration Set Updates

This release includes the following changes to model configuration parameters and configuration sets.

- This release includes a new command, `openDialog`, that displays the **Configuration Parameters** dialog box for a specified configuration set. This command allows display of configuration sets that are not attached to any model.
- The `attachConfigSet` command now includes an `allowRename` option that determines how the command handles naming conflicts when attaching a configuration set to a model.
- This release includes a new `attachConfigSetCopy` command that attaches a copy of a specified configuration set to a model.
- The new **Sample hit time adjusting** diagnostic controls whether Simulink software notifies you when the solver has to adjust a sample time specified by your model to solve the model. The associated model parameter is `TimeAdjustmentMsg`.
- The default value of the **Multitask data store** diagnostic has changed from `Warning` to `Error` for new models. This change does not affect existing models.
- The name of the **Block reduction optimization** parameter has changed to **Block reduction**.

Command to Initiate Data Logging During Simulation

The command

```
set_param(bdroot, 'SimulationCommand', 'WriteDataLogs')
```

writes all logging variables during simulation. See [Exporting Signal Data Using Signal Logging](#) for more information.

Commands for Obtaining Model and Subsystem Checksums

This release includes commands for obtaining model and subsystem checksums.

- `Simulink.BlockDiagram.getChecksum`

Get checksum for a model. Simulink Accelerator software uses this checksum to control regeneration of simulation targets. You can use this command to diagnose target rebuild problems.

- `Simulink.SubSystem.getChecksum`

Get checksum for a subsystem. Real-Time Workshop software uses this checksum to control reuse of code generated from a subsystem that occurs more than once in a model. You can use the checksum to diagnose code reuse problems.

Sample Hit Time Adjusting Diagnostic

The **Sample hit time adjusting** diagnostic controls whether Simulink software notifies you when the solver has to adjust a sample time specified by your model to solve the model. The associated model parameter is `TimeAdjustmentMsg`.

Function-Call Models Can Now Run Without Being Referenced

This release allows you to simulate a function-call model, i.e., a model that contains a root-level function-call trigger block, without having to reference the model. In previous releases, the function-call model had to be referenced by another model in order to be simulated.

Signal Builder Supports Printing of Signal Groups

This release adds printing options to the Signal Builder block's editor. It allows you to print waveforms displayed in the editor to a printer, file, the clipboard, or a figure window. For details, see [Printing, Exporting, and Copying Waveforms](#).

Method for Comparing Simulink Data Objects

This release introduces an `isContentEqual` method for Simulink data objects that allows you to determine whether a Simulink data object has the same property values as another Simulink data object. For more information, see [Comparing Data Objects](#).

Unified Font Preferences Dialog Box

In this release, the **Simulink Preferences** dialog box displays font settings for blocks, lines, and annotations on a single pane instead of on separate tabbed panes as in previous releases. This simplifies selection of font preferences.

Limitation on Number of Referenced Models Eliminated for Single References

In previous releases, all distinct models referenced in a model hierarchy counted against the limitation imposed by Microsoft Windows on the number of distinct referenced models that can occur in a hierarchy. In this release, models configured to be instantiable only once do not account against this limit. This means that a model hierarchy can reference any number of distinct models on Windows platforms as long as they are referenced only once and are configured to be instantiable only once (see [Model Referencing Limitations](#) for more information).

Parameter Objects Can Now Be Used to Specify Model Configuration Parameters

This release allows you to use `Simulink.Parameter` objects to specify model configuration as well as block parameters. For example, you can specify a model's fixed step size as `Ts` and its stop time as `20*Ts` where `Ts` is a workspace variable that references a parameter object. When compiling a model, Simulink software replaces a reference to a parameter object in a model configuration parameter expression with the object's value.

Version History

In previous releases, you could use expressions of the form `p.Value()`, where `p` references a parameter object, in model configuration parameter expressions. Such expressions cause expression evaluation errors in this release when you compile a model. You should replace such expressions with a simple reference to the parameter object itself, i.e., replace `p.Value()` with `p`.

