

Simulink Release Notes

How to Contact MathWorks



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



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



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

Simulink Release Notes

© COPYRIGHT 2000–2014 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	2
Annotations with rich text, graphics, and hyperlinks	2
Diagnostic Viewer to collect information, warnings, and error messages	2
Option to bring contents of a hierarchical subsystem into the parent subsystem with one click	3
Support for native OS touch gestures, such as pinch-to-zoom and panning	3
Operating system print options for models	4
Preference for line crossing style	4
Scalable graphics output to clipboard for Macintosh	4
Sliders, dials, and spinboxes available as parameter controls in masks	4
 Component-Based Modeling	 5
Option to choose default variants	5
Option to choose variants that differ in number of input and output ports	5
Advisor-based workflow for converting subsystems to Model blocks	5
Single-model workflow for algorithm partitioning and targeting of multicore processors and FPGAs	6
Easier MATLAB System block creation via autocompletion and browsing for System object names	6
Improved algebraic loop handling and reduced data copies with the Bus Selector block	6
Faster response time when opening bus routing block dialog boxes and propagating signal labels	7
Usability enhancements to configure a model for concurrent execution on a target	7
Default setting of Underspecified initialization detection diagnostic is Simplified	8
Discrete-Time Integrator block has dialog box changes for initialization	9
System objects Propagates mixin methods	9

Simulation Analysis and Performance	10
Reduced setup and build time for Model blocks when using Rapid Accelerator mode	10
Performance Advisor checks that validate overall performance improvement for all suggested changes and set code generation option for MATLAB System block ..	11
Improved navigation of the Performance Advisor report ..	11
Block behavior for asynchronous initiator with constant sample time	11
Global setting for validation of checks in Performance Advisor	12
Guided setup in Performance Advisor	12
Project and File Management	13
Branching support through Git source control	13
Comparison of project dependency analysis results	13
Impact graph layout algorithm improved for easier identification of top models and their dependencies	13
Impact analysis example for finding and running impacted tests	14
Performance improvements for common scripting operations such as adding and removing files and labels	14
Conflict resolution tools to extract conflict markers	14
Updated Power Window Example	14
Data Management	16
Data dictionary for defining and managing design data associated with models	16
Rapid Accelerator mode signal logging enhanced to avoid rebUILds and to support buses and referenced models ..	16
Simplified tuning of all parameters in referenced models ..	17
Simulink.findVars supported in referenced models	18
Saving workspace variables and their values to a MATLAB script	18
Frame-based signals in the To Workspace block	19
Simulation mode consistency for Data Import/Export pane output options parameter	20
Dimension mismatch handling for root Inport blocks improved	20
Simulink.Bus.createObject support for structures of timeseries objects	21
Signal logging override for model reference variants	21

Improved To Workspace block default for fixed-point data	21
Legacy Code Tool support for 2-D row-major matrix	22
Model Explorer property name filtering refined	22
Connection to Educational Hardware	23
Support for Arduino Due hardware	23
Support for Arduino WiFi Shield hardware	23
Support for LEGO MINDSTORMS EV3 hardware	24
Updates to support for LEGO MINDSTORMS NXT hardware	24
Support for Samsung GALAXY Android devices	25
Block Enhancements	27
Enumerated data types in the Direct Lookup Table (n-D) block	27
Improved performance and code readability in linear search algorithm for Prelookup and n-D Lookup Table blocks	27
System object file templates	27
Relay block output of fixed-in-minor-step continuous signal for continuous input	27
MATLAB Function Blocks	28
Generating Simulation Target typedefs for imported bus and enumerated data types	28
Complex data types in data stores	28
Unicode character support for MATLAB Function block names	28
Support for int64 and uint64 data types in MATLAB Function blocks	28
Streamlined MEX compiler setup and improved troubleshooting	29
Code generation for additional Image Processing Toolbox functions	29
Code generation for additional Signal Processing Toolbox, Communications System Toolbox, and DSP System Toolbox functions and System objects	29
Code generation for MATLAB fminsearch optimization function, additional interpolation functions, and additional interp1 and interp2 interpolation methods	30
Code generation for fread function	31

Enhanced code generation for switch statements	31
Code generation for value classes with <code>set.prop</code> methods	32
Code generation error for properties that use <code>AbortSet</code> attribute	32
Toolbox functions for code generation	32
Modeling Guidelines	35
Modeling guidelines for high-integrity systems	35
Model Advisor	36
Improved navigation of the Model Advisor report, including a navigation pane, collapsible content, and filters based on check status	36
Option to run Model Advisor checks in the background ...	36
Upgrade Advisor check for <code>get_param</code> calls for block <code>CompiledSampleTime</code>	36
Upgrade Advisor check for signal logging in Rapid Accelerator mode	37

R2013b

New Simulink Editor	40
Ability to add rich controls, links, and images to customized block interfaces using the Mask Editor	40
Content preview for subsystems and Stateflow charts	40
Comment-through capability to temporarily delete blocks and connect input signals to output signals	41
New options added to <code>find_system</code> command	41
Visual cues for signal lines that cross	41
UTF-16 character support for block names, signal labels, and annotations in local languages	42
Unified Print Model dialog box for printing	43
Block Parameters dialog box access from Block Properties dialog box	44
Notification bar icon indicator for multiple notifications ..	44
Component-Based Modeling	45

MATLAB System block for including System objects in Simulink models	45
Variant Manager that manages all the variants in a model in one place	45
Improved componentization capabilities for modeling scheduling diagrams with root-level function-call inports	45
Array of buses signal logging in model reference accelerator mode	45
Ability to add, delete, and move input signals within Bus Creator block	46
Streamlined approach to migrating from Classic to Simplified initialization mode	46
Simplified display of sorted execution order	46
Enhanced model reference rebuild algorithm for MATLAB Function blocks	46
Simulation Analysis and Performance	47
LCC compiler included on Windows 64-bit platform for running simulations	47
Signal logging in Rapid Accelerator mode	47
Performance Advisor checks for Rapid Accelerator mode and data store memory diagnostics	47
Long long integers in simulation targets for faster simulation on Win64 machines	48
Auto-insertion of rate transition block	49
Compiled sample time for multi-rate blocks returns cell array of all sample times	49
Improvement to model reference parallel build check in Performance Advisor	52
Improved readability in Performance Advisor reports	52
Simulation Data Inspector launch using simplot command	52
Project and File Management	53
Impact analysis by exploring modified or selected files to find dependencies	53
Option to export impact analysis results to the workspace, batch processing, or image files	53
Identification of requirements documents during project dependency analysis	54
Simplified label creation by dragging a label onto files in any view	54

Shortcut renaming, grouping, and execution from any view using the Toolstrip	54
Data Management	56
Streamlined selection of one or more signals for signal logging	56
Simplified modeling of single-precision designs	56
Connection status visualization and connection method customization for root inport mapping	58
Conversion of numeric variables into Simulink.Parameter objects	59
Model Explorer search options summary hidden by default	59
Simulink.DualScaledParameter class	59
Legacy data type specification functions return numeric objects	60
Root Inport Mapping Error Messages	63
Root inport mapping example	63
Connection to Educational Hardware	64
Ability to run models on target hardware from the Simulink toolbar	64
Support for Arduino hardware available on Mac OS X ...	65
Support for Arduino Ethernet Shield and Arduino Nano 3.0 hardware	65
Signal Management	67
Port number display to help resolve error messages	67
Enforced bus diagnostic behavior	67
Block Enhancements	69
Improved performance of LUT block intermediate calculations	69
Name changes that unify storage class parameters	69
Warnings when using old parameter names with spaces ..	69
Strictly monotonically increasing time values on Repeating Sequence block	70
pow function in Math function block that supports Code Replacement Library (CRL)	70
Continuous Linear Block improvements, such as diagnostics, readability, and stricter checks	70

MATLAB Function Blocks	72
Code generation for Statistics Toolbox and Phased Array System Toolbox	72
Toolbox functions for code generation	72
External C library integration using coder.ExternalDependency	73
Updating build information using coder.updateBuildInfo	73
Conversion of MATLAB expressions into C constants using coder.const	73
Highlighting of constant function arguments in the compilation report	74
coder.target syntax change	74
LCC compiler included on Windows 64-bit platform for running simulations	74
Modeling Guidelines	75
Modeling guidelines for high-integrity systems	75
Model Advisor	76
Collapsible content within Model Advisor reports	76
Reorganization of Model Advisor checks	76
Check for strict single precision usage	76

R2013a

New Simulink Editor	78
Reordering of tabs in tabbed windows	78
Scalable vector graphics for mask icons	78
Simulation Stepper Default Value Change	78
Component-Based Modeling	79
Direct active variant control via logical expressions	79
Live update for variant systems and commented-out blocks	79
Masking of linked library blocks	79
Target profiling for concurrent execution to visualize task execution times and task-to-core assignment	79

Incremental block-to-task mapping workflow support enabled by automatic block-to-task assignment for multicore execution on embedded targets	80
PIL and SIL modes for concurrent execution	80
Parameterized task periods for concurrent execution	80
Relaxed configuration parameter setting requirements ...	80
Connection to Educational Hardware	81
Support for Gumstix Overo hardware	81
Support for Raspberry Pi hardware	82
Blocks for GPIO, LED, and eSpeak Text to Speech on BeagleBoard and PandaBoard	83
Blocks for Compass and IR Receiver sensors on LEGO MINDSTORMS NXT	83
Project and File Management	85
Simplified scripting interface for automating Simulink Project tasks	85
Option to use elements from multiple templates when creating a new project	85
Saving and reloading of dependency analysis results	85
Robust loading of projects with conflicted metadata project definition files	86
New project preferences to control logging and warnings ..	86
Data Management	87
Fixed-Point Advisor support for model reference	87
Arrays of buses loading and logging	87
Root Inport Mapping tool changes	87
New Root Inport Mapping Examples	88
Level-1 data classes not supported	88
Simulink data type classes do not support inexact enumerated property value matching	89
Simulation Analysis and Performance	91
Simulation Performance Advisor report that shows both check results and actions taken	91
Improved simulation performance when stepping back is enabled	91
Simulation Data Inspector run-configuration options for names and placement in run list	91
Arrays of buses displayed in Simulation Data Inspector ..	91

Simulation Data Inspector overwrite run specification . . .	91
Signal Management	93
Referenced models sample times	93
Triggered subsystem sample times	93
Simulation of variable-size scalar signals	94
Block Enhancements	95
CORDIC approximation method for atan2 function of Trigonometric Function block	95
Product and Gain blocks support Basic Linear Algebra Subprogram (BLAS) library	95
Performance Advisor check for Delay block circular buffer setting	95
MATLAB Function Blocks	96
Masking of MATLAB Function blocks to customize appearance, parameters, and documentation	96
File I/O function support	96
Support for nonpersistent handle objects	97
Include custom C header files from MATLAB code	97
Load from MAT-files	97
coder.opaque function enhancements	98
Complex trigonometric functions	98
Support for integers in number theory functions	98
Enhanced support for class property initial values	99
Default use of Basic Linear Algebra Subprograms (BLAS) Libraries	100
New toolbox functions supported for code generation	101
Function being removed	102
Modeling Guidelines	103
Modeling Guidelines for High-Integrity Systems	103
MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow	103
Model Advisor	104
Model Advisor checks reorganized in a future release	104
Model Advisor navigation between Upgrade Advisor, Performance Advisor, and Code Generation Advisor	104
Report	104

Preferences dialog box	105
By Product folder not displayed	105

R2012b

New Simulink Editor	108
Tabbed windows and automatic window reuse to minimize window clutter	108
Smart signal routing that determines the simplest signal line path without overlapping blocks	109
Explorer bar to help with navigating through a model	110
Simulation stepper to simulate and rewind a model one step at a time	110
Ability to comment out blocks	110
Subsystem badges to identify and look under masked subsystems	111
Reorganized menu to fit common Model-Based Design workflow	111
Palette for commonly used actions	112
Panning and zooming	112
Display of overlapping blocks	112
Unification of Simulink and Stateflow Editors	113
Simulink Editor preferences	113
Toolbar and status bar control	114
Visual editing based on model objects	114
Improved callback error handling	114
Simulink Editor Changes	115
Connection to Educational Hardware	132
Support for Arduino and PandaBoard hardware	132
Bluetooth download to LEGO MINDSTORMS NXT hardware	133
Performance	134
Simulation Performance Advisor that analyzes your model and provides advice on how to increase simulation performance	134
Project and File Management	135

Simulink default file format SLX that uses the OPC standard	135
Simulink Upgrade Advisor to help migrate files to the current release	136
Built-in SVN adapter for Simulink Projects that provides connectivity to SVN and support for server-based repositories	137
Simulink Project Tool dependency graph that provides highlights by file type, dependency type, and label	137
Redesigned graphical tool for efficient Simulink Projects workflow	138
Batch operation support for files in a Simulink Project ...	139
Create and open recent Simulink Projects from MATLAB	139
Block Enhancements	140
Menu item to convert configurable and normal subsystems to variant subsystems	140
Masking improvements, including the ability to reuse masks, delete existing masks on blocks, and use the shortcut operator in mask callback code	140
Default output data type of Logic blocks changed to boolean	141
Signal Attributes tab of dialog box for Operator blocks renamed to Data Type	141
Parameter name changes for Unit Delay block	141
New variants of Delay block in Discrete library	142
Some Probe block parameters no longer support boolean data type	143
Internationalization of block dialog box titles and buttons and block tooltips	144
Enabled and triggered subsystems	144
Data Management	145
Variable Editor access from within Model Explorer	145
Logged simulation data from Simulation Data Inspector accessible from Simulink toolbar	146
Specify verifySignalAndModelPaths action	147
Import and map data to root-level input ports	147
Dataset signal logging format for increased flexibility and ease of use	147
Data type field displays user-defined data types	148
Simulink.VariableUsage to get variable information ...	149

Customizable line specification in Simulation Data Inspector	149
Simulation Data Inspector report includes harness model information	149
Component-Based Modeling	150
Model configuration for targets with multicore processors	150
New <code>Simulink.GlobalDataTransfer</code> class	150
Reduced memory usage in models with many library links	151
Configuration Reference dialog box to propagate and undo configuration settings to all referenced models	151
Context-dependent function-call subsystem input handling improved	152
<code>Simulink.Variant</code> object and the model <code>InitFcn</code>	154
Signal Management	155
Sample time propagation changes	155
Signal Builder	155
User Interface Enhancements	157
Model Advisor Dashboard	157
Show partial or whole model hierarchy contents	157
Improved icons for model objects	159
Simulink Debugger	159
Model Advisor Checks	160
Verify Syntax of Library Models	160
MATLAB Function Blocks	161
New toolbox functions supported for code generation	161
New System objects supported for code generation	162

R2012a

Component-Based Modeling	166
---------------------------------------	------------

Interactive Library Forwarding Tables for Updating	
Links	166
Automatic Refresh of Links and Model Blocks	166
Model Configuration for Targets with Multicore	
Processors	167
MATLAB Function Blocks	169
Integration of MATLAB Function Block Editor into	
MATLAB Editor	169
Code Generation for MATLAB Objects	169
Specification of Custom Header Files Required for	
Enumerated Types	169
Data Management	170
New Infrastructure for Extending Simulink Data Classes	
Using MATLAB Class Syntax	170
Change in Behavior of isequal	172
isContentEqual Will Be Removed in a Future Release ...	172
Change in Behavior of int32 Property Type	173
RTWInfo Property Renamed	173
deepCopy Method Will Be Removed in a Future Release ..	173
New Methods for Querying Workspace Variables	174
Default Package Specification for Data Objects	174
Simulink.Parameter Enhancements	174
Custom Storage Class Specification for Discrete States on	
Block Dialog Box	175
Enhancement to set_param	175
Simulink.findVars Support for Active Configuration	
Sets	177
Bus Support for To File and From File Blocks	177
Bus Support for To Workspace and From Workspace	
Blocks	177
Logging Fixed-Point Data to the To Workspace Block	177
Improved Algorithm for Best Precision Scaling	178
Enhancement of Mask Parameter Promotion	178
File Management	180
SLX Format for Model Files	180
Simulink Project Enhancements	182
Signal Management	184
Signal Hierarchy Viewer	184

Signal Label Propagation Improvements	184
Frame-Based Processing: Inherited Option of the Input Processing Parameter Now Provides a Warning	185
Logging Frame-Based Signals	187
Frame-Based Processing: Model Reference and Using slupdate	187
Removing Mixed Frameness Support for Bus Signals on Unit Delay and Delay	188
Block Enhancements	189
Delay Block Accepts Buses and Variable-Size Signals at the Data Input Port	189
n-D Lookup Table Block Has New Default Settings	189
Blocks with Discrete States Can Specify Custom Storage Classes in the Dialog Box	190
Inherited Option of the Input Processing Parameter Now Provides a Warning	190
User Interface Enhancements	193
Model Advisor: Highlighting	193
Model Explorer: Grouping Enhancements	193
Model Explorer: Row Filter Button	194
Simulation Data Inspector Enhancements	195
Port Value Displays	196
Modeling Guidelines	197
Modeling Guidelines for High-Integrity Systems	197
MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow	197
Execution on Target Hardware	198
New Feature for Running Models Directly from Simulink on Target Hardware	198

R2011b

Simulation Performance	200
-------------------------------------	-----

Accelerator Mode Now Supports Algebraic Loops	200
Component-Based Modeling	201
For Each Subsystem Support for Continuous Dynamics ..	201
Enable Port as an Input to a Root-Level Model	201
Finder Option for Looking Inside Referenced Models	202
Improved Detection for Rebuilding Model Reference Targets	202
Model Reference Target Generation Closes Unneeded Libraries	202
Concurrent Execution Support	202
Finer Control of Library Links	203
Mask Built-In Blocks with the Mask Editor	204
Parameter Checking in Masked Blocks	204
Menu Options to Control Variants	205
MATLAB Function Blocks	206
Simulation Supported When the Current Folder Is a UNC Path	206
Simulink Data Management	207
Default Design Minimum and Maximum are [],[], Not -inf/inf	207
Bus Elements Now Have Design Minimum and Maximum Properties	207
Compiled Design Minimum and Maximum Values Exposed on Block Inport and Outport	208
Back-Propagated Minimum and Maximum of Portion of Wide Signal Are Now Ignored	209
Easier Importing of Signal Logging Data	209
Partial Specification of External Input Data	210
Command-Line Interface for Signal Logging	210
Access to the Data Import/Export Pane from the Signal Logging Selector	211
Inexact Property Names for User-Defined Data Objects Will Not Be Supported in a Future Release	211
Alias Types No Longer Supported with the slDataTypeAndScale Function	211
Simulink.StructType Objects Will Not Be Supported in a Future Release	212
Old Block-specific Data Type Parameters No Longer Supported	212

Simulink.Signal and Simulink.Parameter Will Not Accept Input Arguments	212
Data Import/Export Pane Changes	213
Simulation Data Inspector Tool Replaces Time Series Tool	213
Simulink File Management	214
Project Management	214
Simulink Signal Management	215
Signal Conversion Block Enhancements	215
Environment Controller Block Support for Non-Bus Signals	216
Sample Time Propagation Changes	216
Frame-Based Processing	217
Block Enhancements	220
New Delay Block That Upgrades the Integer Delay Block	220
Sqrt and Reciprocal Sqrt Blocks Support Explicit Specification of Intermediate Data Type	223
Discrete Zero-Pole Block Supports Single-Precision Inputs and Outputs	224
n-D Lookup Table Block Supports Tunable Table Size ...	225
Boolean Output Data Type Support for Logic Blocks	226
Derivative Block Parameter Change	226
User Interface Enhancements	227
Model Explorer: First Two Columns in Contents Pane Remain Visible	227
Model Explorer: Subsystem Code View Added	227
Model Explorer: New Context Menu Options for Model Configurations	227
Simulation Data Inspector Enhancements	229
Conversion of Error and Warning Message Identifiers ...	229
New Modeling Guidelines	231
Modeling Guidelines for High-Integrity Systems	231
Modeling Guidelines for Code Generation	232

