

Simulink®

Graphical User Interface



MATLAB® & SIMULINK®

R2022b



How to Contact MathWorks



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



Phone: 508-647-7000



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

Simulink® Graphical User Interface

© COPYRIGHT 1990–2022 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

September 2007	Online only	New for Simulink 7.0 (Release 2007b)
March 2008	Online only	Revised for Simulink 7.1 (Release 2008a)
October 2008	Online only	Revised for Simulink 7.2 (Release 2008b)
March 2009	Online only	Revised for Simulink 7.3 (Release 2009a)
September 2009	Online only	Revised for Simulink 7.4 (Release 2009b)
March 2010	Online only	Revised for Simulink 7.5 (Release 2010a)
September 2010	Online only	Revised for Simulink 7.6 (Release 2010b)
April 2011	Online only	Revised for Simulink 7.7 (Release 2011a)
September 2011	Online only	Revised for Simulink 7.8 (Release 2011b)
March 2012	Online only	Revised for Simulink 7.9 (Release 2012a)
September 2012	Online only	Revised for Simulink 8.0 (Release 2012b)
March 2013	Online only	Revised for Simulink 8.1 (Release 2013a)
September 2013	Online only	Revised for Simulink 8.2 (Release 2013b)
March 2014	Online only	Revised for Simulink 8.3 (Release 2014a)
October 2014	Online only	Revised for Simulink 8.4 (Release 2014b)
March 2015	Online only	Revised for Simulink 8.5 (Release 2015a)
September 2015	Online only	Revised for Simulink 8.6 (Release 2015b)
October 2015	Online only	Rereleased for Simulink 8.5.1 (Release 2015aSP1)
March 2016	Online only	Revised for Simulink 8.7 (Release 2016a)
September 2016	Online only	Revised for Simulink 8.8 (Release 2016b)
March 2017	Online only	Revised for Simulink 8.9 (Release 2017a)
September 2017	Online only	Revised for Simulink 9.0 (Release 2017b)
March 2018	Online only	Revised for Simulink 9.1 (Release 2018a)
September 2018	Online only	Revised for Simulink 9.2 (Release 2018b)
March 2019	Online only	Revised for Simulink 9.3 (Release 2019a)
September 2019	Online only	Revised for Simulink 10.0 (Release 2019b)
March 2020	Online only	Revised for Simulink 10.1 (Release 2020a)
September 2020	Online only	Revised for Simulink 10.2 (Release 2020b)
March 2021	Online only	Revised for Simulink 10.3 (Release 2021a)
September 2021	Online only	Revised for Simulink 10.4 (Release 2021b)
March 2022	Online only	Revised for Version 10.5 (Release 2022a)
September 2022	Online only	Revised for Version 10.6 (Release 2022b)

Configuration Parameters Dialog Box

1

Model Configuration Pane	1-2
Model Configuration Overview	1-2
Name	1-2
Description	1-2
Configuration Parameters	1-3

Simulink Configuration Parameters: Advanced

2

Test hardware is the same as production hardware	2-2
Description	2-2
Settings	2-2
Tip	2-2
Dependency	2-2
Recommended settings	2-2
Test device vendor and type	2-3
Description	2-3
Settings	2-3
Tips	2-6
Dependencies	2-13
Command-Line Information	2-13
Recommended Settings	2-14
Number of bits: char	2-15
Description	2-15
Settings	2-15
Tip	2-15
Dependencies	2-15
Command-Line Information	2-15
Recommended Settings	2-15
Number of bits: short	2-17
Description	2-17
Settings	2-17
Tip	2-17
Dependencies	2-17
Command-Line Information	2-17
Recommended Settings	2-17