Parameter Pooling Is Now Always Enabled

In previous releases, the **Parameter Pooling** optimization was optional and was enabled by default. Due to internal improvements, disabling **Parameter Pooling** would no longer be useful in any context. The optimization is therefore part of standard R2006b operation, and has been removed from the user interface.

Version History

Upgrading a model to R2006b inherently provides the effect that enabling **Parameter Pooling** did in previous releases. No compatibility considerations result from this change. If the optimization was disabled in an existing model, a warning is generated when the model is first upgraded to R2006b. This warning requires no action and can be ignored.

Attempting to Reference a Symbol in an Uninitialized Mask Workspace Generates an Error

In this release, attempting to reference a symbol in an uninitialized mask workspace generates an error. This can happen, for example, if a masked subsystem's initialization code attempts to set a parameter of a block that resides in a masked subsystem in the subsystem being initialized and one or more of the block's parameters reference variables defined by the mask of the subsystem in which it resides (see Initialization Command Limitations for more information).

Version History

In this release, updating or simulating models created in previous releases may generate unresolvable symbol error messages. This can happen if the model contains masked subsystems whose initialization code sets parameters on blocks residing in lower-level masked subsystems residing in the top-level masked subsystem. To eliminate these errors, change the initialization code to avoid the use of `set_param` commands to set parameters in lower-level masked subsystems. Instead, use mask variables in upper-level masked subsystems to specify the values of parameters of blocks residing in lower-level masked subsystems. See Defining Mask Parameters for information on using mask variables to specify block parameter values.

Changes to Integrator Block's Level Reset Options

This release changes the behavior of the `level` reset option of the Integrator block. In releases before Simulink 6.3, the `level` reset option resets the integrator's state if the reset signal is nonzero or changes from nonzero in the previous time step to zero in the current time step. In Simulink 6.3, 6.4, and 6.4.1, the option resets the integrator only if the reset signal is nonzero. This release restores the `level` reset behavior of releases that preceded Simulink 6.3. It also adds a `level hold` option that behaves like the `level` reset option of Simulink 6.3, 6.4, and 6.4.1.

Version History

A model that uses the `level` reset option could produce results that differ in this release from those produced in Simulink 6.3, 6.4, and 6.4.1. To reproduce the results of previous releases, change the model to use the new `level hold` option instead.

Embedded MATLAB Function Block Features and Changes

Support for Structures

You can now define structures as inputs, outputs, local, and persistent variables in Embedded MATLAB Function blocks. With support for structures, Embedded MATLAB Function blocks give you the ability to read and write Simulink bus signals at inputs and outputs of Embedded MATLAB Function blocks. See “Using Structures” in the Embedded MATLAB documentation.

Embedded MATLAB Editor Analyzes Code with M-Lint

The Embedded MATLAB Editor uses the MATLAB M-Lint Code Analyzer to automatically check your Embedded MATLAB function code for errors and recommend corrections. The editor displays an M-Lint bar that highlights offending lines of code and displays Embedded MATLAB diagnostics as well as MATLAB messages. See “Using M-Lint with Embedded MATLAB” in the Embedded MATLAB documentation.

New Embedded MATLAB Runtime Library Functions

Embedded MATLAB Function blocks provide 36 new runtime library functions in the following categories:

- “Data Analysis” on page 37-6
- “Discrete Math” on page 37-7
- “Exponential” on page 37-7
- “Interpolation and Computational Geometry” on page 37-7
- “Linear Algebra” on page 37-7
- “Logical” on page 37-8
- “Specialized Plotting” on page 37-8
- “Transforms” on page 37-8
- “Trigonometric” on page 37-8

Data Analysis

- `cov`

- `ifftshift`
- `std`
- `var`

Discrete Math

- `gcd`
- `lcm`

Exponential

- `expm1`
- `log10`
- `log1p`
- `log2`
- `nextpow2`
- `nthroot`
- `reallog`
- `realpow`
- `realsqrt`

Interpolation and Computational Geometry

- `cart2pol`
- `cart2sph`
- `pol2cart`
- `sph2cart`

Linear Algebra

- `cond`
- `det`
- `ipermute`
- `kron`
- `permute`
- `planerot`
- `rand`
- `randn`
- `rank`
- `shiftdim`
- `squeeze`
- `subspace`
- `trace`