Simulation Performance	234
Restore SimState in Models Created in Earlier Simulink Versions	234
Improved Absolute Tolerance Implementation	234
Component-Based Modeling	235
Refreshing Linked Blocks and Model Blocks	235
Enhanced Model Block Displays Variant Model Choices ..	235
Creating a Protected Model Using the Simulink Editor ...	236
MATLAB Function Blocks	237
Embedded MATLAB Function Block Renamed as MATLAB Function Block	237
Support for Buses in Data Store Memory	237
Simulink Data Management	238
Signal Logging Selector	238
Dataset Format Option for Signal Logging Data	238
From File Block Supports Zero-Crossing Detection	239
Signal Builder Block Now Supports Virtual Bus Output ..	240
Signal Builder Block Now Shows the Currently Active Group	240
signalbuilder Function Change	240
Range-Checking Logic for Fixed-Point Data During Simulation Improved	240
Data Object Wizard Now Supports Boolean, Enumerated, and Structured Data Types for Parameters	242
Error Now Generated When Initialized Signal Objects Back Propagate to Output Port of Ground Block	242
No Longer Able to Set RTWInfo or CustomAttributes Property of Simulink Data Objects	242
Global Data Stores Now Treat Vector Signals as One or Two Dimensional	243
No Longer Able to Use Trigger Signals Defined as Enumerations	244
Conversions of Simulink.Parameter Object Structure Field Data to Corresponding Bus Element Type Supported for double Only	245

Simulink.CustomParameter and Simulink.CustomSignal Data Classes To Be Deprecated in a Future Release . . .	245
Parts of Data Class Infrastructure No Longer Available . .	246
Simulink Signal Management	248
Data Store Support for Bus Signals	248
Accessing Bus and Matrix Elements in Data Stores	248
Array of Buses Support for Permute Dimensions, Probe, and Reshape Blocks	249
Using the Bus Editor to Create Simulink.Parameter Objects and MATLAB Structures	249
Block Enhancements	251
Lookup Table, Lookup Table (2-D), and Lookup Table (n-D) Blocks Replaced with Newer Versions in the Simulink Library	251
Magnitude-Angle to Complex Block Supports CORDIC Algorithm and Fixed-Point Data Types	257
Trigonometric Function Block Supports Complex Exponential Output	258
Shift Arithmetic Block Supports Specification of Bit Shift Values as Input Signal	259
Multiple Lookup Table Blocks Enable Removal of Range-Checking Code	260
Enhanced Dialog Layout for the Prelookup and Interpolation Using Prelookup Blocks	262
Product of Elements Block Uses a Single Algorithm for Element-Wise Complex Division	264
Sign Block Supports Complex Floating-Point Inputs	265
MATLAB Fcn Block Renamed to Interpreted MATLAB Function Block	265
Environment Controller Block Port Renamed from RTW to Coder	265
Block Parameters on the State Attributes Tab Renamed . .	266
Block Parameters and Values Renamed for Lookup Table Blocks	266
Performance Improvement for Single-Precision Computations of Elementary Math Operations	267
Dead Zone Block Expands the Region of Zero Output	267
Enhanced PID Controller Blocks Display Compensator Formula in Block Dialog Box	268
Ground Block Always Has Constant Sample Time	268
New Function-Call Feedback Latch Block	268

Output Driving Merge Block Does Not Require IC in Simplified Initializaton Mode	270
Discrete Filter, Discrete FIR Filter, and Discrete Transfer Fcn Blocks Now Have Input Processing Parameter	270
Model Blocks Can Now Use the GetSet Custom Storage Class	271
User Interface Enhancements	272
Model Explorer: Hiding the Group Column	272
Simulation Data Inspector Enhancements	272
Model Advisor	274
Configuration Parameters Dialog Box Changes	275
S-Functions	276
S-Functions Generated with legacy_code function and singleCPPMexFile S-Function Option Must Be Regenerated	276

R2010bSP2

Bug Fixes

R2010bSP1

Bug Fixes

R2010b

Simulation Performance	282
Elimination of Regenerating Code for Rebuilds	282
Component-Based Modeling	283
Model Workspace Is Read-Only During Compilation	283

Support for Multiple Normal Mode Instances of a Referenced Model	283
New Variant Subsystem Block for Managing Subsystem Design Alternatives	284
Support for Bus and Enumerated Data Types on Masks ..	285
sl_convert_to_model_reference Function Removed	285
Verbose Accelerator Builds Parameter Applies to Model Reference SIM Target Builds in All Cases	285
Embedded MATLAB Function Blocks	286
Specialization of Embedded MATLAB Function Blocks in Simulink Libraries	286
Support for Creation and Processing of Arrays of Buses ..	286
Ability to Include MATLAB Code as Comments in Generated C Code	286
Data Properties Dialog Box Enhancements	287
Simulink Data Management	288
Enhanced Support for Bus Objects as Data Types	288
Enhancements to Simulink.NumericType Class	289
Importing Signal Data Sets into the Signal Builder Block	290
signalbuilder Function Changes	290
From File Block Enhancements	291
Finding Variables Used by a Model or Block	291
enumeration Function Replaced With MATLAB Equivalent	292
Programmatic Creation of Enumerations	292
Simulink.Signal and Simulink.Parameter Objects Now Obey Model Data Type Override Settings	292
Simulink File Management	294
Autosave Upgrade Backup	294
Model Dependencies Tools	294
Simulink Signal Management	295
Arrays of Buses	295
Loading Bus Data to Root Input Ports	297
Block Enhancements	298
Prelookup Block Supports Dynamic Breakpoint Data	298

Interpolation Using Prelookup Block Supports Dynamic Table Data	298
Multiport Switch Block Supports Specification of Default Case for Out-of-Range Control Input	298
Switch Block Icon Shows Criteria and Threshold Values ..	298
Trigonometric Function Block Supports Expanded Input Range for CORDIC Algorithm	299
Repeating Sequence Stair Block Supports Enumerated Data Types	300
Abs Block Supports Specification of Minimum Output Value	300
Saturation Block Supports Logging of Minimum and Maximum Values for the Fixed-Point Tool	300
Vector Concatenate Block Now Appears in the Commonly Used and Signal Routing Libraries	300
Model Discretizer Support for Second-Order Integrator Block	300
Integer Delay and Unit Delay Blocks Now Have Input Processing Parameter	301
Data Store Read Block Sample Time Default Changed to -1	302
Support of Frame-Based Signals Being Removed From the Bias Block	302
Relaxation of Limitations for Function-Call Split Block ..	303
User Interface Enhancements	304
Model Explorer and Command-Line Support for Saving and Loading Configuration Sets	304
Model Explorer: Grouping by a Property	304
Model Explorer: Filtering Contents	305
Model Explorer: Finding Variables That Are Used by a Model or Block	305
Model Explorer: Finding Blocks That Use a Variable	306
Model Explorer: Exporting and Importing Workspace Variables	306
Model Explorer: Link to System	307
Lookup Table Editor Can Now Propagate Changes in Table Data to Workspace Variables with Nonstandard Data Format	307
Enhanced Designation of Hybrid Sample Time	307
Inspect Solver Jacobian Pattern	308
Inspection of Values of Elements in Checksum	308
Conversion of Error and Warning Messages Identifiers ...	308

View and Compare Logged Signal Data from Multiple Simulations Using New Simulation Data Inspector Tool	309
Viewing Requirements Linked to Model Objects	309
S-Functions	310
Legacy Code Tool Support for Arrays of Simulink.Bus ...	310
S-Functions Generated with legacy_code function and singleCPPMexFile S-Function Option Must Be Regenerated	310
Level-2 M-File S-Function Block Name Changed to Level-2 MATLAB S-Function	310
Functions Removed	312
Function Being Removed in a Future Release	312

R2010a

Simulation Performance	314
Computation of Sparse and Analytical Jacobian for Implicit Simulink Solvers	314
Sparse Perturbation Support for RSim and Rapid Accelerator Mode	314
Increased Accuracy in Detecting Zero-Crossing Events ...	314
Saving Code Generated by Accelerating Models to slprj Folder	315
Component-Based Modeling	316
Defining Mask Icon Variables	316
For Each Subsystem Block	316
New Function-Call Split Block	317
Trigger Port Enhancements	317
Model Reference Support for Custom Code	318
Embedded MATLAB Function Blocks	319
New Ability to Use Global Data	319
Support for Logical Indexing	319
Support for Variable-Size Matrices in Buses	319

Support for Tunable Structure Parameters	320
Check Box for 'Treat as atomic unit' Now Always Selected	320
Simulink Data Management	321
New Function Finds Variables Used by Models and Blocks	321
MATLAB Structures as Tunable Structure Parameters ..	321
Simulink.saveVars Documentation Added	322
Custom Floating-Point Types No Longer Supported	322
Data Store Logging	323
Models with No States Now Return Empty Variables	323
To File Block Enhancements	324
From File Block Enhancements	325
Root Inport Support for Fixed-Point Data Contained in a Structure	325
Simulink Signal Management	326
Enhanced Support for Proper Use of Bus Signals	326
Bus Initialization	327
S-Functions for Working with Buses	327
Command Line API for Accessing Information About Bus Signals	329
Signal Name Propagation for Bus Selector Block	329
Block Enhancements	330
New Square Root Block	330
New Second-Order Integrator Block	330
New Find Nonzero Elements Block	331
PauseFcn and ContinueFcn Callback Support for Blocks and Block Diagrams	331
Gain Block Can Inherit Parameter Data Type from Gain Value	332
Direct Lookup Table (n-D) Block Enhancements	332
Multiport Switch Block Allows Explicit Specification of Data Port Indices	332
Trigonometric Function Block Supports CORDIC Algorithm and Fixed-Point Data Types	335
Enhanced Block Support for Enumerated Data Types	336
Lookup Table Dynamic Block Supports Direct Selection of Built-In Data Types for Outputs	336
Compare To Zero and Wrap To Zero Blocks Now Support Parameter Overflow Diagnostic	337

Data Type Duplicate Block Enhancement	337
Lookup Table and Lookup Table (2-D) Blocks To Be Deprecated in a Future Release	338
Elementary Math Block Now Obsolete	342
DocBlock Block RTF File Compression	342
Simulink Extras PID Controller Blocks Deprecated	343
User Interface Enhancements	344
Model Explorer Column Views	344
Model Explorer Display of Masked Subsystems and Linked Library Subsystems	345
Model Explorer Object Count	346
Model Explorer Search Option for Variable Usage	346
Model Explorer Display of Signal Logging and Storage Class Properties	346
Model Explorer Column Insertion Options	347
Diagnostics for Data Store Memory Blocks	347
New Command-Line Option for RSim Targets	347
Simulink.SimulationOutput.get Method for Obtaining Simulation Results	347
Simulink.SimState.ModelSimState Class has New snapshotTime Property	348
Simulink.ConfigSet.saveAs to Save Configuration Sets ...	348
S-Functions	349
Building C MEX-Files from Ada and an Example Ada Wrapper	349
New S-Function API Checks for Branched Function-Calls	349
New C MEX S-Function API and M-File S-Function Flag for Compliance with For Each Subsystem	349
Legacy Code Tool Enhanced to Support Enumerated Data Types and Structured Tunable Parameters	350
Documentation Improvements	351
Modeling Guidelines for High-Integrity Systems	351
MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow Included in Help	351

Bug Fixes

R2009b

Simulation Performance	356
Single-Output <code>sim</code> Syntax	356
Expanded Support by Rapid Accelerator	356
SimState Support in Accelerator Mode	356
Integer Arithmetic Applied to Sample Hit Computations ..	356
Improved Accuracy of Variable-Step Discrete Solver	357
Component-Based Modeling	358
Enhanced Library Link Management	358
Enhanced Mask Editor Provides Tabs and Signal Attributes	358
Model Reference Variants	358
Protected Referenced Models	359
Simulink Manifest Tools	360
S-Function Builder	360
Embedded MATLAB Function Blocks	361
Support for Variable-Size Arrays and Matrices	361
Change in Text and Visibility of Parameter Prompt for Easier Use with Fixed-Point Advisor and Fixed-Point Tool	361
New Compilation Report for Embedded MATLAB Function Blocks	361
New Options for Controlling Run-time Checks for Faster Performance	362
Embedded MATLAB Function Blocks Improve Size Propagation Behavior	362
Simulink Data Management	364
New Function Exports Workspace Variables and Values ..	364
New Enumerated Constant Block Outputs Enumerated Data	364

Enhanced Switch Case Block Supports Enumerated Data	364
Code for Multiport Switch Block Shows Enumerated Values	364
Data Class Infrastructure Partially Deprecated	365
Saving Simulation Results to a Single Object	365
Simulation Restart in R2009b	365
Removing Support for Custom Floating-Point Types in Future Release	366
Simulink File Management	367
Removal of Functions	367
Deprecation of SaveAs to R12 and R13	367
Improved Behavior of Save_System	367
Simulink Signal Management	368
Variable-Size Signals	368
Block Enhancements	369
New Turnkey PID Controller Blocks for Convenient Controller Simulation and Tuning	369
New Enumerated Constant Block Outputs Enumerated Data	370
Enhanced Switch Case Block Supports Enumerated Data	370
Code for Multiport Switch Block Shows Enumerated Values	370
Discrete Transfer Fcn Block Has Performance, Data Type, Dimension, and Complexity Enhancements	370
Lookup Table (n-D) Block Supports Parameter Data Types Different from Signal Data Types	371
Reduced Memory Use and More Efficient Code for Evenly Spaced Breakpoints in Prelookup and Lookup Table (n-D) Blocks	372
Math Function Block Computes Reciprocal of Square Root	373
Math Function Block Enhancements for Real-Time Workshop Code Generation	373
Relational Operator Block Detects Signals That Are Infinite, NaN, or Finite	374
Changes in Text and Visibility of Dialog Box Prompts for Easier Use with Fixed-Point Advisor and Fixed-Point Tool	374

Direct Lookup Table (n-D) Block Enhancements	376
Unary Minus Block Enhancements	377
Weighted Sample Time Block Enhancements	378
Switch Case Block Parameter Change	378
Signal Conversion Block Parameter Change	378
Compare To Constant and Compare To Zero Blocks Use New Default Setting for Zero-Crossing Detection	379
Signal Builder Block Change	379
User Interface Enhancements	380
Context-Sensitive Help for Simulink Blocks in the Continuous Library	380
Adding Blocks from a Most Frequently Used Blocks List ..	380
Highlighting for Duplicate Inport Blocks	381
Using the Model Explorer to Add a Simulink.NumericType Object	381
Block Output Display Dialog Has OK and Cancel Buttons	381
Improved Definition of Hybrid Sample Time	381
Find Option in the Model Advisor	382

R2009a

Simulation Performance	384
Saving and Restoring the Complete SimState	384
Save Simulink Profiler Results	384
Component-Based Modeling	385
Port Value Displays in Referenced Models	385
Parallel Builds Enable Faster Diagram Updates for Large Model Reference Hierarchies In Accelerator Mode	385
Embedded MATLAB Function Blocks	387
Support for Enumerated Types	387
Use of Basic Linear Algebra Subprograms (BLAS) Libraries for Speed	387
Data Management	388

Signal Can Resolve to at Most One Signal Object	388
“Signed” Renamed to “Signedness” in the Simulink.NumericType class	388
“Sign” Renamed to “Signedness” in the Data Type Assistant	389
Tab Completion for Enumerated Data Types	389
Simulink File Management	390
Model Dependencies Tools	390
Block Enhancements	391
Prelookup and Interpolation Using Prelookup Blocks Support Parameter Data Types Different from Signal Data Types	391
Lookup Table (n-D) and Interpolation Using Prelookup Blocks Perform Efficient Fixed-Point Interpolations ...	391
Expanded Support for Simplest Rounding Mode to Maximize Block Efficiency	392
New Rounding Modes Added to Multiple Blocks	393
Lookup Table (n-D) Block Performs Faster Calculation of Index and Fraction for Power of 2 Evenly-Spaced Breakpoint Data	395
Discrete FIR Filter Block Supports More Filter Structures	395
Discrete Filter Block Performance, Data Type, Dimension, and Complexity Enhancements	396
MinMax Block Performs More Efficient and Accurate Comparison Operations	397
Logical Operator Block Supports NXOR Boolean Operator	397
Discrete-Time Integrator Block Uses Efficient Integration-Limiting Algorithm for Forward Euler Method	398
Dot Product Block Converted from S-Function to Core Block	398
Pulse Generator Block Uses New Default Values for Period and Pulse Width	399
Random Number, Uniform Random Number, and Unit Delay Blocks Use New Default Values for Sample Time	399
Trigonometric Function Block Provides Better Support of Accelerator Mode	399
Reshape Block Enhanced with New Input Port	400

Multidimensional Signals in Simulink Blocks	400
Subsystem Blocks Enhanced with Read-Only Property That Indicates Virtual Status	401
User Interface Enhancements	402
Port Value Displays in Referenced Models	402
Print Sample Time Legend	402
M-API for Access to Compiled Sample Time Information ..	402
Model Advisor Report Enhancements	402
Counterclockwise Block Rotation	403
Physical Port Rotation for Masked Blocks	403
Smart Guides	403
Customizing the Library Browser's User Interface	403
Subsystem Creation Command	403
Removal of Lookup Table Designer from the Lookup Table Editor	403
S-Functions	405
Level-1 Fortran S-Functions	405

R2008b

Simulation Performance	408
Parallel Simulations in Rapid Accelerator Mode	408
Improved Rebuild Mechanism in Rapid Accelerator Mode	408
Data Type Size Limit on Accelerated Simulation Removed	408
New Initialization Behavior in Conditional, Action, and Iterator Subsystems	409
Component-Based Modeling	410
Processor-in-the-Loop Mode in Model Block	410
Conditionally Executed Subsystem Initial Conditions	410
Model Block Input Enhancement	412
One Parameter Controls Accelerator Mode Build Verbosity	412

Embedded MATLAB Function Blocks	415
Support for Fixed-Point Word Lengths Up to 128 Bits	415
Enhanced Simulation and Code Generation Options for Embedded MATLAB Function Blocks	415
Data Type Override Now Works Consistently on Outputs	415
Improperly-Scaled Fixed-Point Relational Operators Now Match MATLAB Results	416
Data Management	417
Support for Enumerated Data Types	417
Simulink Bus Editor Enhancements	417
New Model Advisor Check for Data Store Memory Usage	417
Simulink File Management	418
Model Dependencies Tools	418
Block Enhancements	419
Trigonometric Function Block	419
Math Function Block	419
Merge Block	419
Discrete-Time Integrator Block	419
Modifying a Link to a Library Block in a Callback Function Can Cause Illegal Modification Errors	419
Random Number Block	420
Signal Generator Block	420
Sum Block	420
Switch Block	421
Uniform Random Number Block	421
User Interface Enhancements	422
Sample Time	422
Model Advisor	422
“What’s This?” Context-Sensitive Help for Commonly Used Blocks	423
Compact Icon Option Displays More Blocks in Library Browser	424
Signal Logging and Test Points Are Controlled Independently	424
Signal Logging Consistently Retains Duplicate Signal Regions	425

Simulink Configuration Parameters	426
Model Help Menu Update	428
Unified Simulation and Embeddable Code Generation Options	428
Mapping of Target Object Properties to Parameters in the Configuration Parameters Dialog Box	444
New Parameters in the Configuration Parameters Dialog Box for Simulation and Embeddable Code Generation ..	452
S-Functions	455
Ada S-Functions	455
Legacy Code Tool Enhancement	455
MATLAB Changes Affecting Simulink	457
Changes to MATLAB Startup Options	457
Handle Graphics Not Supported Under -nojvm Startup Option	457