Number of bits: int	2-19
Description	2-19
Settings	2-19
Tip	2-19
Dependencies	2-19
Command-Line Information	2-19
Recommended Settings	2-19
Number of bits: long	2-21
Description	2-21
Settings	2-21
Tip	2-21
Dependencies	2-21
Command-Line Information	2-21
Recommended Settings	2-21
Number of bits: long long	2-23
Description	2-23
Settings	2-23
Tips	2-23
Dependencies	2-23
Command-Line Information	2-23
Recommended Settings	2-23
Number of bits: float	2-25
Description	2-25
Settings	2-25
Command-Line Information	2-25
Recommended Settings	2-25
Number of bits: double	2-26
Description	2-26
Settings	2-26
Command-Line Information	2-26
Recommended Settings	2-26
Number of bits: native	2-27
Description	2-27
Settings	2-27
Tip	2-27
Dependencies	2-27
Command-Line Information	2-27
Recommended Settings	2-27
Number of bits: pointer	2-29
Description	2-29
Settings	2-29
Dependencies	2-29
Command-Line Information	2-29
Recommended Settings	2-29
Number of bits: size_t	2-30
Description	2-30
Settings	2-30
Dependencies	2-30

Command-Line Information	2-30
Recommended Settings	2-30
Number of bits: ptrdiff_t	2-32
Description	2-32
Settings	2-32
Dependencies	2-32
Command-Line Information	2-32
Recommended Settings	2-32
Largest atomic size: integer	2-34
Description	2-34
Settings	2-34
Tip	2-34
Dependencies	2-34
Command-Line Information	2-34
Recommended Settings	2-35
Largest atomic size: floating-point	2-36
Description	2-36
Settings	2-36
Tip	2-36
Dependencies	2-36
Command-Line Information	2-36
Recommended Settings	2-36
Byte ordering	2-38
Description	2-38
Settings	2-38
Dependencies	2-38
Command-Line Information	2-38
Recommended Settings	2-38
Signed integer division rounds to	2-40
Description	2-40
Settings	2-40
Tips	2-40
Dependency	2-41
Command-Line Information	2-41
Recommended settings	2-41
Shift right on a signed integer as arithmetic shift	2-42
Description	2-42
Settings	2-42
Tips	2-42
Dependency	2-42
Command-Line Information	2-42
Recommended settings	2-42
Support long long	2-44
Description	2-44
Settings	2-44
Tips	2-44
Dependencies	2-44
Command-Line Information	2-44

Recommended Settings	2-44
Allowed unit systems	2-46
Description	2-46
Settings	2-46
Tip	2-46
Command-Line Information	2-47
Units inconsistency messages	2-48
Description	2-48
Settings	2-48
Command-Line Information	2-48
Allow automatic unit conversions	2-49
Description	2-49
Settings	2-49
Command-Line Information	2-49
Dataset signal format	2-50
Description	2-50
Settings	2-50
Comparison of Formats	2-50
Tips	2-52
Programmatic Use	2-52
Recommended Settings	2-53
Stream To Workspace blocks	2-54
Description	2-54
Settings	2-54
Tips	2-54
Programmatic Use	2-54
Recommended Settings	2-55
Array bounds exceeded	2-56
Description	2-56
Settings	2-56
Tips	2-56
Command-Line Information	2-56
Recommended Settings	2-57
Model Verification block enabling	2-58
Description	2-58
Settings	2-58
Dependency	2-58
Command-Line Information	2-58
Recommended Settings	2-58
Check undefined subsystem initial output	2-60
Description	2-60
Settings	2-60
Tips	2-60
Dependency	2-61
Command-Line Information	2-61
Recommended Settings	2-62

Detect multiple driving blocks executing at the same time step	2-63
Description	2-63
Settings	2-63
Tips	2-63
Command-Line Information	2-63
Recommended Settings	2-63
Underspecified initialization detection	2-65
Description	2-65
Settings	2-65
Tips	2-65
Dependencies	2-65
Command-Line Information	2-66
Recommended Settings	2-66
Solver data inconsistency	2-67
Description	2-67
Settings	2-67
Tips	2-67
Command-Line Information	2-68
Recommended Settings	2-68
Ignored zero crossings	2-69
Description	2-69
Settings	2-69
Tips	2-69
Command-Line Information	2-69
Recommended Settings	2-70
Masked zero crossings	2-71
Description	2-71
Settings	2-71
Tips	2-71
Command-Line Information	2-71
Recommended Settings	2-71
Block diagram contains disabled library links	2-72
Description	2-72
Settings	2-72
Tip	2-72
Command-Line Information	2-72
Recommended Settings	2-72
Block diagram contains parameterized library links	2-74
Description	2-74
Settings	2-74
Tips	2-74
Command-Line Information	2-74
Recommended Settings	2-74
Initial state is array	2-75
Description	2-75
Settings	2-75
Tips	2-75
Command-Line Information	2-75