Logical

- `isstruct`

Specialized Plotting

- `histc`

Transforms

- `bitrevorder`

Trigonometric

- `hypot`

New Requirement for Calling MATLAB Functions from Embedded MATLAB Function Blocks

To call external MATLAB functions from Embedded MATLAB Function blocks, you must first declare the functions to be extrinsic. (External MATLAB functions are functions that have not been implemented in the Embedded MATLAB runtime library.) MATLAB Function blocks do not compile or generate code for extrinsic functions; instead, they send the function to MATLAB for execution during simulation. There are two ways to call MATLAB functions as extrinsic functions in Embedded MATLAB Function blocks:

- Use the new construct `eml.extrinsic` to declare the function extrinsic
- Call the function using `feval`

For details, see [Calling MATLAB Functions](#).

Version History

Currently, Embedded MATLAB Function blocks use implicit rules to handle calls to external functions:

- For simulation, Embedded MATLAB Function blocks send the function to MATLAB for execution
- For code generation, Embedded MATLAB Function blocks check whether the function affects the output of the Embedded MATLAB function in which it is called. If there is no effect on output, Embedded MATLAB Function blocks proceed with code generation, but exclude the function call from the generated code. Otherwise, Embedded MATLAB Function blocks generate a compiler error.

In future releases, Embedded MATLAB Function blocks will apply these rules only to external functions that you call as extrinsic functions. Otherwise, they will compile external functions by default, potentially causing unpredictable behavior or generating errors. For reliable simulation and code generation, MathWorks recommends that you call external MATLAB functions as extrinsic functions.

Type and Size Mismatch of Values Returned from MATLAB Functions Generates Error

Embedded MATLAB Function blocks now generate an error if the type and size of a value returned by a MATLAB function does not match the predeclared type and size.

Version History

In previous releases, Embedded MATLAB Function blocks attempted to silently convert values returned by MATLAB functions to predeclared data type and sizes if a mismatch occurred. Now, such mismatches always generate an error, as in this example:

```
x = int8(zeros(3,3)); % Predeclaration
x = eval('5'); % Calls MATLAB function eval
```

This code now generates an error because the Embedded MATLAB function predeclares `x` as a 3-by-3 matrix, but MATLAB function returns `x` as a scalar double. To avoid errors, reconcile predeclared data types and sizes with the actual types and sizes returned by MATLAB function calls in your Embedded MATLAB Function blocks.

Embedded MATLAB Function Blocks Cannot Output Character Data

Embedded MATLAB Function blocks now generate an error if any of its outputs is character data.

Version History

In the previous release, Embedded MATLAB Function blocks silently cast character array outputs to `int8` scalar arrays. This behavior does not match MATLAB, which represents characters in 16-bit unicode.

R2006a+

Version: 6.4.1

No New Features or Changes

R2006a

Version: 6.4

New Features

Version History

Signal Object Initialization

This release introduces the use of signal objects to specify initial values for signals and states. This allows you to initialize signals or states in the model, not just those generated by blocks that have initial condition or value parameters. For details, see *Using Signal Objects to Initialize Signals and Discrete States* in the online Simulink documentation.

Icon Shape Property for Logical Operator Block

The Logical Operator block's parameter dialog box contains a new property, **Icon shape**, settings for which can be either **rectangular** or **distinctive**. If you select **rectangular** (the default), the block appears as it does in previous releases. If you select **distinctive**, the block appears as the IEEE® standard graphic symbol for the selected logic operator.

Data Type Property of Parameter Objects Now Settable

This release allows you to set the data type of a Simulink.Parameter object via either its **Value** property or via its **Data type** property. In previous releases, you could specify the data type of a parameter object only by setting the object's **Value** property to a typed value expression.

Range-Checking for Parameter and Signal Object Values

This release introduces range checking for Simulink.Parameter and Simulink.Signal objects. Simulink software checks whether a parameter's **Value** or a signal's **Initial value** falls within the values you specify for the object's **Minimum** and **Maximum** properties. If not, Simulink software generates a warning or error.