R2008a

Simulation Performance	460
Rapid Accelerator	460
Additional Zero Crossing Algorithm	460
Component-Based Modeling	461
Efficient Parent Model Rebuilds	461
Scalar Root Inputs Passed Only by Reference	461
Unlimited Referenced Models	461
Embedded MATLAB Function Blocks	463
Nontunable Structure Parameters	463
Bidirectional Traceability	463
Specify Scaling Explicitly for Fixed-Point Data	463
Data Management	465
Array Format Cannot Be Used to Export Multiple Matrix Signals	465
Bus Editor Upgraded	465

Changing Nontunable Values Does Not Affect the Current Simulation	466
Detection of Illegal Rate Transitions	466
Explicit Scaling Required for Fixed-Point Data	466
Fixed-Point Details Display Available	467
More than 2GB of Simulation Data Can be Logged on 64-Bit Platforms	467
Order of Simulink and MPT Parameter and Signal Fields Changed	469
Range Checking for Complex Numbers	471
Rate Transition Blocks Needed on Virtual Buses	471
Sample Times for Virtual Blocks	472
Signals Needing Resolution Are Graphically Indicated ...	472
Simulink File Management	473
Autosave	473
Old Version Notification	473
Model Dependencies Tools	473
Block Enhancements	474
New Discrete FIR Filter Block Replaces Weighted Moving Average Block	474
Rate Transition Block Enhancements	474
Enhanced Lookup Table (n-D) Block	475
New Accumulator Parameter on Sum Block	475
User Interface Enhancements	476
Simulink Library Browser	476
Simulink Preferences Window	476
Model Advisor	476
Solver Controls	477
“What’s This?” Context-Sensitive Help Available for Simulink Configuration Parameters Dialog	479
S-Functions	479

R2007b

Simulation Performance	482
Simulink Accelerator	482

Simulink Profiler	482
Compiler Optimization Level	482
Variable-Step Discrete Solver	483
Referenced Models Can Execute in Normal or Accelerator Mode	483
Accelerator and Model Reference Targets Now Use Standard Internal Functions	484
Component-Based Modeling	485
New Instance View Option for the Model Dependency Viewer	485
Mask Editor Now Requires Java	485
Embedded MATLAB Function Blocks	486
Complex and Fixed-Point Parameters	486
Support for Algorithms That Span Multiple M-Files	486
Loading R2007b Embedded MATLAB Function Blocks in Earlier Versions of Simulink Software	486
Data Management	488
New Diagnostic for Continuous Sample Time on Non-Floating-Point Signals	488
New Standardized User Interface for Specifying Data Types	488
New Block Parameters for Specifying Minimum and Maximum Values	490
New Range Checking of Block Parameters	491
New Diagnostic for Checking Signal Ranges During Simulation	491
Configuration Management	492
Disabled Library Link Management	492
Model Dependencies Tools	492
Embedded Software Design	493
Legacy Code Tool Enhancement	493
Block Enhancements	494
Product Block Reorders Inputs Internally	494
Block Data Tips Now Work on All Platforms	494
Enhanced Data Type Support for Blocks	494

New Simulink Data Class Block Object Properties	495
New Break Link Options for save_system Command	495
Simulink Software Checks Data Type of the Initial Condition Signal of the Integrator Block	495
Usability Enhancements	497
Model Advisor	497
Alignment Commands	497
S-Functions	498
New S-Function APIs to Support Singleton Dimension Handling	498
New Level-2 M-File S-Function Example	498

R2007a+

Bug Fixes

R2007a

Multidimensional Signals	502
New Block Parameters	505
GNU Compiler Upgrade	505
Changes to Concatenate Block	506
Changes to Assignment Block	506
Changes to Selector Block	507
Improved Model Advisor Navigation and Display	508
Change to Simulink.ModelAdvisor.getModelAdvisor Method	509
New Simulink Blocks	510
Change to Level-2 MATLAB S-Function Block	510
Model Dependency Analysis	510
Model File Monitoring	510
Legacy Code Tool Enhancements	510
Continuous State Names	512
Changes to Embedded MATLAB Function Block	512
Referenced Models Support Non-Zero Start Time	517

New Functions Copy a Model to a Subsystem or Subsystem to Model	518
New Functions Empty a Model or Subsystem	518
Default for Signal Resolution Parameter Has Changed ...	519
Referencing Configuration Sets	520
New Block, Model Advisor Check, and Utility Function for Bus to Vector Conversion	521
Enhanced Support for Tunable Parameters in Expressions	522
New Loss of Tunability Diagnostic	522
Port Parameter Evaluation Has Changed	522
Data Type Objects Can Be Passed Via Mask Parameters ..	523
Expanded Options for Displaying Subsystem Port Labels	524
Model Explorer Customization Option Displays Properties of Selected Object	524
Change to PaperPositionMode Parameter	524
New Simulink.Bus.objectToCell Function	525
Simulink.Bus.save Function Enhanced To Allow Suppression of Bus Object Creation	525
Change in Version 6.5 (R2006b) Introduced Incompatibility	525
Nonverbose Output During Code Generation	525
SimulationMode Removed From Configuration Set	525

R2006b

Model Dependency Viewer	528
Enhanced Lookup Table Blocks	528
Legacy Code Tool	528
Simulink Software Now Uses Internal MATLAB Functions for Math Operations	529
Enhanced Integer Support in Math Function Block	529
Configuration Set Updates	530
Command to Initiate Data Logging During Simulation ...	530
Commands for Obtaining Model and Subsystem Checksums	531
Sample Hit Time Adjusting Diagnostic	531
Function-Call Models Can Now Run Without Being Referenced	531
Signal Builder Supports Printing of Signal Groups	531
Method for Comparing Simulink Data Objects	532

Unified Font Preferences Dialog Box	532
Limitation on Number of Referenced Models Eliminated for Single References	532
Parameter Objects Can Now Be Used to Specify Model Configuration Parameters	532
Parameter Pooling Is Now Always Enabled	533
Attempting to Reference a Symbol in an Uninitialized Mask Workspace Generates an Error	533
Changes to Integrator Block's Level Reset Options	534
Embedded MATLAB Function Block Features and Changes	535

R2006a+

No New Features or Changes

R2006a

Signal Object Initialization	544
Icon Shape Property for Logical Operator Block	544
Data Type Property of Parameter Objects Now Settable ..	544
Range-Checking for Parameter and Signal Object Values	544
Expanded Menu Customization	545
Bringing the MATLAB Desktop Forward	545
Converting Atomic Subsystems to Model References	545
Concatenate Block	545
Model Advisor Changes	546
Built-in Block's Initial Appearance Reflects Parameter Settings	546
Double-Click Model Block to Open Referenced Model	547
Signal Logs Reflect Bus Hierarchy	547
Tiled Printing	547
Solver Diagnostic Controls	547
Diagnostic Added for Multitasking Conditionally Executed Subsystems	548
Embedded MATLAB Function Block Features and Changes	548

Model Referencing	558
Function-Call Models	558
Using Noninlined S-Functions in Referenced Models	558
Referenced Models Without Root I/O Can Inherit Sample Times	558
Referenced Models Can Use Variable Step Solvers	558
Model Dependency Graphs Accessible from the Tools Menu	559
Command That Converts Atomic Subsystems to Model References	559
Model Reference Demos	559
Block Enhancements	560
Variable Transport Delay, Variable Time Delay Blocks ...	560
Additional Reset Trigger for Discrete-Time Integrator Block	560
Input Port Latching Enhancements	560
Improved Function-Call Inputs Warning Label	561
Parameter Object Expressions No Longer Supported in Dialog Boxes	561
Modeling Enhancements	563
Annotations	563
Custom Signal Viewers and Generators	563
Model Explorer Search Option	563
Using Signal Objects to Assign Signal Properties	563
Bus Utility Functions	564
Fixed-Point Support in Embedded MATLAB Function Blocks	564
Embedded MATLAB Function Editor	564
Input Trigger and Function-Call Output Support in Embedded MATLAB Function Blocks	564
Find Options Added to the Data Object Wizard	565
Fixed-Point Functions No Longer Supported for Use in Signal Objects	565
Simulation Enhancements	566
Viewing Logged Signal Data	566
Importing Time-Series Data	566

Using a Variable-Step Solver with Rate Transition	
Blocks	566
Additional Diagnostics	566
Data Integrity Diagnostics Pane Renamed, Reorganized ..	567
Improved Sample-Time Independence Error Messages ...	567
User Interface Enhancements	568
Model Viewing	568
Customizing the Simulink User Interface	568
MEX-Files	569
MEX-Files on Windows Systems	569
MEX-File Extension Changed	569

R14SP2

Multiple Signals on Single Set of Axes	572
Logging Signals to the MATLAB Workspace	572
Legends that Identify Signal Traces	572
Displaying Tic Labels	572
Opening Parameters Dialog Box	572
Rootlevel Input Ports	572

R2014a

Version: 8.3

New Features: Yes

Bug Fixes: Yes

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

Compatibility Considerations: Yes

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”

Compatibility Considerations

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 **Compatibility Considerations: Yes**

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

Compatibility Considerations

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 Output block in referenced model honors the **Invalid root Inport/Output 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

Compatibility Considerations: Yes

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”.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

- 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: } 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	Variable name is not preserved. Block parameter name is preserved.	Expression is considered tunable. Variable name is substituted by parameter value.

Block parameter expression	Code generated previously	Code generated in R2014a
value of MP being a nontunable expression.	<pre>struct Parameters_model_ { real_T Gain_Gain; // Expression: } y = model_P.Gain_Gain*u;</pre>	<pre>struct Parameters_model_ { MP*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)	<p>Variable name is not preserved. Block parameter name is preserved.</p> <pre>struct Subsystem { Gain_Gain; // Expression: Arg+1 } y = model_P.Subsystem1.Gain_Gain*u;</pre>	<p>Variable name is preserved as an argument name.</p> <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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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”.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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].

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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:

- LEGO MINDSTORMS EV3 Button
- LEGO MINDSTORMS EV3 Color Sensor
- LEGO MINDSTORMS EV3 Encoder
- LEGO MINDSTORMS EV3 Gyro Sensor
- LEGO MINDSTORMS EV3 Infrared Sensor
- LEGO MINDSTORMS EV3 Display
- LEGO MINDSTORMS EV3 Motor
- LEGO MINDSTORMS EV3 Speaker
- LEGO MINDSTORMS EV3 Touch Sensor
- LEGO MINDSTORMS EV3 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 LEGO MINDSTORMS NXT 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 LEGO MINDSTORMS EV3 hardware.

Use the **Simulink Support Package for Samsung GALAXY Android Devices** block library to access the Samsung GALAXY Android™ hardware:

- Samsung GALAXY Android Accelerometer
- Samsung GALAXY Android Ambient Temperature Sensor
- Samsung GALAXY Audio Capture
- Samsung GALAXY Audio Playback
- Samsung GALAXY Android Camera
- Samsung GALAXY Android Display
- Samsung GALAXY Android FromApp
- Samsung GALAXY Android Gyroscope
- Samsung GALAXY Android Humidity Sensor
- Samsung GALAXY Android Light Sensor
- Samsung GALAXY Android Location Sensor
- Samsung GALAXY Android Pressure Sensor
- Samsung GALAXY Android ToApp
- Samsung GALAXY Android UDP Receive
- Samsung GALAXY Android UDP Send

To install or update this support package, perform the steps described in “Install Support for Samsung GALAXY Android Devices”.

For more information, see “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

<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>tf2c1</code>
<code>firgr</code>	<code>firpr2chfb</code>	<code>iirlpnormc</code>	<code>dsp.DCBlocker</code>

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>

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: Yes

Bug Fixes: Yes

New Simulink Editor

Ability to add rich controls, links, and images to customized block interfaces using the Mask Editor

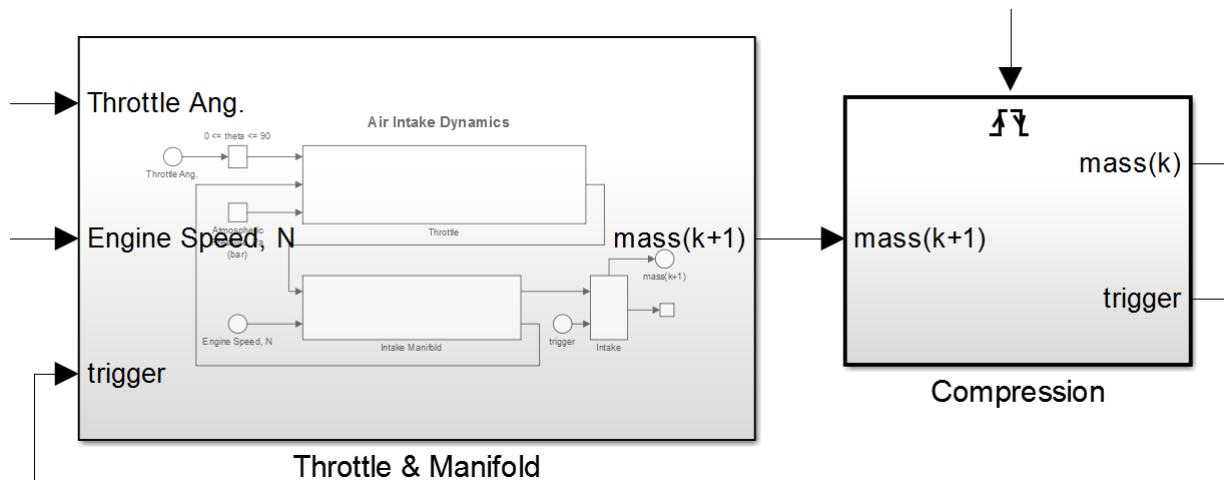
The mask editor is enhanced to support designing mask dialogs using controls such as images, links, and buttons. For details, see Masking.

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

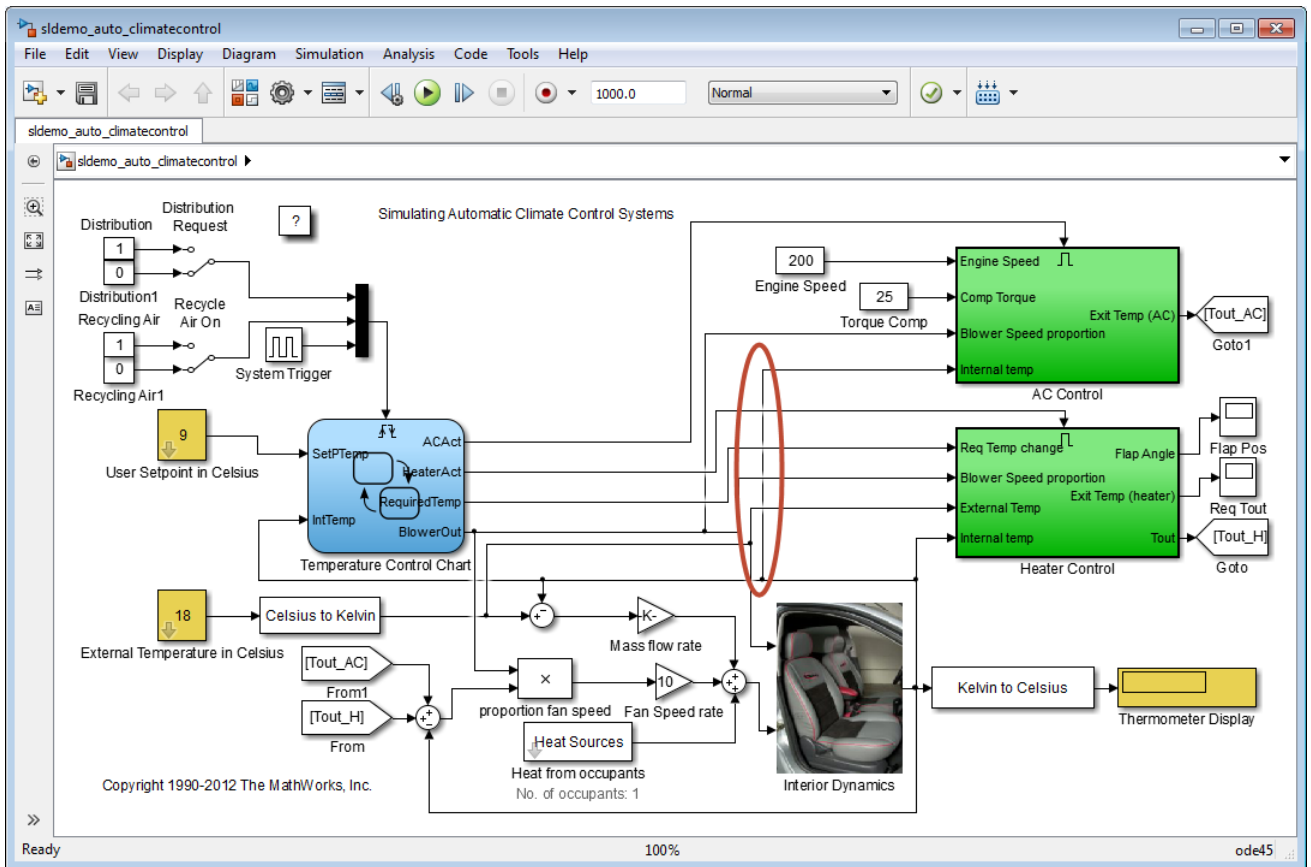
You can comment through blocks such that they appear transparent during simulation. For more information, see [Comment Blocks](#).

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

Compatibility Considerations: Yes

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](#).

Compatibility Considerations

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 http://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

Compatibility Considerations: Yes

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](#).

Compatibility Considerations

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

Compatibility Considerations: Yes

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](#).

Compatibility Considerations

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 `handle-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

Compatibility Considerations: Yes

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 56.
- 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 57.
- 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.