Recommended Settings	2-75
Insufficient maximum identifier length	2-77
Description	2-77
Settings	2-77
Command-Line Information	2-77
Recommended Settings	2-77
Import custom code	2-79
Description	2-79
Settings	2-79
Command-Line Information	2-80
Recommended Settings	2-80
Compiler optimization level	2-81
Description	2-81
Settings	2-81
Tips	2-81
Command-Line Information	2-81
Recommended Settings	2-81
Verbose accelerator builds	2-83
Description	2-83
Settings	2-83
Command-Line Information	2-83
Recommended Settings	2-83
Implement logic signals as Boolean data (vs. double)	2-84
Description	2-84
Settings	2-84
Tips	2-84
Dependencies	2-84
Command-Line Information	2-85
Recommended Settings	2-85
Block reduction	2-86
Description	2-86
Settings	2-86
Tips	2-86
Dead Code Elimination	2-86
Highlight Reduced Blocks	2-87
Command-Line Information	2-88
Recommended Settings	2-89
Conditional input branch execution	2-90
Description	2-90
Settings	2-90
Command-Line Information	2-90
Recommended Settings	2-90
Break on Ctrl+C	2-92
Description	2-92
Settings	2-92
Command-Line Information	2-92
Recommended Settings	2-92

Compile-time recursion limit for MATLAB functions	2-93
Description	2-93
Settings	2-93
Command-Line Information	2-93
Enable implicit expansion in MATLAB functions	2-94
Description	2-94
Settings	2-94
Command-Line Information	2-94
Enable run-time recursion for MATLAB functions	2-95
Description	2-95
Settings	2-95
Command-Line Information	2-95
Dynamic memory allocation in MATLAB functions	2-96
Description	2-96
Settings	2-96
Dependency	2-96
Tips	2-96
Command-Line Information	2-96
Recommended Settings	2-97
Dynamic memory allocation threshold in MATLAB functions	2-98
Description	2-98
Settings	2-98
Dependency	2-98
Command-Line Information	2-98
Recommended Settings	2-98
Echo expressions without semicolons	2-100
Description	2-100
Settings	2-100
Tip	2-100
Command-Line Information	2-100
Recommended Settings	2-100
Enable continuous-time MATLAB functions to write to initialized persistent variables	2-101
Description	2-101
Settings	2-101
Tips	2-101
Command-Line Information	2-101
Recommended Settings	2-101
Allow setting breakpoints during simulation	2-103
Description	2-103
Settings	2-103
Tips	2-103
Command-Line Information	2-103
Recommended Settings	2-103
Reserved names	2-105
Description	2-105
Settings	2-105

Tips	2-105
Command-Line Information	2-105
Recommended Settings	2-105
Enable memory integrity checks	2-107
Description	2-107
Settings	2-107
Tips	2-107
Command-Line Information	2-107
Recommended Settings	2-107
Generate typedefs for imported bus and enumeration types	2-109
Description	2-109
Settings	2-109
Tips	2-109
Command-Line Information	2-109
Use local custom code settings (do not inherit from main model)	2-110
Description	2-110
Settings	2-110
Dependency	2-110
Command-Line Information	2-110
Recommended Settings	2-110
Allow symbolic dimension specification	2-111
Description	2-111
Settings	2-111
Command-Line Information	2-111
Recommended Settings	2-111
Enable decoupled continuous integration	2-112
Description	2-112
Settings	2-112
Command-Line Information	2-112
Recommended Settings	2-112
Enable minimal zero-crossing impact integration	2-114
Settings	2-114
Dependencies	2-114
Tips	2-114
Command-Line Information	2-114
Recommended Settings	2-114
Detect ambiguous custom storage class final values	2-115
Description	2-115
Settings	2-115
Tip	2-115
Command-Line Information	2-115
Recommended Settings	2-116
Detect non-reused custom storage classes	2-117
Description	2-117
Settings	2-117
Tip	2-118
Command-Line Information	2-118