Version History

Previous releases ignored such violations since the **Minimum** and **Maximum** properties were intended for use in documenting parameter and signal objects. In this release, Simulink software displays a warning if you load a parameter object or a signal object does not specify a valid range or its value falls outside the specified range. If you get such a warning, change the parameter or signal object's **Value** or **Minimum** or **Maximum** values so that the **Value** falls within a valid range.

Expanded Menu Customization

The previous release of Simulink software allows you to customize the Simulink editor's **Tools** menu. This release goes a step further and allows you to customize any Simulink (or Stateflow) editor menu (see *Customizing the Simulink User Interface* in the online Simulink documentation).

Bringing the MATLAB Desktop Forward

The Model Editor's **View** menu includes a new command, **MATLAB Desktop**, that brings the MATLAB desktop to the front of the windows displayed on your screen.

Converting Atomic Subsystems to Model References

This release adds a command, **Convert to Model Block**, to the context (right-click) menu of an atomic subsystem. Selecting this command converts an atomic subsystem to a model reference. See Atomic Subsystem and Converting a Subsystem to a Referenced Model for more information.

The function `sl_convert_to_model_reference`, which provided some of the same capabilities as **Convert to Model Block**, is obsolete and has been removed from the documentation. The function continues to work, so no incompatibility arises, but it posts a warning when called. The function will be removed in a future release.

Concatenate Block

The new Concatenate block concatenates its input signals to create a single output signal whose elements occupy contiguous locations in memory. The block typically uses less memory than the Matrix Concatenation block that it replaces, thereby reducing model memory requirements.

Version History

This release replaces obsolete Matrix Concatenation blocks with Concatenate blocks when loading models created in previous releases.

Model Advisor Changes

Model Advisor Tasks Introduced

This release introduces Model Advisor tasks for referencing models and upgrading a model to the current version of Simulink software. See Consulting the Model Advisor in the online Simulink documentation for more information.

Model Advisor API

This release introduces an application program interface (API) that enables you to run the Model Advisor from the MATLAB command line or from M-file programs. For example, you can use the API to create M-file programs that determine whether a model passes selected Model Advisor checks whenever you open, simulate, or generate code from the model. See Running the Model Advisor Programmatically in the online Simulink documentation for more information.

Built-in Block's Initial Appearance Reflects Parameter Settings

In this release, when you load a model containing nonmasked, built-in blocks whose appearance depends on their parameter settings, such as the Selector block, the appearance of the blocks reflect their parameter settings. You no longer have to update the model to update the appearance of such blocks.

Version History

In previous releases, model or block callback functions that use `set_param` to set a built-in, nonmasked block's parameters could silently put the block in an unusable state. In this release, such callbacks will trigger error messages if they put blocks in an unusable state.

Double-Click Model Block to Open Referenced Model

In this release, double-clicking a Model block that specifies a valid referenced model opens the referenced model, rather than the Block Parameters dialog box as in previous releases. To open the Block Parameters dialog box, choose **Model Reference Parameters** from the **Context** or **Edit** menu. See Navigating a Model Block for details.

Signal Logs Reflect Bus Hierarchy

In this release, signal logs containing buses reflect the structure of the buses themselves instead of flattening bus data as in previous releases (see `Simulink.TsArray`).

Tiled Printing

This release introduces a tiled printing option that allows you to distribute a block diagram over multiple pages. You can control the number of pages over which Simulink software distributes the block diagram, and hence, the total size of the printed image. See Tiled Printing in the online Simulink documentation for more information.

Solver Diagnostic Controls

In this release, the **Configuration Parameters** dialog box includes the following enhancements:

- The **Diagnostics** pane contains a new diagnostic, **Consecutive zero crossings violation**, that alerts you if Simulink software detects the maximum number of consecutive zero crossings allowed. You can specify the criteria that Simulink software uses to trigger this diagnostic using two new **Solver diagnostic controls** on the **Solver** pane:
 - **Consecutive zero crossings relative tolerance**
 - **Number of consecutive zero crossings allowed**

For more information, see Preventing Excessive Zero Crossings in the online Simulink documentation.