Compatibility Considerations

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	$Y = \text{single}(2^{-127}) * \text{single}(2^{128} - 2^{100})$ $= \text{single}(2^{-127}) * \text{Inf}$	Inf	Strict singles
Previous releases	$Y = \text{single}(\text{double}(2^{-127}) * \text{double}(2^{128} - 2^{100}))$ $= \text{single}(2^{-127} * 3.402823656532e+38)$	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, `getS1RootInportMap`. 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
myObject = Simulink.Parameter(myVar); % Create data object and assign variable value to data object value
```

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

Compatibility Considerations: Yes

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

Compatibility Considerations

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.

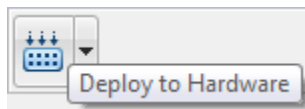
- 1 If you have not done so already, use `supportPackageInstaller` and install a Simulink support package.
- 2 In a model, select **Tools > Run on Target Hardware > Prepare to Run**.

The Configuration Parameters dialog box opens and displays the Run on Target Hardware pane.

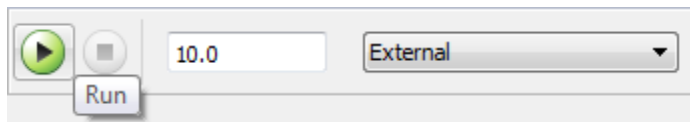
- 3 Set the **Target hardware** parameter to match your target hardware.

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

Compatibility Considerations: Yes

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

Compatibility Considerations

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 error when you save the model	None
Bus primitives and diagnostic is set to error	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.
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 **Compatibility Considerations: Yes**

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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**Compatibility Considerations: Yes**

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: Yes

Bug Fixes: Yes

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 `sldemo_concurrent_execution` 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 and PandaBoard

The block libraries for BeagleBoard and PandaBoard 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 or PandaBoard hardware. For more information, see the BeagleBoard or PandaBoard topics.

After installing the support package, you can open the updated block libraries. In a MATLAB Command Window, enter `beagleboardlib` or `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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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 `slexAutotransRootInportsExample` 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

Compatibility Considerations: Yes

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](#).

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

You may encounter errors or warnings if your code relies on setting enumerated property values of data type objects using inexact matches. In these cases, 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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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 **Compatibility Considerations: Yes**

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

The `emlmex` function has been removed.

Compatibility Considerations

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

Compatibility Considerations: Yes

In a future release, the Model Advisor checks will be reorganized.

Compatibility Considerations

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 Accordian** 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: Yes

Bug Fixes: Yes

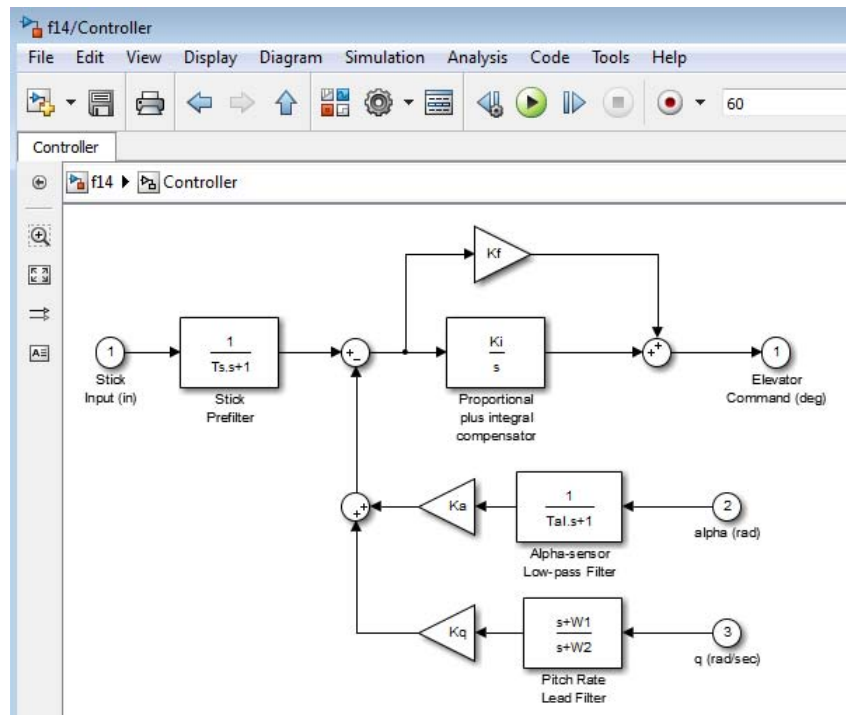
New Simulink Editor

Tabbed windows and automatic window reuse to minimize window clutter Compatibility Considerations: Yes

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.

Compatibility Considerations

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.

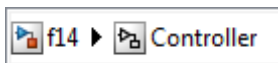
Smart signal routing that determines the simplest signal line path without overlapping blocks

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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 115.

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

Compatibility Considerations: Yes

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**.

Compatibility Considerations

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 129.

Toolbar and status bar control

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

When an interactive operation triggers a callback that causes an error in MATLAB, Simulink:

- Undoes the operation
- Issues an error message

Compatibility Considerations

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 115
- “File Menu” on page 116
- “Edit Menu” on page 116
- “View Menu” on page 117
- “Simulation Menu” on page 118
- “Format Menu” on page 119
- “Tools Menu” on page 120
- “Help Menu” on page 122
- “Simulink Editor Context Menu Changes” on page 123
- “Simulink Editor Mouse and Keyboard Shortcut Changes” on page 126
- “Simulink Editor Badges Changes” on page 129
- “Simulink Preferences Changes” on page 129
- “Simulink and Stateflow Editor Customization Changes” on page 130

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.

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
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 Forward Go To Parent	Access each option using View > Navigate . Go To Parent changed to Up 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 .

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Zoom In Zoom Out Fit System To View Normal (100%)	Access each option using View > Zoom . Fit System to View changed to Fit to View . Normal (100%) changed to Normal View (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 Accelerator Rapid Accelerator Software-in-the-Loop (SIL) Processor-in-the-Loop External	Access each option using Simulation > Mode .
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 Enable TeX Commands	Access each option using Diagram > Format.
Show Name	Diagram > Format > Show Block Name.
Show Drop Shadow	Diagram > Format > Block Shadow.
Show Port Labels	Diagram > Format > Port Labels.
Foreground Color Background Color	Access each option using Diagram > Format.
Screen Color	Diagram > Format > Canvas Color.
Show Smart Guides	View > Smart Guides.
Align Blocks Distribute Blocks Resize Blocks	Access using block alignment, distribution, and resizing options using Diagram > Arrange.
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.

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
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
<p>Compare Simulink XML Files</p> <p>Control Design</p> <p>Coverage Settings</p> <p>Design Verifier</p> <p>Fixed-Point Tool</p> <p>Model Advisor</p> <p>Model Dependencies</p> <p>Parameter Estimation</p> <p>Profiler</p> <p>Requirements</p>	<p>Access these tools using Analysis.</p> <p>The menu text for these items has changed:</p> <ul style="list-style-type: none"> • Coverage Settings changed to Coverage. • Fixed-Point changed to Fixed Point Tool (i.e., there is no hyphen now) • Model Dependencies > View/Edit Manifest Contents changed to Model Dependencies > Edit Manifest Contents. • Profiler changed to Show Profiler Report.
Simulink Debugger	Simulation > Debug > Debug Model.
<p>Bus Editor</p> <p>Lookup Table Editor</p>	Access these tools using Edit .
Inspect Logged Signals	Simulation > Output > Simulation Data Inspector.
Signal & Scope Manager	Diagram > Signals & Ports > Signal & Scope Manager.
<p>Code Generation</p> <p>External Mode Control Panel</p> <p>HDL Code Generation</p> <p>Simulink Code Inspector</p> <p>Verification Wizards</p>	<p>Access these options using Code.</p> <p>Code Generation changed to C/C++ Code.</p> <p>HDL Code Generation changed to HDL Code.</p>

R2012a Simulink Editor Menu Bar Item	New Simulink Editor Equivalent
Define Data Classes Data Object Wizard	Access these options using Code > Data Objects . 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 Blocksets	Help > Simulink > Blocks & Blocksets Reference .
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.
Foreground Color Background Color	Access these options using Format .
Convert to Model Block (for Subsystem block context menu)	Subsystem & Model Reference > Convert Subsystem to > Referenced Model.
Format > Flip Block Format > Flip Name Format > Rotate Block	Access these options using Rotate & Flip .
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.

R2012a Simulink Editor Context Menu	New Simulink Editor Equivalent
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 .
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 .

Task	R2012a Simulink Editor Shortcut	New Simulink Editor Equivalent
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 .
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 <i>Navigate Model Hierarchy</i> .
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 132
- “Support for PandaBoard hardware” on page 133

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 `arduino1lib` 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

Compatibility Considerations: Yes

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](#).

Compatibility Considerations

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:

Old Name	New Name
X0	InitialCondition
StateIdentifier	StateName
RTWStateStorageClass	CodeGenStateStorageClass
RTWStateStorageTypeQualifier	CodeGenStateStorageTypeQualifier

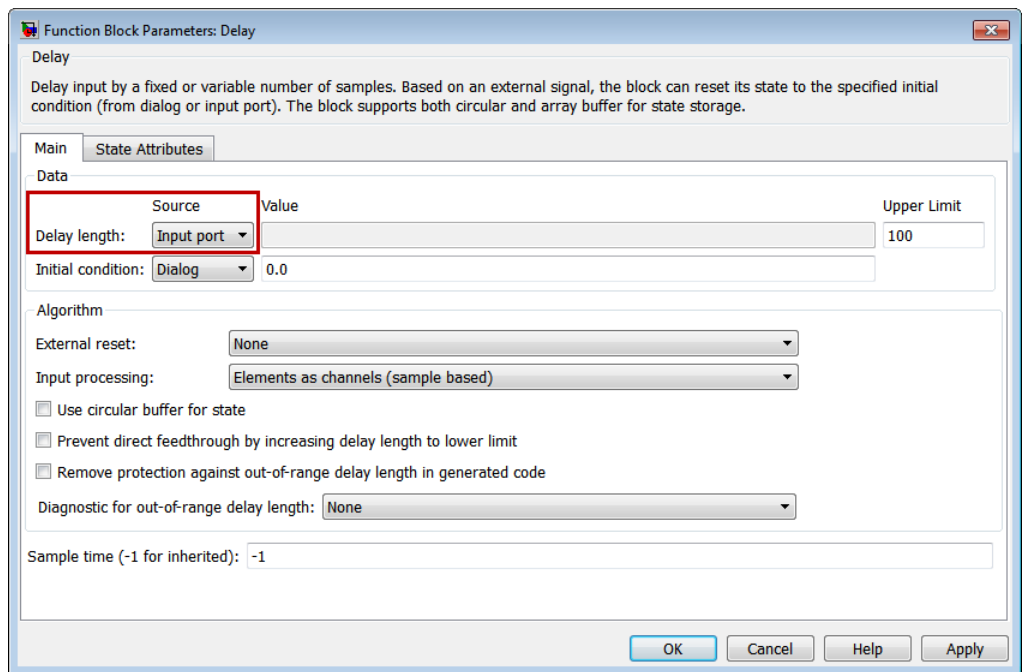
New variants of Delay block in Discrete library

The Discrete library now contains the following additional variants of the Delay block:

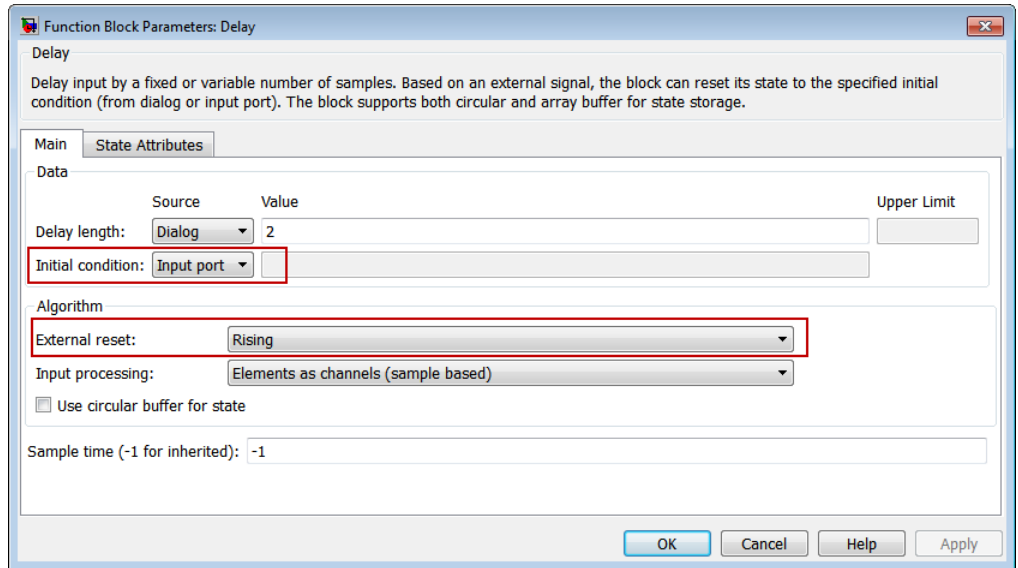
- Variable Integer Delay block
- Resettable Delay block

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.

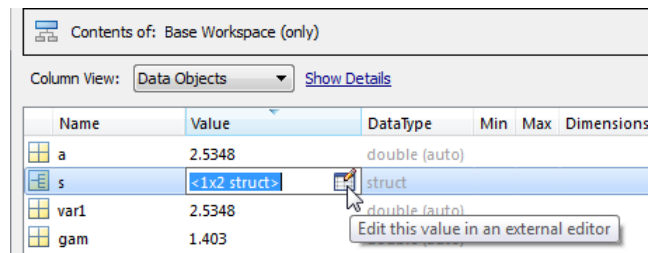
Data Management

Variable Editor access from within Model Explorer

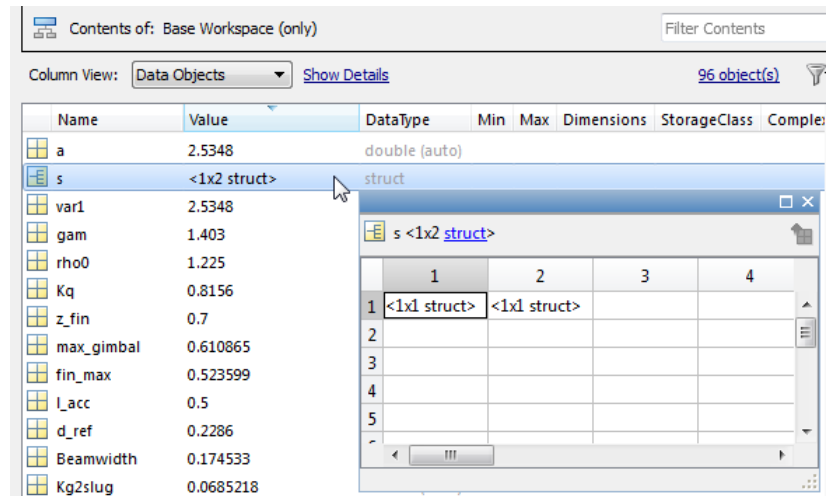
In the Model Explorer **Contents** pane, you can use the Variable Editor to edit variables from the MATLAB workspace or model workspace. The Variable Editor is available for editing large arrays and structures.

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

Compatibility Considerations: Yes

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](#).

Compatibility Considerations

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 Inputs](#).

Dataset signal logging format for increased flexibility and ease of use

Compatibility Considerations: Yes

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](#).

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

`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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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 `EnableAll` argument has been replaced by two settings: `EnableAllAsWarning` and `EnableAllAsError`. The `EnableAllAsError` setting is now the default. In R2012a and R2011b, `EnableAll` 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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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.

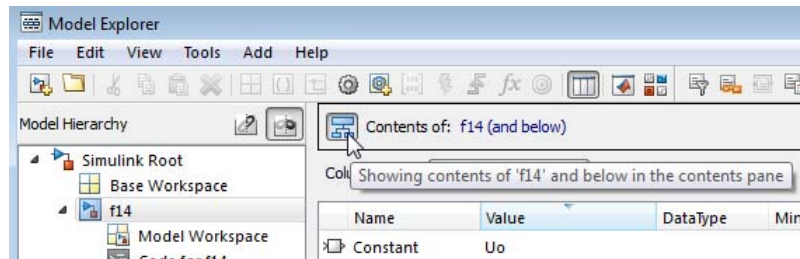
Show partial or whole model hierarchy contents

Compatibility Considerations: Yes

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](#).

Compatibility Considerations

The Model Explorer search bar no longer provides the **Show 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

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: Yes

Bug Fixes: Yes

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 `slupdate` 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

Compatibility Considerations: Yes

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"
```

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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 **Compatibility Considerations: Yes**

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

In R2012a, the property `RTWInfo` of a Simulink data object has been renamed as `CoderInfo`.

Compatibility Considerations

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` **Compatibility Considerations: Yes**

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.

Compatibility Considerations

- 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 Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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`.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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	Binary format of SLX files can cause third-party	Register <code>.slx</code> as a binary file format with

Release	Operations with Possible Compatibility Considerations	What Happens	Action
	a text format by default.	tools to corrupt the files when you submit them.	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

Compatibility Considerations: Yes

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 in the Simulink documentation.

Compatibility Considerations

Functionality	What Happens When You Use This Functionality?	Use This Functionality Instead	Compatibility Considerations
getRootDirectory	Warns	getRootFolder	Replace all instances of <code>getRootDirectory</code> with <code>getRootFolder</code>

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 Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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 sample-based.
- The input signal is a vector, matrix, or N-dimensional array.

Compatibility Considerations

To eliminate this warning, you must upgrade your existing models using the `slupdate` 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 `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 `slupdate` function. Therefore, you should upgrade your existing modes using `slupdate` 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 and Using `slupdate`

Compatibility Considerations: Yes

In this release, the `Model` block has been updated so that its operation does not depend on the frame status of its input signals.

Compatibility Considerations

In a future release, signals will not have a framedness 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 slupdate.
- 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 framedness 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

Compatibility Considerations: Yes

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

Compatibility Considerations

To eliminate this warning, you must upgrade your existing models using the `sIupdate` 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 `slupdate` function. Therefore, upgrade your existing models using `slupdate` 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.

Column View: Block Data Types [Show Details](#)


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.

Contents of: f14 (only) Filter Contents **Model Properties: f14**

Column View: Block Data Types [Show Details](#) 39 of 41 object(s) 

Name	BlockType	OutDataTypeStr	Out
Model Worksp...			
Code for f14			
Advice for f14			
Simulink Desig...			
Configuration ...			
u	Inport	Inherit: auto	
Actuator Model	TransferFcn		
Aircraft Dynam...	SubSystem		
Angle of Attack	Scope		
Controller	SubSystem		
Dryden Wind ...	SubSystem		
Gain	Gain	Inherit: Same as input	

- All Simulink Objects
- Blocks
- Named Signals/Connections
- All Signals/Connections
- Annotations
- All Stateflow Objects
- States/Functions/Boxes/Etc.
- Transitions
- Junctions
- Events
- Data

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: Yes

Bug Fixes: Yes

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

Compatibility Considerations: Yes

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

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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 Output

In R2011b, you can view the compiled design minimum and maximum values at a block output 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 output 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 output 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_md1ref_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

Compatibility Considerations: Yes

Simulink no longer supports calls to `s1DataTypeAndScale` when:

- The first argument is a `Simulink.AliasType` object
- The first argument is a `Simulink.NumericType` object with property `IsAlias` set to true

Compatibility Considerations

If your model calls the internal function `s1DataTypeAndScale`, 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 `s1DataTypeAndScale`.

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**Compatibility Considerations: Yes**

Simulink generates an error if you pass an input argument to the classes `Simulink.Signal` and `Simulink.Parameter`.

Compatibility Considerations

`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
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

Compatibility Considerations: Yes

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.

Compatibility Considerations

In R2011b, the Time Series tool (`tstool`) 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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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](#).

Compatibility Considerations

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.

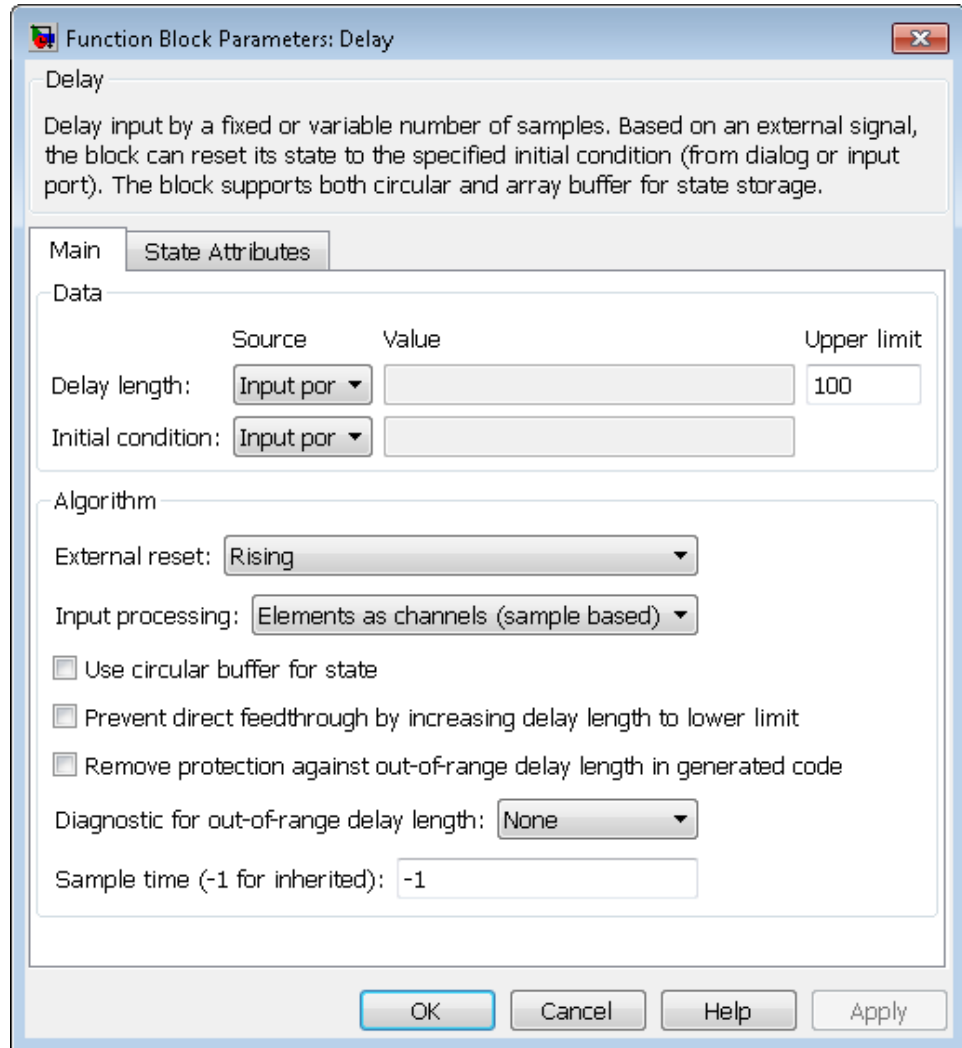
Block Enhancements

New Delay Block That Upgrades the Integer Delay Block

Compatibility Considerations: Yes

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.