Recommended Settings	2-118
Combine output and update methods for code generation and simulation	2-119
Description	2-119
Settings	2-119
Tips	2-119
Command-Line Information	2-119
Recommended Settings	2-120
Include custom code for referenced models	2-121
Description	2-121
Settings	2-121
Tips	2-121
Command-Line Information	2-121
Recommended Settings	2-121
Hardware acceleration	2-122
Description	2-122
Settings	2-122
Command-Line Information	2-122
Recommended Settings	2-122
Behavior when pregenerated library subsystem code is missing	2-123
Description	2-123
Settings	2-123
Tip	2-123
Command-Line Information	2-123
Recommended Settings	2-123
Arithmetic operations in variant conditions	2-125
Description	2-125
Settings	2-125
Command-Line Information	2-125
Recommended Settings	2-125
Variant activation time inherited from Simulink.VariantControl	2-127
Description	2-127
Settings	2-127
Command-Line Information	2-127
Recommended Settings	2-127
FMU Import blocks	2-129
Description	2-129
Settings	2-129
Command-Line Information	2-129
Recommended Settings	2-129
Variant condition mismatch at signal source and destination	2-130
Description	2-130
Settings	2-130
Command-Line Information	2-130
Recommended Settings	2-130
Prevent Creation of Unused Variables for Lenient Variant Choices ...	2-132

Prevent Creation of Unused Variables for Unconditional and Conditional Variant Choices	2-135
Variant configuration not used by top model	2-138
Description	2-138
Settings	2-138
Command-Line Information	2-138
Recommended Settings	2-138

Data Import/Export Parameters

3

Model Configuration Parameters: Data Import/Export	3-2
Input	3-4
Description	3-4
Settings	3-4
Tips	3-4
Programmatic Use	3-4
Recommended Settings	3-5
Initial state	3-6
Description	3-6
Settings	3-6
Tips	3-6
Programmatic Use	3-7
Recommended Settings	3-7
Time	3-8
Description	3-8
Settings	3-8
Tips	3-8
Programmatic Use	3-8
Recommended Settings	3-9
States	3-10
Description	3-10
Settings	3-10
Tips	3-10
Programmatic Use	3-10
Recommended Settings	3-11
Output	3-12
Description	3-12
Settings	3-12
Tips	3-12
Programmatic Use	3-13
Recommended Settings	3-13
Final states	3-14
Description	3-14
Settings	3-14

Tips	3-14
Programmatic Use	3-14
Recommended Settings	3-15
Format	3-16
Description	3-16
Settings	3-16
Tips	3-16
Programmatic Use	3-17
Recommended Settings	3-17
Limit data points to last	3-18
Description	3-18
Settings	3-18
Tips	3-18
Programmatic Use	3-18
Recommended Settings	3-18
Decimation	3-20
Description	3-20
Settings	3-20
Tips	3-20
Programmatic Use	3-20
Recommended Settings	3-20
Save complete SimState in final state	3-22
Description	3-22
Settings	3-22
Tips	3-22
Dependencies	3-22
Programmatic Use	3-22
Recommended Settings	3-22
Save final operating point	3-24
Description	3-24
Settings	3-24
Tips	3-24
Dependencies	3-24
Programmatic Use	3-24
Recommended Settings	3-24
Signal logging	3-26
Description	3-26
Settings	3-26
Tips	3-26
Dependencies	3-27
Programmatic Use	3-27
Recommended Settings	3-27
Data stores	3-28
Description	3-28
Settings	3-28
Tips	3-28
Programmatic Use	3-28
Recommended Settings	3-28