- The **Solver** pane contains a new solver diagnostic control, **Number of consecutive min step size violations allowed**, that Simulink software uses to trigger the **Min step size violation** diagnostic (see Number of consecutive min steps in the online Simulink documentation).

Diagnostic Added for Multitasking Conditionally Executed Subsystems

This release adds a sample-time diagnostic that detects an enabled subsystem in multitasking solver mode that operates at multiple rates or a conditionally executed subsystem that contain an asynchronous subsystem. Such subsystems can cause corrupted data or non-deterministic behavior in a real-time system using code generated from the model. See the documentation for the **Multitask Conditionally Executed Subsystem** diagnostic for more information.

Embedded MATLAB Function Block Features and Changes

Option to Disable Saturation on Integer Overflow

The properties dialog for Embedded MATLAB Function blocks provides a new **Saturate on Integer Overflow** check box that lets you disable saturation on integer overflow to generate more efficient

code. When you enable saturation on integer overflow, Embedded MATLAB Function blocks add additional checks in the generated code to detect integer overflow or underflow. Therefore, it is more efficient to disable this option if your algorithm does not rely on overflow behavior. For more information, see MATLAB Function Block Properties in the online Simulink documentation.

Nontunable Option Allows Use of Parameters in Constant Expressions

The **Data** properties dialog for the MATLAB Function (formally called Embedded MATLAB Function) block provides a new **Tunable** check box that lets you specify the tunability (see Tunable Parameters in the online Simulink documentation) of a workspace variable or mask parameter used as data in Embedded MATLAB code. The option is checked by default. Unchecking the option allows you to use a workspace variable or mask parameter as data wherever Embedded MATLAB requires a constant expression, such as a dimension argument to the `zeros` function. For more information, see Adding Data to a MATLAB Function Block in the online Simulink documentation.

Enhanced Support for Fixed-Point Arithmetic

Embedded MATLAB Function blocks support the new fixed-point features introduced in Version 1.4 (R2006a) of the Fixed-Point Toolbox software, including [Slope Bias] scaling (see Specifying Simulink Fixed Point Data Properties in the online Simulink documentation). For information about the features added to the Simulink Fixed Point software, see Fixed-Point Toolbox Release Notes.

Support for Integer Division

Embedded MATLAB Function blocks support the new function `idivide`, which performs integer division with a variety of rounding options. It is recommended that the rounding option used for integer division in Embedded MATLAB Function blocks match the rounding option in the parent Simulink model.

The default rounding option for `idivide` is `'fix'`, which rounds toward zero. This option corresponds to the choice **Zero** in the submenu for **Signed integer division rounds to:**, a parameter that you can set in the Hardware Implementation Pane of the Configuration Parameters dialog in Simulink software (see Hardware Implementation Pane in the online Simulink documentation). If this parameter is set to **Floor** in the Simulink model that contains the Embedded MATLAB Function block, it is recommended that you pass the rounding option `'floor'` to `idivide` in the block.

For a complete list of Embedded MATLAB runtime library functions provided in this release, see “New Embedded MATLAB Runtime Library Functions” on page 39-5.

New Embedded MATLAB Runtime Library Functions

Embedded MATLAB Function blocks provide new runtime library functions in the following categories:

- “Integer Arithmetic” on page 39-6
- “Linear Algebra” on page 39-6
- “Logical” on page 39-6
- “Polynomial” on page 39-6
- “Trigonometric” on page 39-6

Integer Arithmetic

- `idivide`

Linear Algebra

- `compan`
- `dot`
- `eig`
- `fliplr`
- `flipud`
- `freqspace`
- `hilb`
- `ind2sub`
- `invhilb`
- `linspace`
- `logspace`
- `magic`
- `median`
- `meshgrid`
- `pascal`
- `qr`
- `rot90`
- `sub2ind`
- `toeplitz`
- `vander`
- `wilkinson`