Compatibility Considerations

The following incompatibilities might affect simulation of pre-R2011b models that use the Integer Delay block:

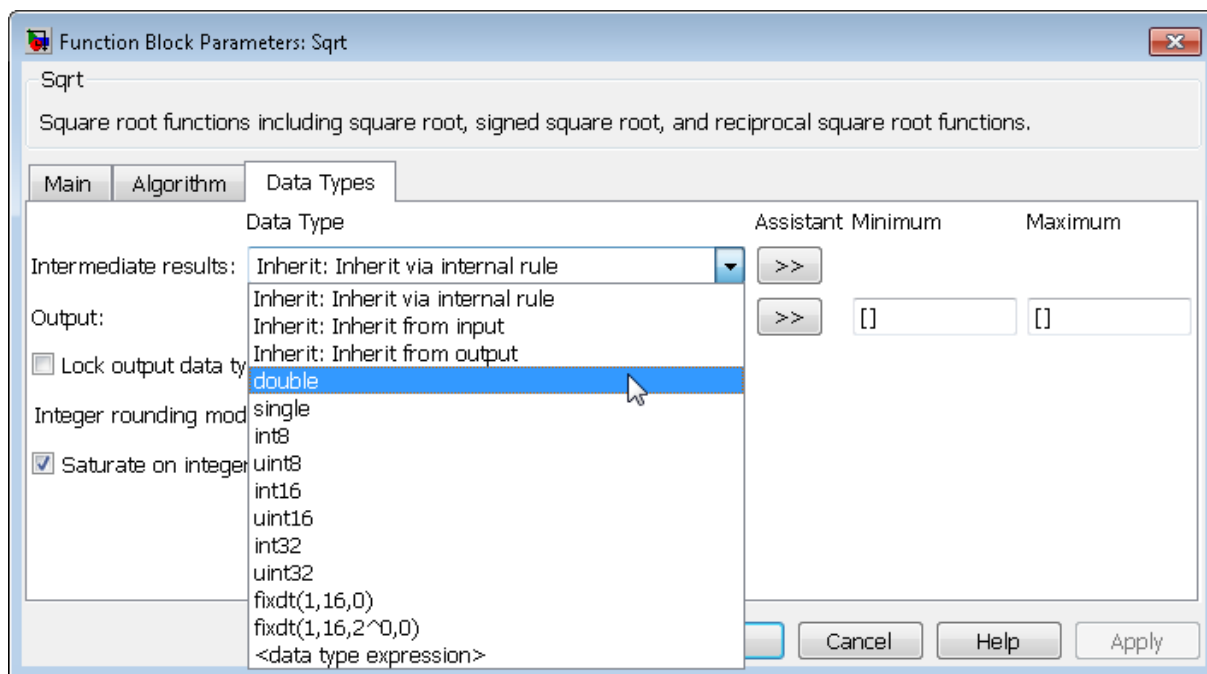
Source of Incompatibility	Behavior in the Integer Delay Block	Behavior in the Delay Block	Rationale	How to Avoid an Error
Initial condition for input signals of N-by-1 or 1-by-N dimensions and sample-based processing	Suppose that delay length is D. For Initial condition , the Integer Delay block supports 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.

Source of Incompatibility	Behavior in the Integer Delay Block	Behavior in the Delay Block	Rationale	How to Avoid an Error
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.

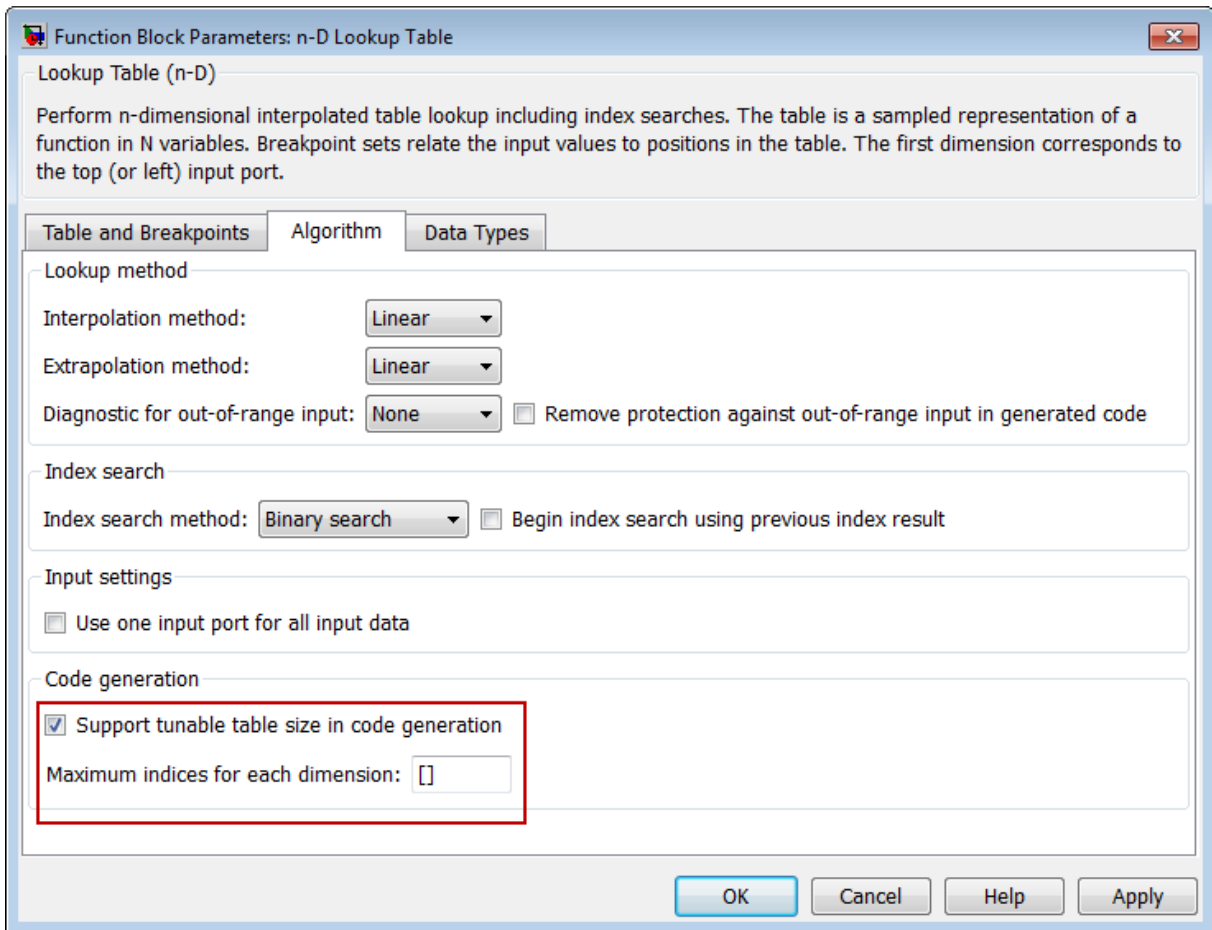
For a summary of data type configurations that are valid (input, output, and intermediate results), refer to the block reference page.

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

To...	Select...
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 Compatibility Considerations: Yes

For R2011b, error and warning message identifiers have changed in Simulink.

Compatibility Considerations

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: Yes

Bug Fixes: Yes

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

Compatibility Considerations: Yes

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**.

Compatibility Considerations

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](#).

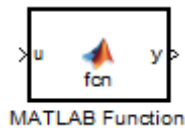
In earlier releases, you had to use the `Simulink.ModelReference.protect` command to create a protected model.

MATLAB Function Blocks

Embedded MATLAB Function Block Renamed as MATLAB Function Block

Compatibility Considerations: Yes

In R2011a, Embedded MATLAB Function blocks were renamed as MATLAB Function blocks in Simulink models. The block also has a new look:



Compatibility Considerations

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

Compatibility Considerations: Yes

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` objectPrior 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](#).

Compatibility Considerations

- 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 **Compatibility Considerations: Yes**

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`.

Compatibility Considerations

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

Compatibility Considerations: Yes

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

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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`.

Compatibility Considerations

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**Compatibility Considerations: Yes**

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*

Compatibility Considerations

- 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

Compatibility Considerations: Yes

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](#).

Compatibility Considerations

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

Compatibility Considerations: Yes

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"> • Block renamed as 2-D Lookup Table

Block	Enhancements to the Previous Version	Other Changes
	<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"> • 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.

Compatibility Considerations

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	Change the extrapolation method or the port data types of the block.	You also see a message that explains possible numerical differences.

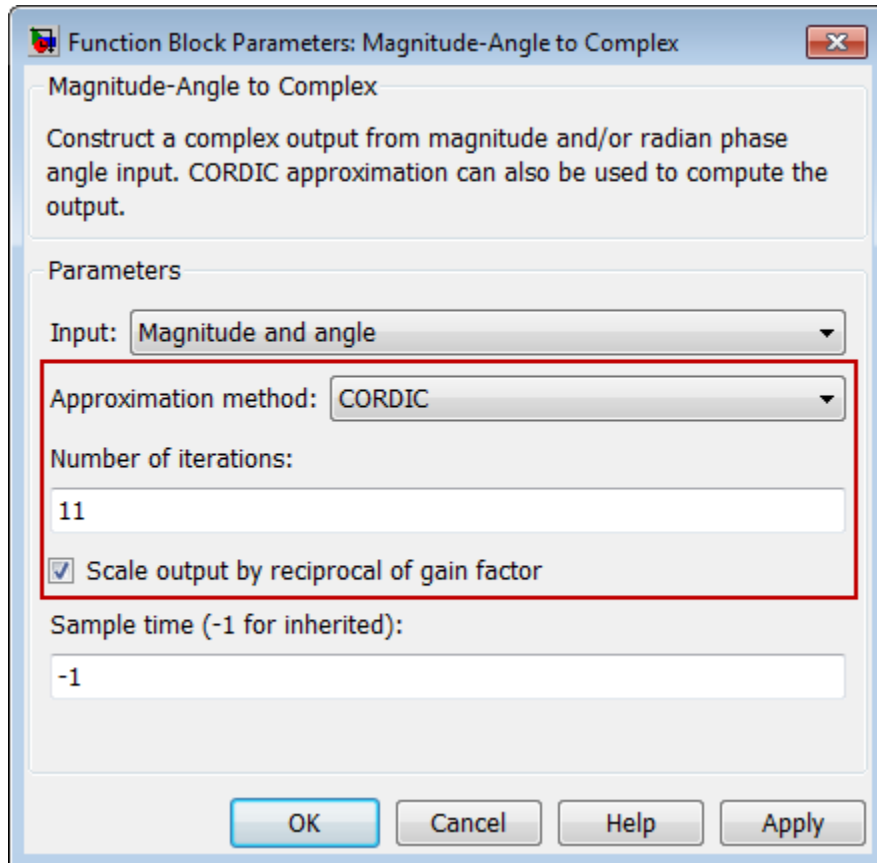
Incompatibility Warning	Recommended Action	What Happens for Automatic Block Replacement
inputs and outputs are the same floating-point type.		
The block uses small fixed-point word lengths, so that interpolation uses only one rounding operation. The new block uses two rounding operations for interpolation.	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

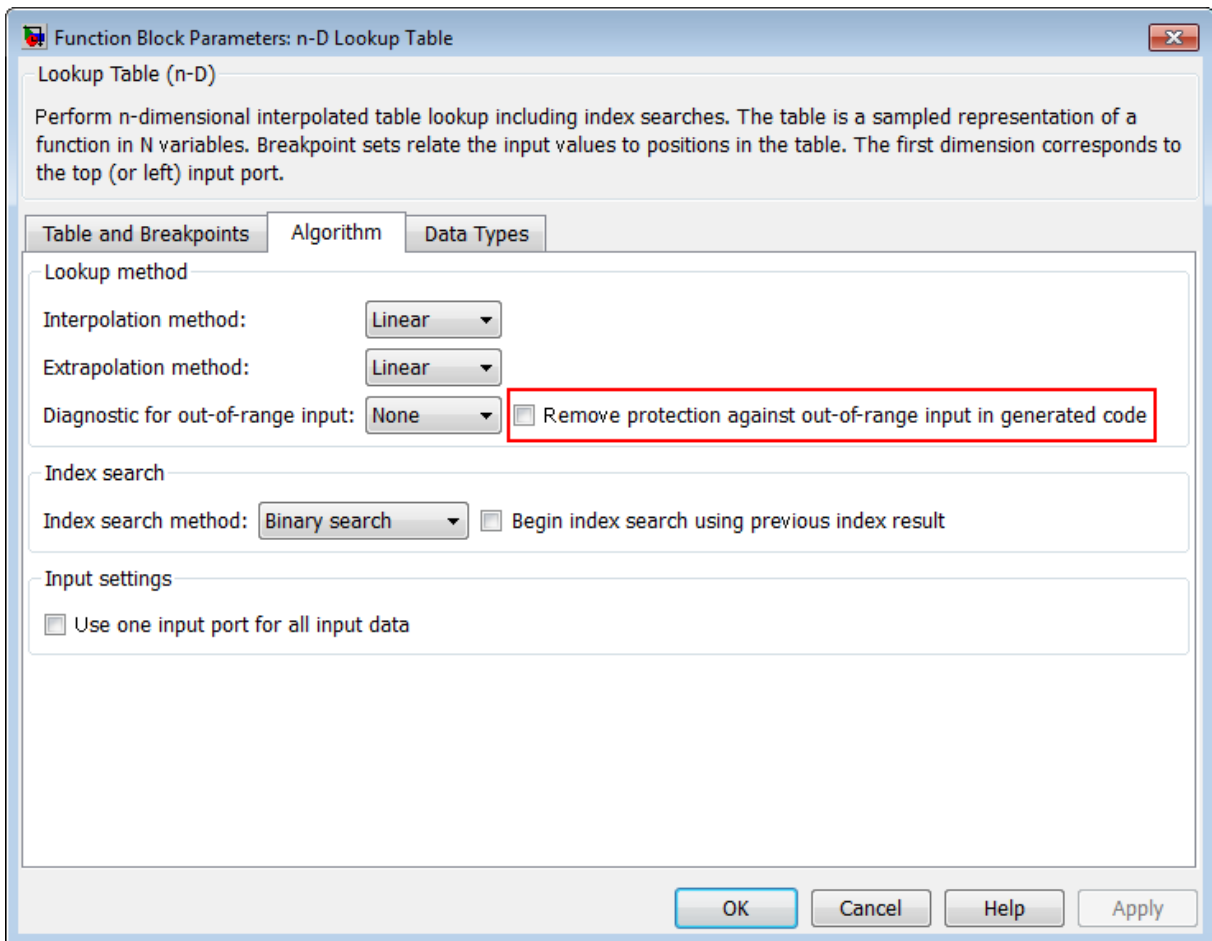
The following parameter changes apply to the Shift Arithmetic block. For backward compatibility, the old command-line parameters continue to work.

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

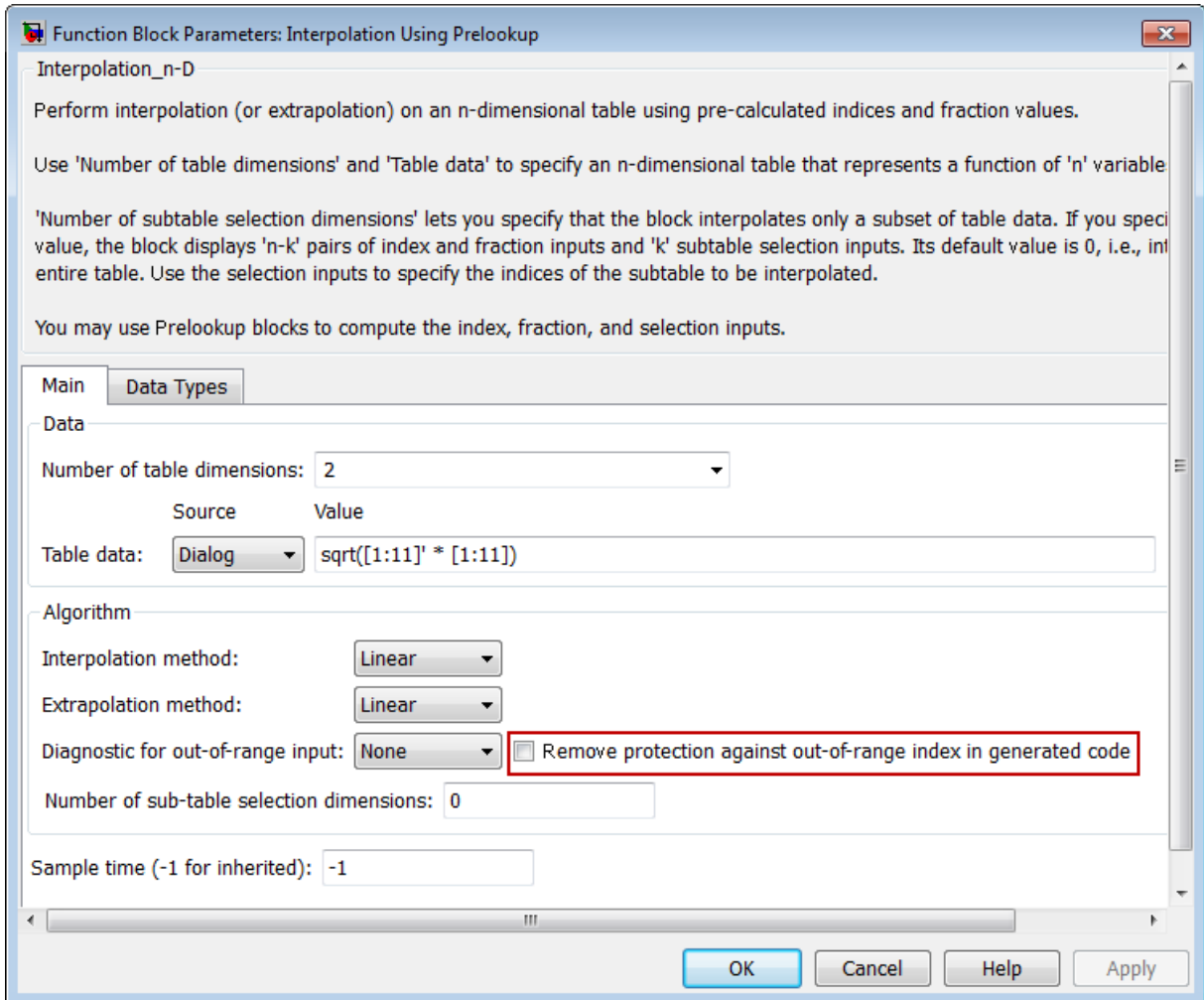
The read-only **BlockType** property has also changed from SubSystem to ArithShift.