Log Dataset data to file	3-30
Description	3-30
Settings	3-30
Tips	3-30
Programmatic Use	3-31
Recommended Settings	3-31
Output options	3-32
Description	3-32
Settings	3-32
Tips	3-32
Dependencies	3-32
Programmatic Use	3-32
Recommended Settings	3-33
Refine factor	3-34
Description	3-34
Settings	3-34
Tip	3-34
Dependency	3-34
Programmatic Use	3-34
Recommended Settings	3-34
Output times	3-35
Description	3-35
Settings	3-35
Tips	3-35
Dependency	3-35
Programmatic Use	3-35
Recommended Settings	3-35
Single simulation output	3-37
Description	3-37
Settings	3-37
Tips	3-37
Programmatic Use	3-37
Recommended Settings	3-38
Logging intervals	3-39
Description	3-39
Settings	3-39
Tips	3-39
Dependencies	3-40
Programmatic Use	3-40
Recommended Settings	3-40
Record logged workspace data in Simulation Data Inspector	3-41
Description	3-41
Settings	3-41
Tips	3-41
Programmatic Use	3-41
Recommended Settings	3-41

Diagnostics Parameters: Compatibility

4

Model Configuration Parameters: Compatibility Diagnostics	4-2
Compatibility Diagnostics Overview	4-3
Configuration	4-3
Tips	4-3
To get help on an option	4-3
S-function upgrades needed	4-4
Description	4-4
Settings	4-4
Command-Line Information	4-4
Recommended Settings	4-4
Block behavior depends on frame status of signal	4-5
Description	4-5
Settings	4-5
Tips	4-5
Command-Line Information	4-5
Recommended Settings	4-6
Operating point object from a different release	4-7
Description	4-7
Settings	4-7
Command-Line Information	4-7
Recommended Settings	4-7

Diagnostics Parameters: Connectivity

5

Model Configuration Parameters: Connectivity Diagnostics	5-2
Signal label mismatch	5-4
Description	5-4
Settings	5-4
Command-Line Information	5-4
Recommended Settings	5-4
Unconnected block input ports	5-5
Description	5-5
Settings	5-5
Command-Line Information	5-5
Recommended Settings	5-5
Unconnected block output ports	5-6
Description	5-6
Settings	5-6
Command-Line Information	5-6
Recommended Settings	5-6

Unconnected line	5-7
Description	5-7
Settings	5-7
Command-Line Information	5-7
Recommended Settings	5-7
Unspecified bus object at root Output block	5-8
Description	5-8
Settings	5-8
Tips	5-8
Command-Line Information	5-8
Recommended Settings	5-8
Element name mismatch	5-10
Description	5-10
Settings	5-10
Tips	5-10
Command-Line Information	5-10
Recommended Settings	5-10
Bus signal treated as vector	5-12
Description	5-12
Settings	5-12
Tips	5-12
Command-Line Information	5-12
Recommended Settings	5-13
Non-bus signals treated as bus signals	5-14
Description	5-14
Settings	5-14
Command-Line Information	5-14
Recommended Settings	5-14
Repair bus selections	5-16
Description	5-16
Settings	5-16
Command-Line Information	5-16
Recommended Settings	5-16
Context-dependent inputs	5-17
Description	5-17
Settings	5-17
Tips	5-17
Command-Line Information	5-17
Recommended Settings	5-17

Diagnostics Parameters: Data Validity

6

Model Configuration Parameters: Data Validity Diagnostics	6-2
--	-----

Data Validity Diagnostics Overview	6-5
Configuration	6-5
Tips	6-5
To get help on an option	6-5
Signal resolution	6-6
Description	6-6
Settings	6-6
Tips	6-6
Command-Line Information	6-7
Recommended Settings	6-7
Division by singular matrix	6-8
Description	6-8
Settings	6-8
Tips	6-8
Command-Line Information	6-8
Recommended Settings	6-8
Underspecified data types	6-10
Description	6-10
Identify and Resolve Underspecified Data Types	6-10
Settings	6-11
Command-Line Information	6-11
Recommended Settings	6-11
Simulation range checking	6-12
Description	6-12
Settings	6-12
Tips	6-12
Command-Line Information	6-12
Recommended Settings	6-12
String truncation checking	6-14
Description	6-14
Settings	6-14
Command-Line Information	6-14
Recommended Settings	6-14
Wrap on overflow	6-15
Description	6-15
Settings	6-15
Tips	6-15
Command-Line Information	6-16
Recommended Settings	6-16
Saturate on overflow	6-17
Description	6-17
Settings	6-17
Tips	6-17
Command-Line Information	6-17
Recommended Settings	6-17
Underspecified dimensions	6-19
Description	6-19