Logical

- `isequal`
- `isinteger`
- `islogical`

Polynomial

- `polyfit`
- `polyval`

Trigonometric

- `acosd`
- `acot`
- `acotd`
- `acoth`

- `acsc`
- `acscd`
- `acsch`
- `asec`
- `asecd`
- `asech`
- `asind`
- `atand`
- `cosd`
- `cot`
- `cotd`
- `coth`
- `csc`
- `cscd`
- `csch`
- `sec`
- `secd`
- `sech`
- `sind`
- `tand`

Setting FIMATH Cast Before Sum to False No Longer Supported in Embedded MATLAB Function Blocks

You can no longer set the FIMATH property `CastBeforeSum` to `false` for fixed-point data in Embedded MATLAB Function blocks.

Version History

The reason for the restriction is that Embedded MATLAB Function blocks do not produce the same numerical results as MATLAB when `CastBeforeSum` is `false`. In the previous release, Embedded MATLAB Function blocks set `CastBeforeSum` to `false` by default for the default FIMATH object. If you have existing models that contain Embedded MATLAB Function blocks in which `CastBeforeSum` is `false`, you will get an error when you compile or update your model. To correct the issue, you must set `CastBeforeSum` to `true`. To automate this process, you can run the utility `supdate` either from the Model Advisor or by typing the following command at the MATLAB command line:

```
supdate ('modelName')
```

where `'modelName'` is the name of the model containing the Embedded MATLAB Function block that generates the error. `supdate` prompts you to update this property by selecting one of these options:

Option	Action
Yes	Updates the first occurrence of <code>CastBeforeSum=false</code> in Embedded MATLAB Function blocks in the offending model and then prompts you for each subsequent one found in the model.
No	Does not update any occurrences of <code>CastBeforeSum=false</code> in the offending model.
All	Updates all occurrences of <code>CastBeforeSum=false</code> in the offending model.

Note `supdate` detects `CastBeforeSum=false` only in *default* FIMATH objects defined for Simulink software signals in Embedded MATLAB Function blocks. If you modified the FIMATH object in an Embedded MATLAB Function block, update `CastBeforeSum` manually in your model and fix the errors as they are reported.

Type Mismatch of Scalar Output Data in Embedded MATLAB Function Blocks Generates Error

Embedded MATLAB Function blocks now generate an error if the output type inferred by the block does not match the type you explicitly set for a scalar output.

Version History

In previous releases, a silent cast was inserted from the computed type to the set type when mismatches occurred. In most cases, you should not need to set the output type for Embedded MATLAB Function blocks. When you do, insert an explicit cast in your Embedded MATLAB script. For example, suppose you declare a scalar output `y` to be of type `int8`, but its actual type is `double`. Replace `y` with a temporary variable `t` in your script and then add the following code:

```
y = int8(t);
```

Implicit Parameter Type Conversions No Longer Supported in Embedded MATLAB Function Blocks

Embedded MATLAB Function blocks now generate an error if the type of a parameter inferred by the block does not match the type you explicitly set for the parameter.

Version History

In the previous release, if the type you set for a parameter did not match the actual parameter value, Embedded MATLAB Function blocks implicitly cast the parameter to the specified type. Now you receive a compile-time error when type mismatches occur for parameters defined in Embedded MATLAB Function blocks.

There are two workarounds:

- Change the scope of the data from **Parameter** to **Input**. Then, connect to the input port a **Constant** block that brings in the parameter and casts it to the desired type.
- Cast the parameter inside your Embedded MATLAB function to the desired type.

Fixed-Point Parameters Not Supported

Embedded MATLAB Function blocks generate a compile-time error if you try to bring a `fi` object defined in the base workspace into Embedded MATLAB Function blocks as a parameter.

There are two workarounds:

- Change the scope of the data from **Parameter** to **Input**. Then, connect to the input port a **Constant** block that brings in the parameter and casts it to fixed-point type.
- Cast the parameter inside your Embedded MATLAB function to fixed-point type.