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:

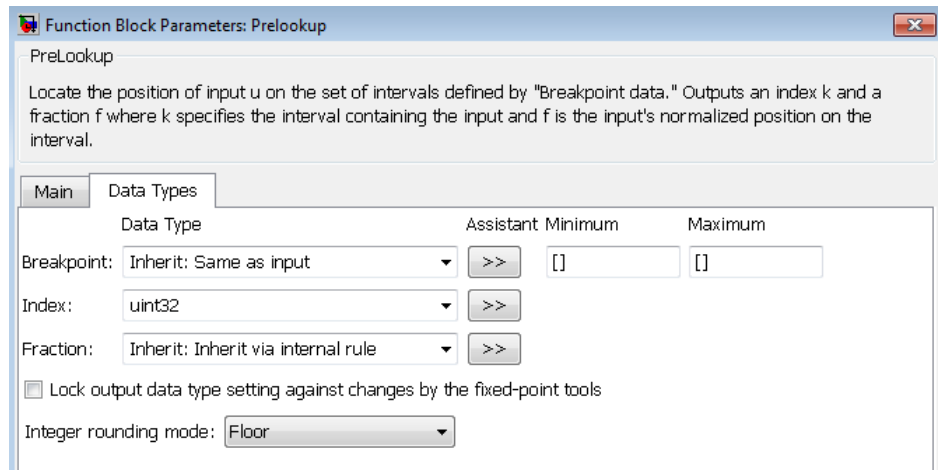
Parameter Setting from Previous Releases	Parameter Setting for R2011a
Check index in generated code is selected.	Remove protection against out-of-range index in generated code is not selected.
Check index in generated code is not selected.	Remove protection against out-of-range index in generated code is selected.

For backward compatibility, the command-line parameter `CheckIndexInCode` continues to work.

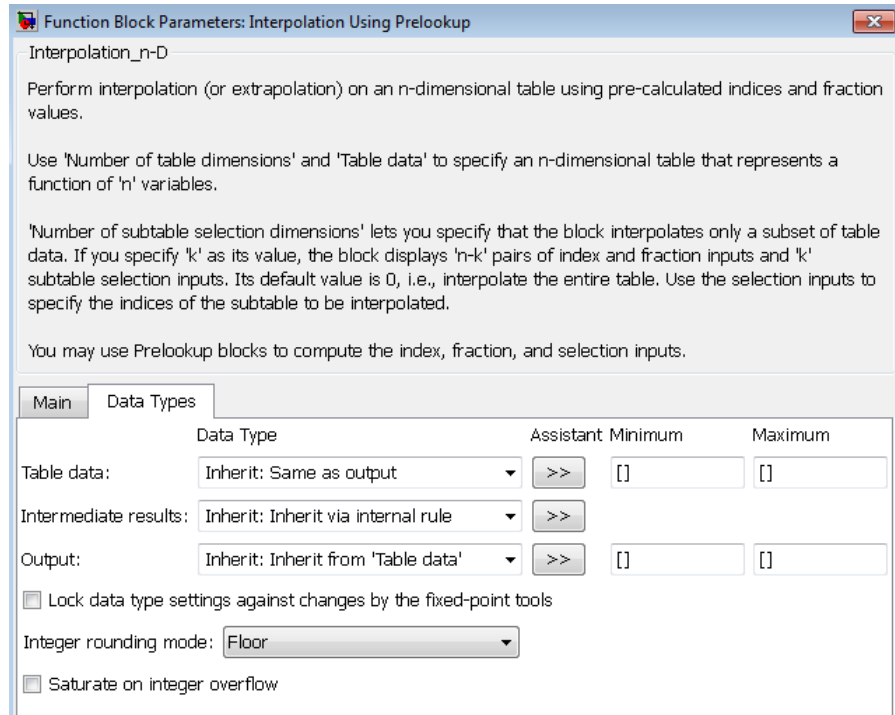
Enhanced Dialog Layout for the Prelookup and Interpolation Using Prelookup Blocks

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 $(((((1/u1)/u2)/u3)/u4) /uN)$.

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 Compatibility Considerations: Yes

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.

Compatibility Considerations

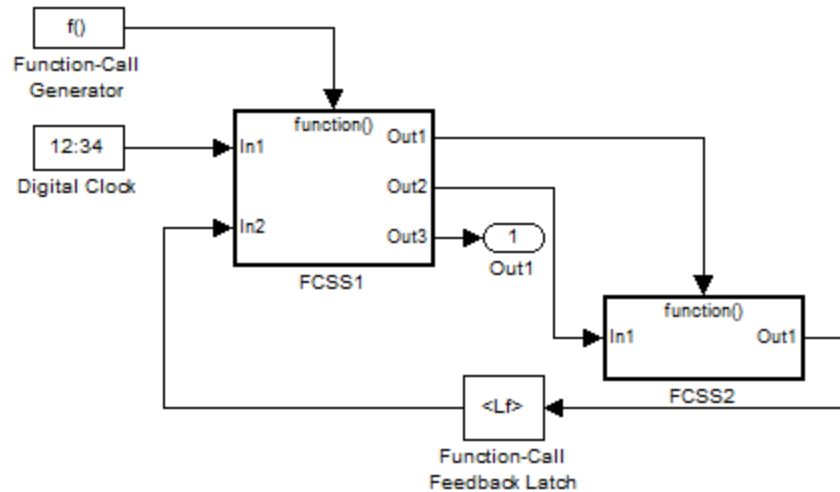
Previously, if **Inline parameters** was off, the sample time of the Ground block depended on sample-time propagation. Now, the following conditions hold true:

- 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.

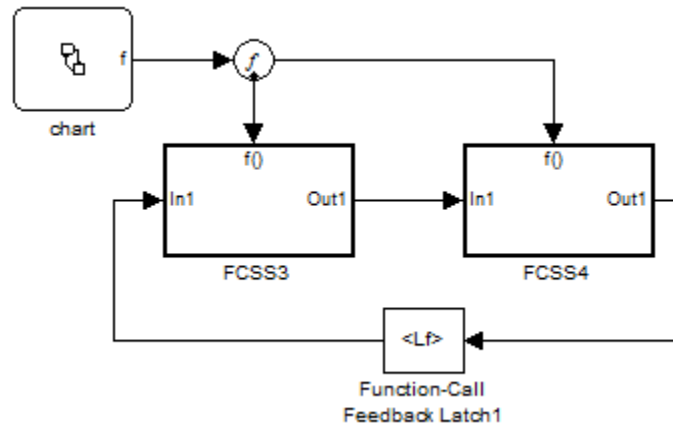
New Function-Call Feedback Latch Block

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 outputs of Model blocks. To assign a GetSet custom storage class to the inport or output of a referenced model block, use one of the following methods.

- 1** Assign the GetSet custom storage class to the root-level inport or output 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 outputs of the referenced model block in the parent model. In this case, the referenced model must use function prototype control to specify that the output 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.

Inspect Signals Compare Signals Compare Runs

Block Path	Signal Name	Line	Plot
Run 1: sldemo_f14			
sldemo_f1...	q, rad/sec	—	<input checked="" type="checkbox"/>
sldemo_f1...	alpha, rad	—	<input checked="" type="checkbox"/>
sldemo_f1...	Stick	—	<input checked="" type="checkbox"/>
sldemo_f1...	NzPilot, g	—	<input checked="" type="checkbox"/>
Run 2: sldemo_f14			
sldemo_f1...	q, rad/sec	—	<input checked="" type="checkbox"/>
sldemo_f1...	alpha, rad	—	<input checked="" type="checkbox"/>
sldemo_f1...	Stick	—	<input checked="" type="checkbox"/>
sldemo_f1...	NzPilot, g	—	<input checked="" type="checkbox"/>

View: sldemo_f14_plot4 [Hide Details](#)

View Name	Layout
Run 1: sldemo_f14	
q, rad/sec sldemo_f14/Aircr... sldem...	[1 1] <input checked="" type="checkbox"/>
alpha, rad sldemo_f14/Aircr... sldem...	[1 2] <input checked="" type="checkbox"/>
Stick sldemo_f14/Pilot sldem...	[2 1] <input checked="" type="checkbox"/>
NzPilot, g sldemo_f14/Pilot ... sldem...	[2 2] <input checked="" type="checkbox"/>
Run 2: sldemo_f14	
NzPilot, g sldemo_f14/Pilot ... sldem...	[2 2] <input checked="" type="checkbox"/>
Stick sldemo_f14/Pilot sldem...	[2 1] <input checked="" type="checkbox"/>
alpha, rad sldemo_f14/Aircr... sldem...	[1 2] <input checked="" type="checkbox"/>
q, rad/sec sldemo_f14/Aircr... sldem...	[1 1] <input checked="" type="checkbox"/>

New view from current Delete view Export... Import... Replace runs...

The figure displays four plots arranged in a 2x2 grid. The top-left plot shows the 'q, rad/sec' signal (black line) over time (0 to 60). The top-right plot shows the 'alpha, rad' signal (green line) over time (0 to 60). The bottom-left plot shows the 'Stick' signal (red line) over time (0 to 60). The bottom-right plot shows the 'NzPilot, g' signal (cyan line) over time (0 to 60). Each plot has a corresponding icon for linking and zooming.

You can create multiple views by clicking the **New view from current** button. In each view, you can:

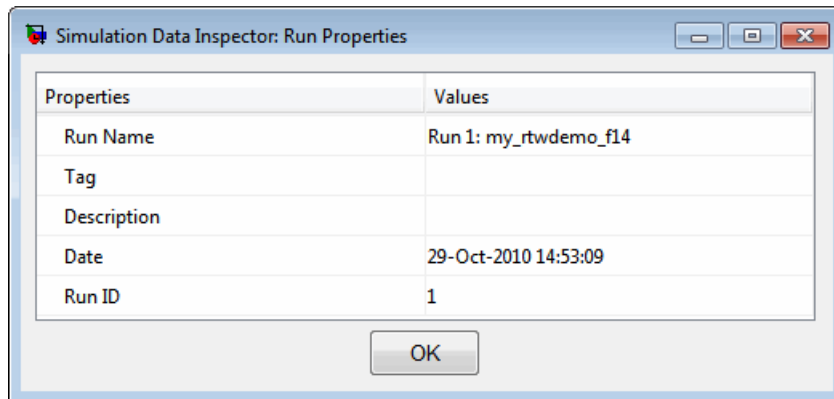
- Modify the number of plots by clicking the **Layout** column to display the plot matrix.

- 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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

New Features: No

Bug Fixes: Yes

R2010bSP1

Version: 7.6.1

New Features: No

Bug Fixes: Yes

R2010b

Version: 7.6

New Features: Yes

Bug Fixes: Yes

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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_md1ref_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](#).

Compatibility Considerations

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

Compatibility Considerations: Yes

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

Compatibility Considerations

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
Outport	Specify properties via bus object Bus object for validating input bus	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 a 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` in the Simulink documentation.

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 in the Simulink Graphical User Interface documentation.

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

Compatibility Considerations: Yes

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
 - Multiport 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](#).

Compatibility Considerations

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.

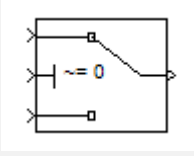
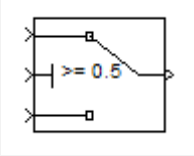
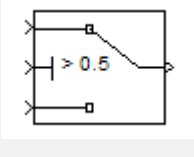
This enhancement enables you to change table data without stopping a simulation. For example, you can incorporate new table data if the physical system you are simulating changes.

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
	<div data-bbox="422 338 1328 546"> <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> </div>
	<div data-bbox="422 598 1328 807"> <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> </div>
	<div data-bbox="422 868 1328 1076"> <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> </div>

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

If you have any R2010a or earlier models that use the Bias block to offset a frame-based signal, you can use the `slupdate` function to upgrade your model. For each instance where you use a Bias block with a frame-based input signal, the `slupdate` 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

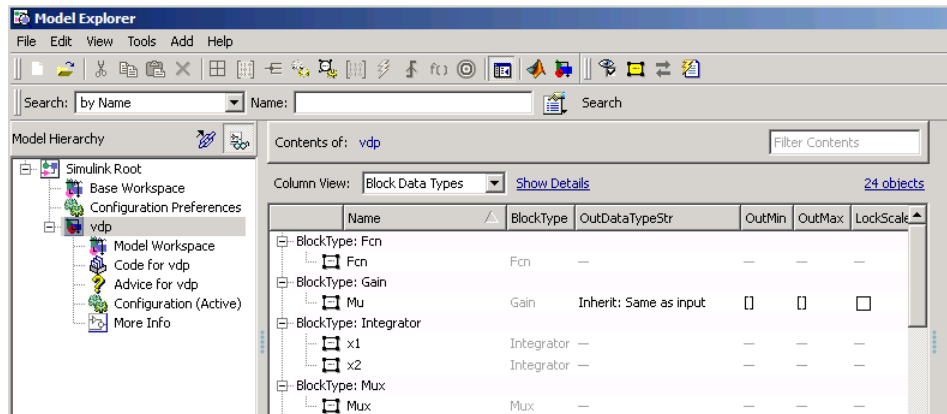
Previously, you could save and load a configuration set from the command line only, requiring many steps. Now you can save and load a configuration set using the Model Explorer. You can also save or load the active configuration set using one function, the `Simulink.BlockDiagram.saveActiveConfigSet` 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

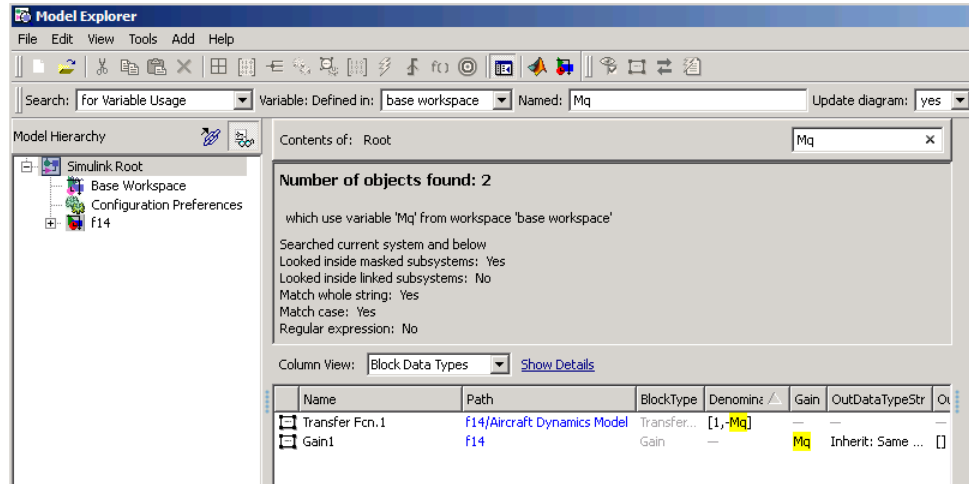
In the Contents pane, you can group data based on a property values. For example, you can group by the `BlockType` property by right-clicking that column heading and selecting the **Group by This Column** menu item. The result looks similar to this:



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

Compatibility Considerations: Yes

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'.

Compatibility Considerations

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 <http://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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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: Yes

Bug Fixes: Yes

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

Compatibility Considerations: Yes

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](#).

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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.

Model Reference Support for Custom Code

Select the new `SupportModelReferenceSimTargetCustomCode` model parameter to have SIM target Accelerator code generation include Stateflow and Embedded MATLAB custom code for a referenced model. The default setting for this parameter is off.

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 **Compatibility Considerations: Yes**

Custom floating-point types, `float(TotalBits, ExpBits)`, are no longer supported.

Compatibility Considerations

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 `numericType` 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_md1ref_dsm` demo.

Models with No States Now Return Empty Variables

Compatibility Considerations: Yes

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).

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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 `s1replace_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
ssGetBusElementComplexSignal	Get the signal complexity for a bus element.
ssGetBusElementDataType	Get the data type identifier for a bus element.
ssGetBusElementDimensions	Get the dimensions of a bus element.
ssGetBusElementName	Get the name of a bus element.
ssGetBusElementNumDimensions	Get the number of dimensions for a bus element.
ssGetBusElementOffset	Get the offset from the start of the bus data type to a bus element.
ssGetNumBusElements	Get the number of elements in a bus signal.
ssGetSFcnParamName	Get the value of a block parameter for an S-function block.
ssIsDataTypeABus	Determine whether a data type identifier represents a bus signal.
ssRegisterTypeFromParameter	Register a data type that a parameter in the Simulink data type table specifies.
ssSetBusInputAsStruct	Specify whether to convert the input bus signal for an S-function from virtual to nonvirtual.
ssSetBusOutputAsStruct	Specify whether the output bus signal from an S-function must be virtual or nonvirtual.
ssSetBusOutputObjectName	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

Compatibility Considerations: Yes

You can use the new Sqrt block to perform square-root calculations. This block includes the following functions:

Function	Icon
sqrt	
signedSqrt	
rSqrt	

Compatibility Considerations

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
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

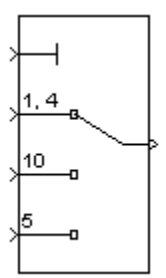
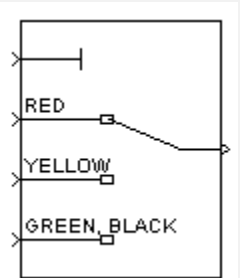
The Direct Lookup Table (n-D) block now supports:

- 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

Compatibility Considerations: Yes

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 <input type="text"/> Data port indices (e.g. {1,[2,3]}): <input type="text" value="{[1,4],10,5}"/>	 <p>The block icon shows a rectangular switch block with a control input on the left and a single output on the right. Three data ports are shown on the left side, each with a small square terminal. The top port is labeled '1, 4', the middle port is labeled '10', and the bottom port is labeled '5'. A switch symbol is positioned between the top and middle ports, indicating that the switch can be controlled by either of these two ports.</p>
Data port order: Specify indices <input type="text"/> Data port indices (e.g. {1,[2,3]}): <input type="text" value="{myColors.RED, myColors.YELLOW, [myColors.GREEN, myColors.BLACK]}"/>	 <p>The block icon shows a rectangular switch block with a control input on the left and a single output on the right. Three data ports are shown on the left side, each with a small square terminal. The top port is labeled 'RED', the middle port is labeled 'YELLOW', and the bottom port is labeled 'GREEN_BLACK'. A switch symbol is positioned between the top and middle ports, indicating that the switch can be controlled by either of these two ports.</p>

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: <input type="text" value="3"/> <input type="checkbox"/> Use zero-based indexing	Data port order: <input type="text" value="One-based contiguous"/> Number of data ports: <input type="text" value="3"/>
Number of inputs: <input type="text" value="3"/> <input checked="" type="checkbox"/> Use zero-based indexing	Data port order: <input type="text" value="Zero-based contiguous"/> Number of data ports: <input type="text" value="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.

Compatibility Considerations

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 CORDIC 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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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>slookup</code> function on your model.	Perform block replacement with the Lookup Table (n-D) block.

Compatibility Considerations

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

Compatibility Considerations: Yes

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

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

In R2010a, if you use `slupdate` 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

Compatibility Considerations: Yes

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 `slupdate` 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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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*.

Compatibility Considerations

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

Compatibility Considerations: Yes

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*.

Compatibility Considerations

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 `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.

In this release, the addition of Technical Note 1821 facilitates your incorporating Ada code into Simulink without using Ada S-function support. This note, “Developing and Building Ada S-Functions for Simulink”, is available at Technical Note 1821 and demonstrates:

- How to build a C MEX S-function from Ada code without using the `mex ada` command
- An example of an Ada wrapper around a C MEX S-Function API

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

Compatibility Considerations: Yes

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 [Legacy Code Tool](#).

- [Integrating Existing C Functions into Simulink Models with the Legacy Code Tool](#) in the [Writing S-Functions](#) documentation
- [legacy_code](#) function reference page

Compatibility Considerations

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 DO-178B, 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

New Features: No

Bug Fixes: Yes

R2009b

Version: 7.4

New Features: Yes

Bug Fixes: Yes

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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 in the Simulink User's Guide.