Settings	6-19
Command-Line Information	6-19
Recommended Settings	6-19
Inf or NaN block output	6-20
Description	6-20
Settings	6-20
Tips	6-20
Command-Line Information	6-20
Recommended Settings	6-21
"rt" prefix for identifiers	6-22
Description	6-22
Settings	6-22
Tips	6-22
Command-Line Information	6-22
Recommended Settings	6-22
Detect downcast	6-24
Description	6-24
Settings	6-24
Tips	6-24
Command-Line Information	6-24
Recommended Settings	6-24
Detect overflow	6-26
Description	6-26
Settings	6-26
Tips	6-26
Command-Line Information	6-26
Recommended Settings	6-27
Detect underflow	6-28
Description	6-28
Settings	6-28
Tips	6-28
Command-Line Information	6-28
Recommended Settings	6-28
Detect precision loss	6-30
Description	6-30
Settings	6-30
Tips	6-30
Command-Line Information	6-30
Recommended Settings	6-31
Detect loss of tunability	6-32
Description	6-32
Settings	6-32
Tips	6-32
Command-Line Information	6-32
Recommended Settings	6-32
Detect read before write	6-34
Description	6-34

Settings	6-34
Command-Line Information	6-34
Recommended Settings	6-35
Detect write after read	6-36
Description	6-36
Settings	6-36
Command-Line Information	6-36
Recommended Settings	6-37
Detect write after write	6-38
Description	6-38
Settings	6-38
Command-Line Information	6-38
Recommended Settings	6-39
Multitask data store	6-40
Description	6-40
Settings	6-40
Tips	6-40
Command-Line Information	6-40
Recommended Settings	6-40
Duplicate data store names	6-42
Description	6-42
Settings	6-42
Tip	6-42
Command-Line Information	6-42
Recommended Settings	6-42

Diagnostics Parameters: Model Referencing

7

Model Configuration Parameters: Model Referencing Diagnostics	7-2
Model block version mismatch	7-3
Description	7-3
Settings	7-3
Tips	7-3
Command-Line Information	7-3
Recommended Settings	7-3
Port and parameter mismatch	7-5
Description	7-5
Settings	7-5
Tips	7-5
Command-Line Information	7-5
Recommended Settings	7-5
Invalid root Inport/Outport block connection	7-7
Description	7-7
Settings	7-7

Tips	7-7
Command-Line Information	7-9
Recommended Settings	7-10
Unsupported data logging	7-11
Description	7-11
Settings	7-11
Tips	7-11
Command-Line Information	7-11
Recommended Settings	7-11
No explicit final value for model arguments	7-13
Description	7-13
Settings	7-13
Command-Line Information	7-13
Recommended Settings	7-13

Diagnostics Parameters: Sample Time

8

Model Configuration Parameters: Sample Time Diagnostics	8-2
Source block specifies -1 sample time	8-3
Description	8-3
Settings	8-3
Tips	8-3
Command-Line Information	8-3
Recommended Settings	8-3
Multitask data transfer	8-5
Description	8-5
Settings	8-5
Tips	8-5
Command-Line Information	8-5
Recommended Settings	8-5
Single task data transfer	8-7
Description	8-7
Settings	8-7
Tips	8-7
Command-Line Information	8-7
Recommended Settings	8-7
Multitask conditionally executed subsystem	8-9
Description	8-9
Settings	8-9
Tips	8-9
Command-Line Information	8-9
Recommended Settings	8-10
Tasks with equal priority	8-11
Description	8-11

Settings	8-11
Tips	8-11
Command-Line Information	8-11
Recommended Settings	8-11
Enforce sample times specified by Signal Specification blocks	8-13
Description	8-13
Settings	8-13
Tips	8-13
Command-Line Information	8-13
Recommended Settings	8-13
Exported tasks rate transition	8-15
Description	8-15
Settings	8-15
Command-Line Information	8-15
Recommended Settings	8-15
Unspecified inheritability of sample time	8-16
Description	8-16
Settings	8-16
Tips	8-16
Command-Line Information	8-16
Recommended Settings	8-16