Embedded MATLAB Function Blocks Require C Compiler for Windows 64

No C compiler ships with MATLAB and Simulink products on Windows 64. Because Embedded MATLAB Function blocks perform simulation through code generation, you must supply your own MEX-supported C compiler to use these blocks. The C compilers available at the time of this writing for Windows 64 include Microsoft Visual Studio 2005 and the Microsoft Platform SDK.

R14SP3

Version: 6.3

New Features

Version History

Model Referencing

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 Subsystems in the Simulink documentation 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 Real-Time Workshop software to generate a standalone executable (Real-Time Workshop target) for the model. See Model Referencing Limitations for information on other limitations.

Referenced Models Without Root I/O Can Inherit Sample Times

Previous releases of Simulink software 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 software 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 Viewing a Model Reference Hierarchy 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` in the Simulink Reference documentation 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 `democs` 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 software 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 `` 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 40-4).

Parameter Object Expressions No Longer Supported in Dialog Boxes

Version History

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.

Modeling Enhancements

Annotations

This release introduces the following enhancements to model annotations:

- Annotation properties dialog box (see [Annotations Properties Dialog Box](#) in the Simulink documentation)
- Annotation callback functions (see [Annotation Callback Functions](#) in the Simulink documentation)
- Annotation application programming interface (see [Annotations API](#) in the Simulink documentation)

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 [Visualizing and Comparing Simulation Results](#) in the Simulink documentation 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` in the Simulink Reference documentation 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 Function Blocks

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 MATLAB functions. For more information, see [Code Acceleration and Code Generation from MATLAB for Fixed-Point Algorithms](#) in the Fixed-Point Toolbox documentation.

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 Function Blocks

Embedded MATLAB Function blocks now support input triggers and function-call outputs. See Ports and Data Manager in the Simulink documentation for more information.

Find Options Added to the 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 in the Simulink documentation for more information.

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

Version History

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 built-in data types and `Simulink.NumericType` objects to specify the data types of `Simulink.Signal` objects. See the `Simulink.Signal` documentation 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` in the Simulink Reference documentation) and time-series array (see `Simulink.TSArray` in the Simulink Reference documentation) objects residing in the MATLAB workspace. See Importing MATLAB timeseries Data in the Simulink documentation 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 software 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 release adds the following simulation diagnostics:

- Enforce sample times specified by Signal Specification blocks in the online Simulink documentation
- Extraneous discrete derivative signals in the online Simulink documentation
- Detect read before write in the online Simulink documentation
- Detect write after read in the online Simulink documentation
- Detect write after write in the online Simulink documentation

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 Diagnostics Pane: Data Validity in the online Simulink documentation 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 software 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 [Viewing Command History](#) in the Simulink documentation)
- Keyboard shortcuts for panning model views (see [Model Viewing Shortcuts](#) in the Simulink documentation)

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 [Disabling and Hiding Dialog Box Controls](#) in the Simulink documentation)
- Disable, or hide widgets on Simulink dialog boxes (see [Disabling and Hiding Dialog Box Controls](#) in the Simulink documentation)

MEX-Files

MEX-Files on Windows Systems

In this release, the extension for files created by the MATLAB `mex` command on Windows systems has changed from `dll` to `mexw32` or `mexw64`.

Version History

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

MEX-File Extension Changed

In this release, the extension for files created by the MATLAB `mex` command has changed from `dll` to `mexw32` (and `mexw64`).

Version History

If you use a `mexopts.bat` file other than the one created by the `mex` command in your MATLAB preferences directory to build Accelerator targets, you should recreate the file from the `mexopts.bat` template that comes with this release.

R14SP2

Version: 6.2

New Features

Version History

Multiple Signals on Single Set of Axes

Viewers can now display multiple signals on a single set of axes.

Logging Signals to the MATLAB Workspace

Viewers can now log the signals that they display to the MATLAB base workspace. See [Exporting Signal Data Using Signal Logging](#) for more information.

Legends that Identify Signal Traces

Viewers can now display a legend that identifies signal traces.

Displaying Tic Labels

Viewers can now display tic labels both inside and outside scope axes.

Opening Parameters Dialog Box

You can open a viewer's parameters dialog box by right-clicking on the viewer scope.

Rootlevel Input Ports

Version History

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.