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

Compatibility Considerations: Yes

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

Compatibility Considerations

The new compilation report is not supported by the MATLAB internal browser on Sun™ Solaris™ 64-bit platforms. To view the compilation report on Sun Solaris 64-bit platforms, you must 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 **Compatibility Considerations: Yes**

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

In order to restart an R2009a simulation in R2009b, you should first regenerate the initial `SimState` in R2009b.

Compatibility Considerations

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 **Compatibility Considerations: Yes**

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.

Compatibility Considerations

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: $1/\text{sqrt}$. 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 magnitude², 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
- Multiport 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

Compatibility Considerations: Yes

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	maskTabDims	NumberOfTableDimensions
Inputs select this object from table	outDims	InputsSelectThisObjectFromTable
Make table an input	tabIsInput	TableIsInput
Table data	mXTable	Table

Prompt on Block Dialog Box	Old Command-Line Parameter	New Command-Line Parameter
Action for out-of-range input	clipFlag	ActionForOutOfRangeInput
Sample time	samptime	SampleTime

The read-only **BlockType** parameter has also changed from S-Function to LookupNDDirect.

Compatibility Considerations

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

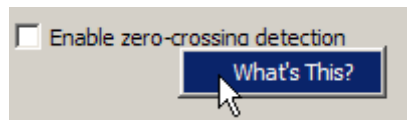
Context-Sensitive Help for Simulink Blocks in the Continuous Library

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: Yes

Bug Fixes: Yes

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](#) in the Simulink documentation.

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](#) in the Simulink documentation.

Data Management

Signal Can Resolve to at Most One Signal Object

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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 magnitude², 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

Compatibility Considerations: Yes

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 magnitude², 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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

In R2009a, the Lookup Table Designer is no longer available in the Lookup Table Editor.

Compatibility Considerations

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: Yes

Bug Fixes: Yes

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 Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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`.

Compatibility Considerations

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 428.

Data Type Override Now Works Consistently on Outputs

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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 410.

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 410.

Modifying a Link to a Library Block in a Callback Function Can Cause Illegal Modification Errors

Compatibility Considerations: Yes

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.

Compatibility Considerations

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:

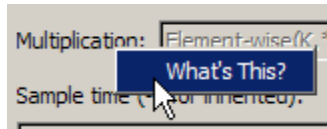
- Bus Creator
- Bus Selector
- Constant
- Data Type Conversion
- Demux
- Discrete-Time Integrator
- Gain
- Inport
- Integrator
- Logical Operator
- Mux
- Outport
- Product
- Relational Operator
- Saturation
- Subsystem
- Sum
- Switch
- Terminator
- Unit Delay

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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
Solver	Consecutive min step size violations / MaxConsecutiveMinStep	Number of consecutive min steps / MaxConsecutiveMinStep

Location	Previous Parameter	New Parameter
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 obsoleted.

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:

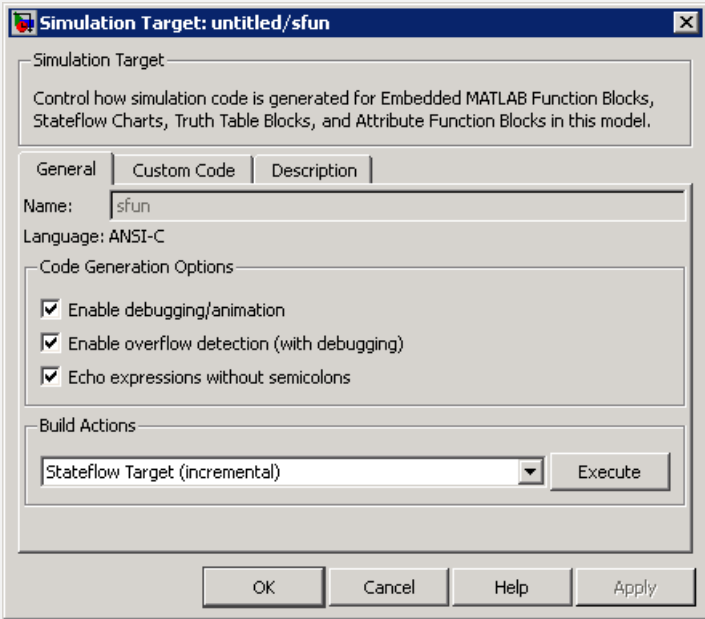
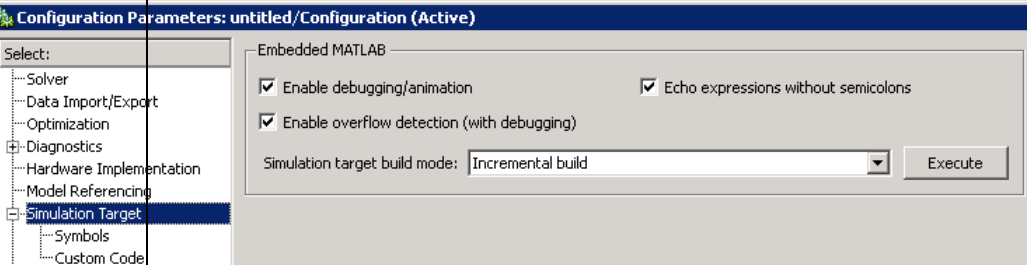
Type of Model	Simulation Options	Embeddable Code Generation Options
Nonlibrary	<p>Migrated from the Simulation Target dialog box to the Configuration Parameters dialog box.</p> <p>See:</p> <ul style="list-style-type: none"> • “Nonlibrary Models: Changes for the General Pane of the Simulation Target Dialog Box” on page 430 • “Nonlibrary Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box” on page 432 • “Nonlibrary Models: Changes for the Description Pane of the Simulation Target Dialog Box” on page 433 	<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 442 • “Nonlibrary Models: Enhancement for the Real-Time Workshop: Custom Code Pane of the Configuration Parameters Dialog Box” on page 443
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 437 • “Library Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box” on page 438 • “Library Models: Changes for the Description Pane of the Simulation Target Dialog Box” on page 439 	<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 443.</p>

For details about the new options, see Configuration Parameters Dialog Box in the Simulink Graphical User Interface documentation. For compatibility information, see .

For changes specific to Stateflow, see Unified Simulation and Embeddable Code Generation Options for Stateflow Charts and Truth Table Blocks in the Stateflow and Stateflow Coder release notes.

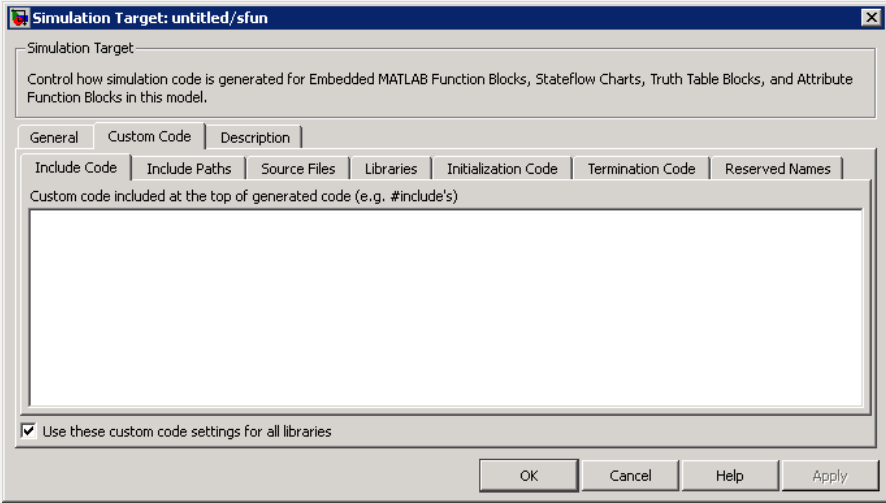
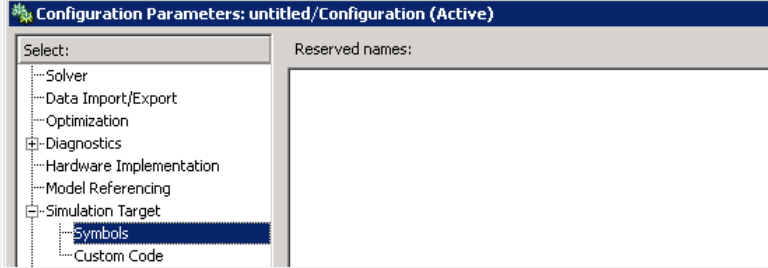
Nonlibrary Models: Changes for the General Pane of the Simulation Target Dialog Box

The following sections describe changes in the panes of the Simulation Target dialog box for nonlibrary models.

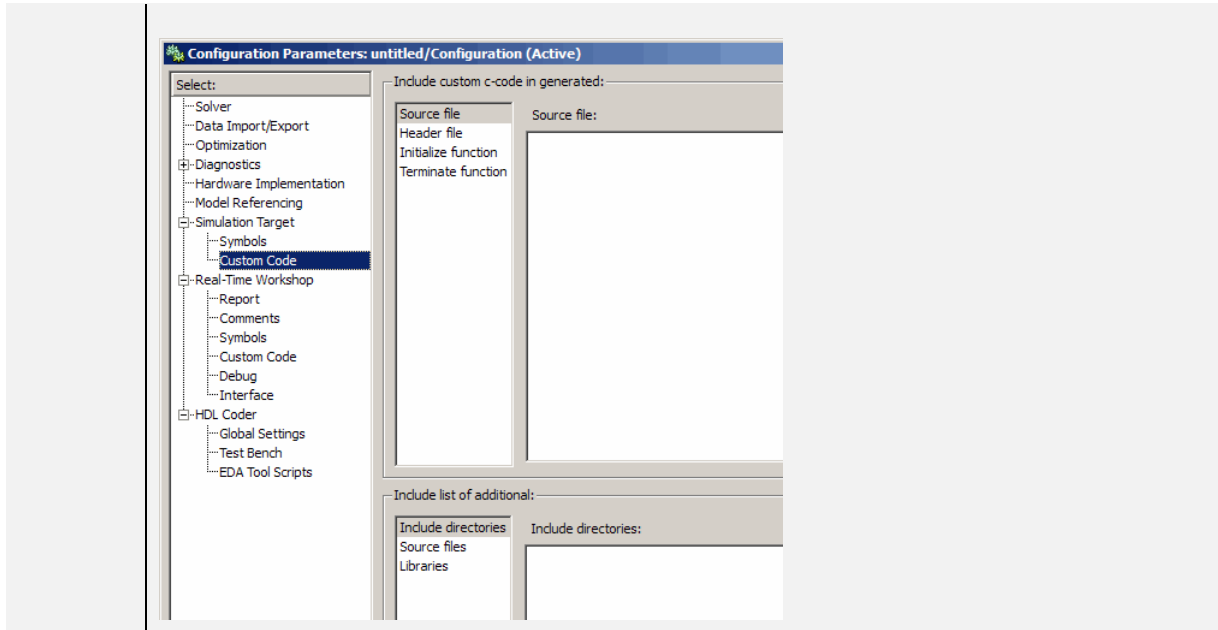
Release	Appearance
Previous	<p data-bbox="278 305 902 331">General pane of the Simulation Target dialog box</p> 
New	<p data-bbox="278 1012 1139 1038">Simulation Target pane of the Configuration Parameters dialog box</p> 

For details, see “Nonlibrary Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box” on page 434.

Nonlibrary Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box

Release	Appearance
Previous	<p data-bbox="278 395 976 423">Custom Code pane of the Simulation Target dialog box</p> 
New	<p data-bbox="278 980 1288 1008">Simulation Target > Symbols pane of the Configuration Parameters dialog box</p> 
New	<p data-bbox="278 1345 1307 1402">Simulation Target > Custom Code pane of the Configuration Parameters dialog box</p>

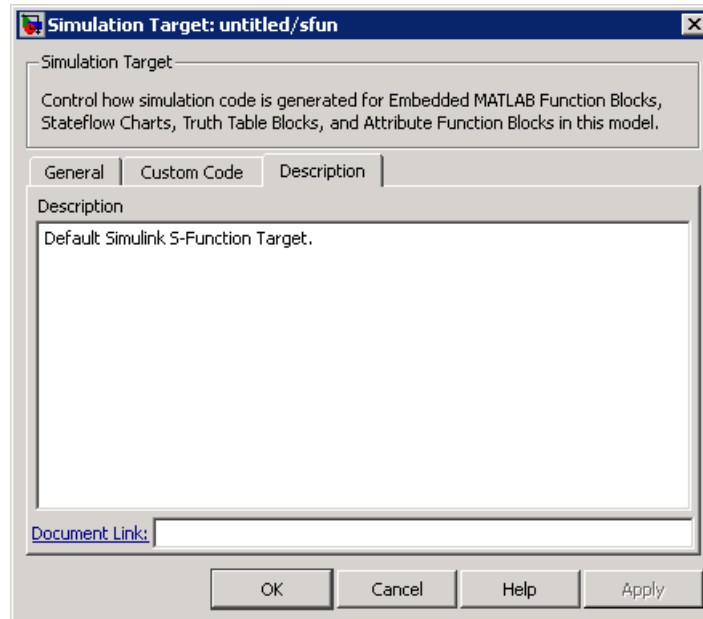
Release	Appearance
----------------	-------------------



For details, see “Nonlibrary Models: Mapping of GUI Options from the Simulation Target Dialog Box to the Configuration Parameters Dialog Box” on page 434.

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.

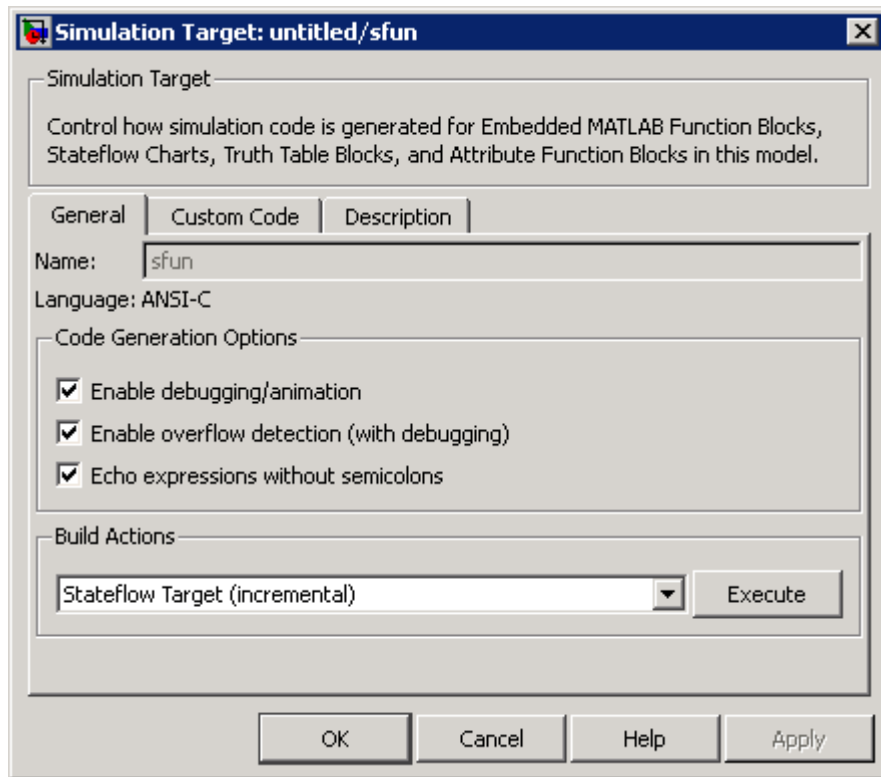
Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
General > Enable debugging / animation	Simulation Target > Enable debugging / animation	on
General > Enable overflow detection (with debugging)	Simulation Target > Enable overflow 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	{}

Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
Custom Code > Use these custom code settings for all libraries	None	Not applicable
Description > Description	<p data-bbox="540 475 609 501">None</p> <hr data-bbox="540 562 921 565"/> <p data-bbox="540 571 914 951">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.</p> <hr data-bbox="540 961 921 965"/>	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.

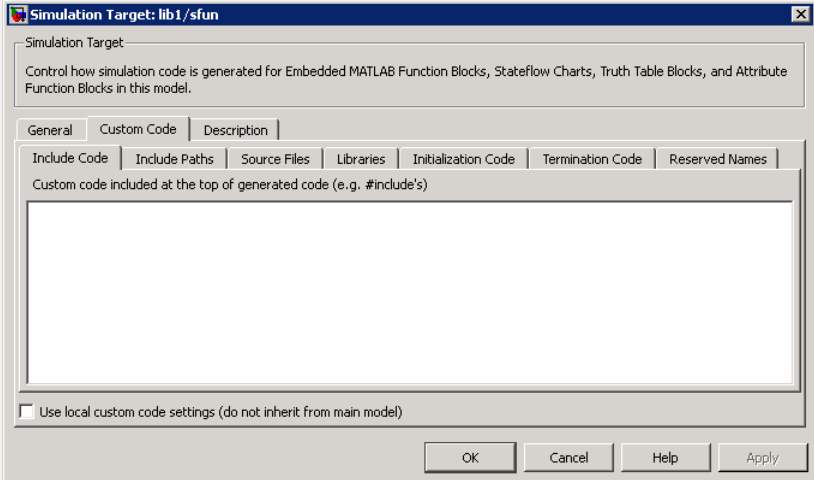
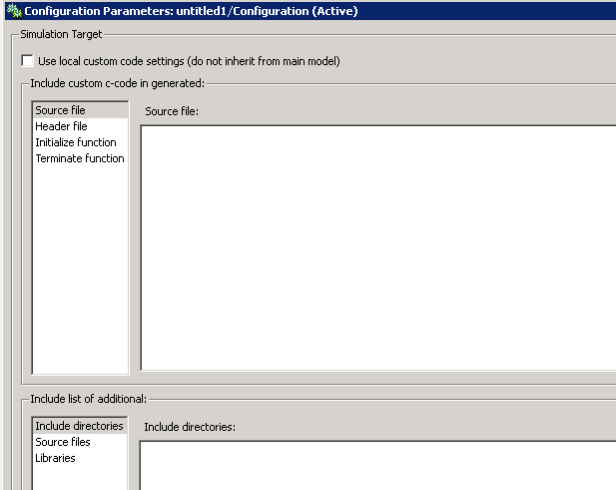
Library Models: Changes for the General Pane of the Simulation Target Dialog Box

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.

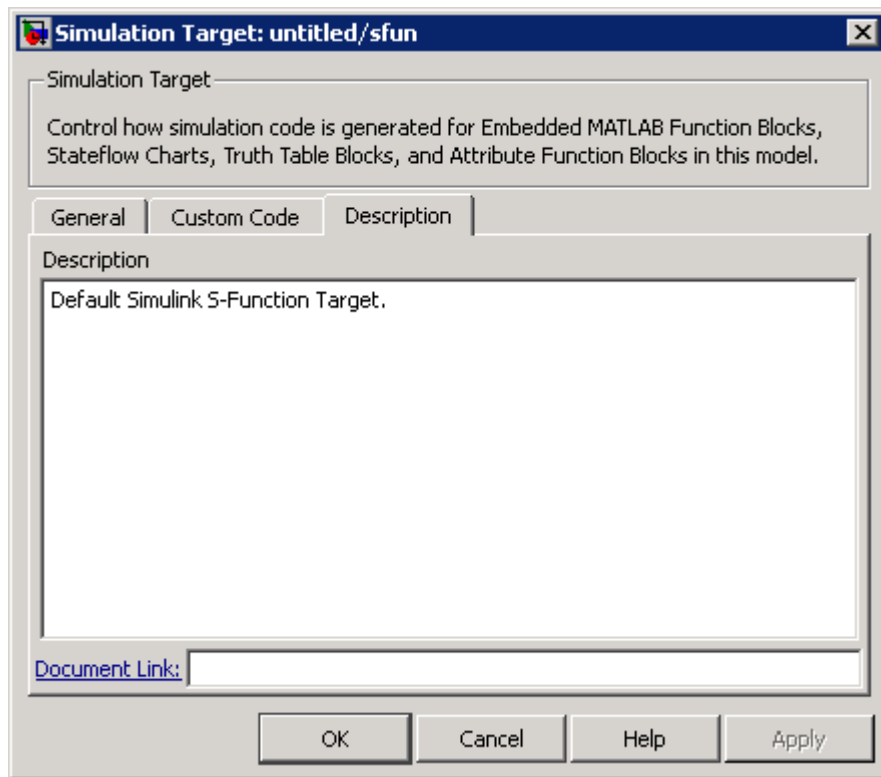
Library Models: Changes for the Custom Code Pane of the Simulation Target Dialog Box

Release	Appearance
Previous	<p data-bbox="278 395 976 423">Custom Code pane of the Simulation Target dialog box</p> 
New	<p data-bbox="278 963 1139 991">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 440.

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.

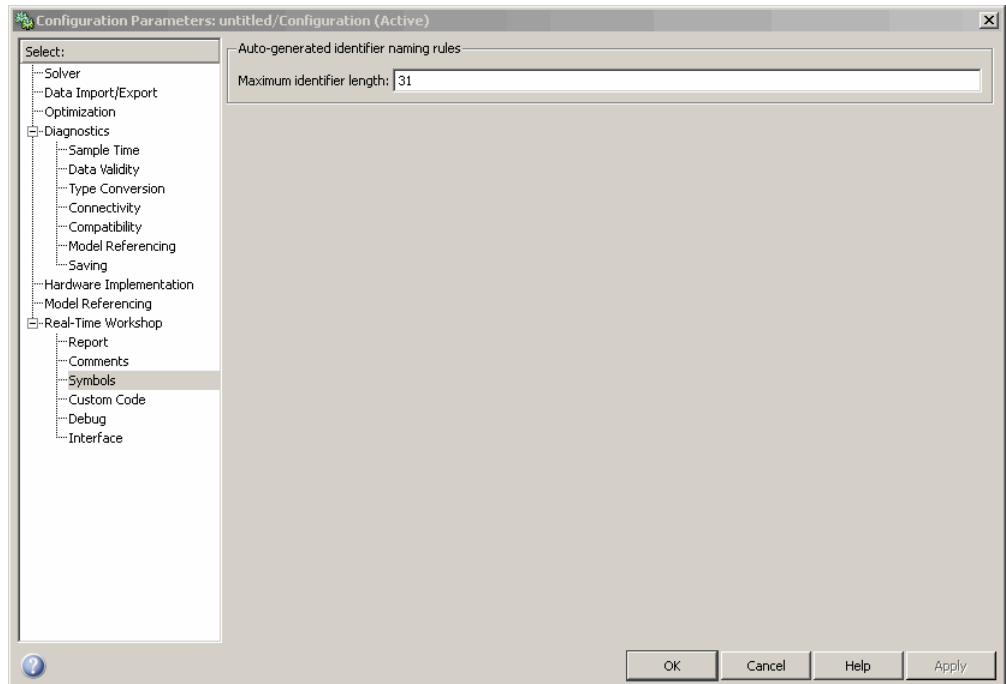
Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
General > Enable debugging / animation	None	Not applicable
General > Enable overflow detection (with debugging)	None	Not applicable
General > Echo expressions without semicolons	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	' '
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

Old Option in the Simulation Target Dialog Box	New Option in the Configuration Parameters Dialog Box	Default Value of New Option
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.

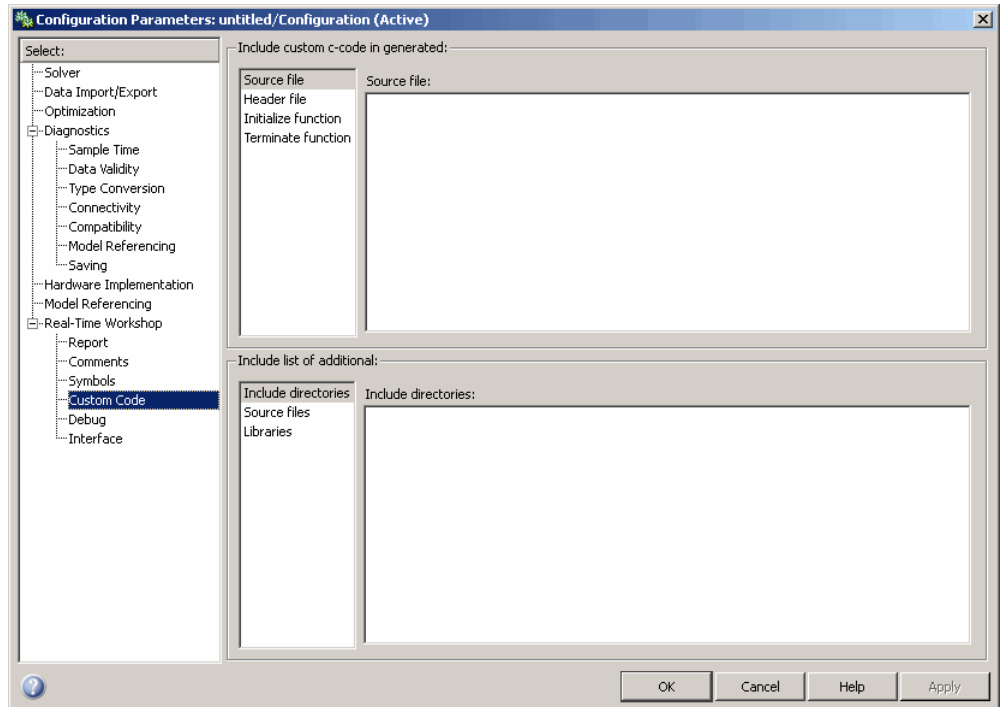


In R2008b, a new option is available in this pane: **Reserved names**. You can use this option to specify a set of keywords that the Real-Time Workshop build process should not use. This action prevents naming conflicts between functions and variables from external environments and identifiers in the 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

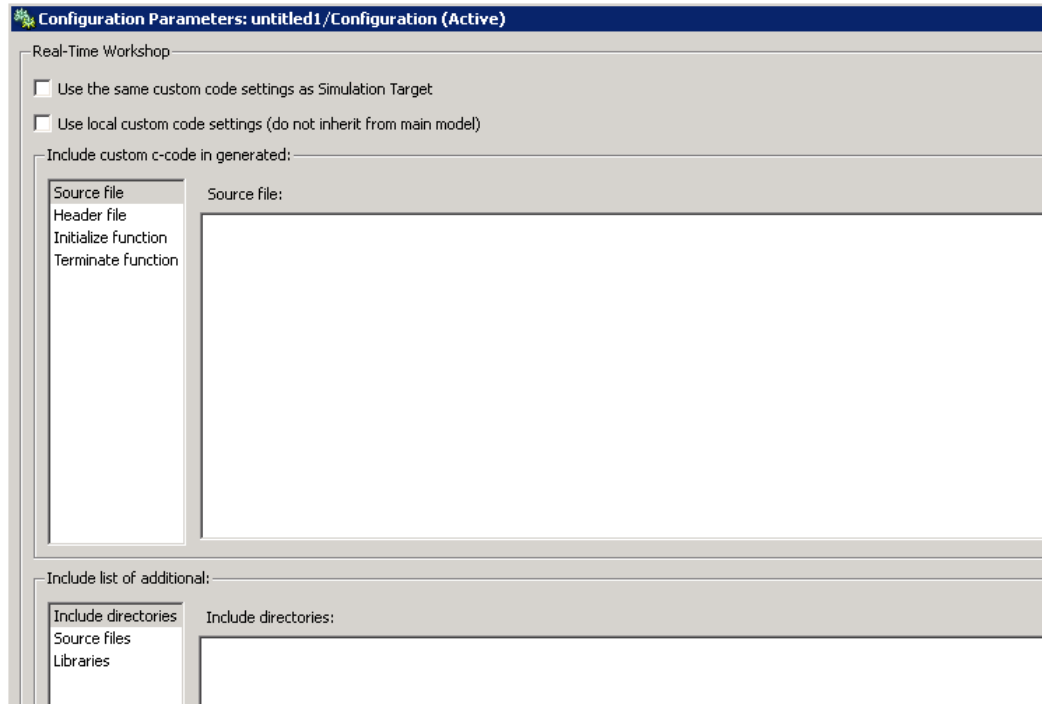
In previous releases, the **Real-Time Workshop > Custom Code** pane of the Configuration Parameters dialog box appeared as follows.



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

Compatibility Considerations: Yes

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 .

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
CodeFlagsInfo ('debug')	General > Enable debugging / animation	SFSimEnableDebug string - off, on	Simulation Target > Enable debugging / animation
CodeFlagsInfo ('overflow')	General > Enable overflow detection (with debugging)	SFSimOverflowDetection string - off, on	Simulation Target > Enable overflow detection (with debugging)
CodeFlagsInfo ('echo')	General > Echo expressions without semicolons	SFSimEcho string - off, on	Simulation Target > Echo expressions without semicolons
CustomCode	Custom Code > Include Code	SimCustomHeaderCode string - ''	Simulation Target > Custom Code > Header file
CustomInitializer	Custom Code > Initialization Code	SimCustomInitializer string - ''	Simulation Target > Custom Code > Initialize function

Old sfun Object Property	Old Option in the Simulation Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
CustomTerminator	Custom Code > Termination Code	SimCustomTerminator <i>string</i> - ''	Simulation Target > Custom Code > Terminate function
ReservedNames	Custom Code > Reserved Names	SimReservedNameArray <i>string array</i> - {}	Simulation Target > Symbols > Reserved names
UserIncludeDirs	Custom Code > Include Paths	SimUserIncludeDirs <i>string</i> - ''	Simulation Target > Custom Code > Include directories
UserLibraries	Custom Code > Libraries	SimUserLibraries <i>string</i> - ''	Simulation Target > Custom Code > Libraries
UserSources	Custom Code > Source Files	SimUserSources <i>string</i> - ''	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 <i>string</i> - ''	Simulation Target > Header file
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</i> - off , on	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 <code>rtw</code> Object Property	Old Option in the RTW Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
<code>CustomCode</code>	Custom Code > Include Code	<code>CustomHeaderCode</code> <i>string</i> - ''	Real-Time Workshop > Header file
<code>CustomInitializer</code>	Custom Code > Initialization Code	<code>CustomInitializer</code> <i>string</i> - ''	Real-Time Workshop > Initialize function
<code>CustomTerminator</code>	Custom Code > Termination Code	<code>CustomTerminator</code> <i>string</i> - ''	Real-Time Workshop > Terminate function
<code>UseLocalCustomCodeSettings</code>	Custom Code > Use local custom code settings (do not inherit from main model)	<code>RTWUseLocalCustomCode</code> <i>string</i> - off , on	Real-Time Workshop > Use local custom code settings (do not inherit from main model)
<code>UserIncludeDirs</code>	Custom Code > Include Paths	<code>CustomInclude</code> <i>string</i> - ''	Real-Time Workshop > Include directories

Old rtw Object Property	Old Option in the RTW Target Dialog Box	New Configuration Parameter	New Option in the Configuration Parameters Dialog Box
UserLibraries	Custom Code > Libraries	CustomLibrary <i>string - ''</i>	Real-Time Workshop > Libraries
UserSources	Custom Code > Source Files	CustomSource <i>string - ''</i>	Real-Time Workshop > Source files

Compatibility Considerations

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
<code>SimBuildMode</code> string – <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.
<code>SimCustomSourceCode</code> string - ''	Simulation Target > Custom Code > Source file	Enter code lines to appear near the top of a generated source code file.

New Simulation Parameter for Library Models

The following table lists the new simulation parameter that applies to library models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
SimCustomSourceCode <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> – off , on	Real-Time Workshop > Custom Code > Use the same custom code settings as Simulation Target	Specify whether to use the same custom code settings as those specified for simulation.
UseSimReservedNames <i>string</i> – off , on	Real-Time Workshop > Symbols > Use the same reserved names as Simulation Target	Specify whether to use the same reserved names as those specified for simulation.

New Code Generation Parameters for Library Models

The following table lists the new code generation parameters that apply to library models.

New Configuration Parameter	New Option in the Configuration Parameters Dialog Box	Description
CustomSourceCode <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

Compatibility Considerations: Yes

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

Compatibility Considerations

- 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.

Handle Graphics 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.

For more information, see “[Changes to -nojvm Startup Option](#)” in the Desktop Tools and Development Environment release notes.

R2008a

Version: 7.1

New Features: Yes

Bug Fixes: Yes

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

Previously, you could omit scaling in data type definitions for fixed-point data. Such data types were treated as integers with the specified sign and word length. This behavior has changed. 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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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](#).

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

In models with asynchronous function calls, some virtual blocks now correctly assign generic sample times instead of triggered sample times.

Compatibility Considerations

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 in the Simulink Graphical User Interface documentation.

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 in the Simulink Graphical User Interface documentation.

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 in the Using Simulink documentation.

Block Enhancements

New Discrete FIR Filter Block Replaces Weighted Moving Average Block

Compatibility Considerations: Yes

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.

Compatibility Considerations

You should replace Weighted Moving Average blocks in your existing models with the Discrete FIR Filter block. To do this, run the `supdate` 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 in the Simulink Graphical User Interface documentation.

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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: Yes

Bug Fixes: Yes

Simulation Performance

Simulink Accelerator

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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*.

Compatibility Considerations

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 **Compatibility Considerations: Yes**

The Mask Editor now requires that the MATLAB product start with Java enabled. See Simulink Mask Editor in Simulink User's Guide.

Compatibility Considerations

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 Compatibility Considerations: Yes

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.

Compatibility Considerations

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 490). 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 490). 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 **Compatibility Considerations: Yes**

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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 `msfcndemo_varpulse.mdl` uses the S-function `msfcn_varpulse.m` to create a variable-width pulse generator.

R2007a+

Version: 6.6.1

New Features: No

Bug Fixes: Yes

R2007a

Version: 6.6

New Features: Yes

Bug Fixes: Yes

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 502
- “Multidimensional Signals in S-Functions” on page 505

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 in the Simulink documentation 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` in the Simulink documentation for details.
- `PreDeleteFcn`: Allows you to assign a function to call before the block is deleted. See `Block Callback Parameters` in the Simulink documentation 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` in the Simulink documentation for details.

GNU Compiler Upgrade

Compatibility Considerations: Yes

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.

Compatibility Considerations

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 in the Simulink documentation*.

Change to Simulink.ModelAdvisor.getModelAdvisor Method

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

- 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:
 - `sldemo_lct_sampletime`
 - `sldemo_lct_work`
 - `sldemo_lct_cplxgain`
 - `sldemo_lct_ndarray`

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](#)

Compatibility Considerations

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 513
- “Support for Multidimensional Arrays” on page 513

- “Support for Function Handles” on page 513
- “Enhanced Support for Frames” on page 514
- “New Embedded MATLAB Runtime Library Functions” on page 514
- “Using & and | Operators in Embedded MATLAB Function Blocks” on page 516
- “Calling get Function from Embedded MATLAB Function Blocks” on page 517
- “Documentation on Embedded MATLAB Subset has Moved” on page 517

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 in the Embedded MATLAB documentation.

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* in the Simulink documentation.

New Embedded MATLAB Runtime Library Functions

Embedded MATLAB Function blocks provide 31 new runtime library functions in the following categories:

- “Casting Functions” on page 514
- “Derivative and Integral Functions” on page 514
- “Discrete Math Functions” on page 515
- “Exponential Functions” on page 515
- “Filtering and Convolution Functions” on page 515
- “Logical Operator Functions” on page 515
- “Matrix and Array Functions” on page 515
- “Polynomial Functions” on page 515
- “Set Functions” on page 516
- “Specialized Math” on page 516
- “Statistical Functions” on page 516

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`
- `betaln`
- `ellipke`
- `erf`
- `erfc`
- `erfcinv`
- `erfcx`
- `erfinv`
- `expint`
- `gamma`
- `gammainc`
- `gammaln`

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.

Compatibility Considerations

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.

Compatibility Considerations

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 in the Embedded MATLAB documentation.

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 and Specifying a Simulation Start and Stop Time for information about these capabilities. See "Referencing Configuration Sets" on page 520 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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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 Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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 533 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

Compatibility Considerations: Yes

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.

Compatibility Considerations

Using `Simulink.ConfigSet.SimulationMode` is not recommended. Use `set_param(modeName, 'SimulationMode', value)` instead.

R2006b

Version: 6.5

New Features: Yes

Bug Fixes: Yes

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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. See [Determine Why Subsystem Code Is Not Reused](#).

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 **Compatibility Considerations: Yes**

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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).

Compatibility Considerations

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 **Compatibility Considerations: Yes**

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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 536
- “Discrete Math” on page 536
- “Exponential” on page 536
- “Interpolation and Computational Geometry” on page 536
- “Linear Algebra” on page 536
- “Logical” on page 537
- “Specialized Plotting” on page 537
- “Transforms” on page 537
- “Trigonometric” on page 537

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 in the Embedded MATLAB documentation](#).

Compatibility Considerations

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.

Compatibility Considerations

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.

Compatibility Considerations

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

New Features: No

Bug Fixes: No

No New Features or Changes

R2006a

Version: 6.4

New Features: Yes

Bug Fixes: No

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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.

Compatibility Considerations

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**Compatibility Considerations: Yes**

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.

Compatibility Considerations

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

Compatibility Considerations: Yes

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 MATLAB 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 550.

New Embedded MATLAB Runtime Library Functions

Embedded MATLAB Function blocks provide new runtime library functions in the following categories:

- “Integer Arithmetic” on page 550
- “Linear Algebra” on page 551
- “Logical” on page 551
- “Polynomial” on page 552
- “Trigonometric” on page 552

Integer Arithmetic

- `idivide`

Linear Algebra

- `compan`
- `dot`
- `eig`
- `flip1r`
- `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.

Compatibility Considerations

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 `slupdate` either from the Model Advisor or by typing the following command at the MATLAB command line:

```
slupdate ('modelName')
```

where `'modelName'` is the name of the model containing the Embedded MATLAB Function block that generates the error. `slupdate` 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 slupdate 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.

Compatibility Considerations

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.

Compatibility Considerations

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: Yes

Bug Fixes: No

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 in the Simulink documentation 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](#) in the Simulink documentation 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 `demos` at the command prompt, then navigating to the Simulink demos category and browsing the demos.

Block Enhancements

Variable Transport Delay, Variable Time Delay Blocks

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

Additional Reset Trigger for Discrete-Time Integrator Block

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

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

Input Port Latching Enhancements

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

Label Clarified for Triggered Subsystem Latch Option

The dialog box for an Inport block contains a check box to latch the signal connected to the system via the port. This check box applies only to triggered subsystems and hence is enabled only when the Inport block resides in a triggered subsystem. In this release, the label for the check box that selects this option has changed from **Latch (buffer) input** to **Latch input by delaying outside signal**. This change is intended to make it clear what the

option does, i.e., cause the subsystem to see the input signal's value at the previous time step when the subsystem executes at the current time step (equivalent to inserting a Memory block at the input outside the subsystem). The Inport block's icon displays <Lo> to indicate that this option is selected.

Latch Option Added for Function-Call Subsystems

This release adds a check box labeled **Latch input by copying inside signal** to the Inport block's dialog box. This option applies only to function-call subsystems and hence is enabled only if the Inport block resides in a function-call subsystem. Selecting this option causes Simulink 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 561).

Parameter Object Expressions No Longer Supported in Dialog Boxes

Compatibility Considerations: Yes**Compatibility Considerations**

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 supports 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

Compatibility Considerations: Yes

Compatibility Considerations

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

Compatibility Considerations: Yes

In this release, the extension for files created by the MATLAB `mex` command on Windows systems has changed from `dll` to `mexw32` or `mexw64`.

Compatibility Considerations

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

Compatibility Considerations: Yes

In this release, the extension for files created by the MATLAB `mex` command has changed from `dll` to `mexw32` (and `mexw64`).

Compatibility Considerations

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: Yes

Bug Fixes: No

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 Compatibility Considerations: Yes

Compatibility Considerations

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.