Diagnostics Parameters

9

Model Configuration Parameters: Diagnostics	9-2
Algebraic loop	9-5
Description	9-5
Settings	9-5
Tips	9-5
Command-Line Information	9-6
Recommended Settings	9-6
Minimize algebraic loop	9-7
Description	9-7
Settings	9-7
Tips	9-7
Command-Line Information	9-7
Recommended Settings	9-7
Block priority violation	9-9
Description	9-9
Settings	9-9
Tips	9-9
Command-Line Information	9-9
Recommended Settings	9-9

Min step size violation	9-11
Description	9-11
Settings	9-11
Tips	9-11
Command-Line Information	9-11
Recommended Settings	9-11
Consecutive zero-crossings violation	9-13
Description	9-13
Settings	9-13
Tips	9-13
Dependency	9-13
Command-Line Information	9-13
Recommended Settings	9-13
Automatic solver parameter selection	9-15
Description	9-15
Settings	9-15
Tips	9-15
Command-Line Information	9-15
Recommended Settings	9-15
Extraneous discrete derivative signals	9-17
Description	9-17
Settings	9-17
Tips	9-17
Dependency	9-17
Command-Line Information	9-17
Recommended Settings	9-18
State name clash	9-19
Description	9-19
Settings	9-19
Tips	9-19
Command-Line Information	9-19
Recommended Settings	9-19
SimState interface checksum mismatch	9-21
Description	9-21
Settings	9-21
Command-Line Information	9-21
Recommended Settings	9-21
Operating point restore interface checksum mismatch	9-23
Description	9-23
Settings	9-23
Command-Line Information	9-23
Recommended Settings	9-23

Model Configuration Parameters: Stateflow Diagnostics	10-2
Unused data, events, messages, and functions	10-4
Description	10-4
Settings	10-4
Tip	10-4
Command-Line Information	10-4
Recommended Settings	10-4
Unexpected backtracking	10-6
Description	10-6
Settings	10-6
Tip	10-6
Command-Line Information	10-6
Recommended Settings	10-6
Invalid input data access in chart initialization	10-8
Description	10-8
Settings	10-8
Tip	10-8
Command-Line Information	10-8
Recommended Settings	10-8
No unconditional default transitions	10-10
Description	10-10
Settings	10-10
Command-Line Information	10-10
Recommended Settings	10-10
Transition outside natural parent	10-12
Description	10-12
Settings	10-12
Command-Line Information	10-12
Recommended Settings	10-12
Undirected event broadcasts	10-13
Description	10-13
Settings	10-13
Command-Line Information	10-13
Recommended Settings	10-13
Transition action specified before condition action	10-14
Description	10-14
Settings	10-14
Command-Line Information	10-14
Recommended Settings	10-14
Read-before-write to output in Moore chart	10-16
Description	10-16
Settings	10-16
Command-Line Information	10-16

Recommended Settings	10-16
Absolute time temporal value shorter than sampling period	10-17
Description	10-17
Settings	10-17
Command-Line Information	10-17
Recommended Settings	10-17
Self transition on leaf state	10-18
Description	10-18
Settings	10-18
Command-Line Information	10-18
Recommended Settings	10-18
Execute-at-Initialization disabled in presence of input events	10-19
Description	10-19
Settings	10-19
Command-Line Information	10-19
Recommended Settings	10-19
Use of machine-parented data instead of Data Store Memory	10-21
Description	10-21
Settings	10-21
Command-Line Information	10-21
Recommended Settings	10-21
Unreachable execution path	10-23
Description	10-23
Settings	10-24
Tip	10-24
Command-Line Information	10-24
Recommended Settings	10-24

Diagnostics Parameters: Type Conversion

11

Model Configuration Parameters: Type Conversion Diagnostics	11-2
Unnecessary type conversions	11-3
Description	11-3
Settings	11-3
Command-Line Information	11-3
Recommended Settings	11-3
Vector/matrix block input conversion	11-4
Description	11-4
Settings	11-4
Tips	11-4
Command-Line Information	11-4
Recommended Settings	11-4

32-bit integer to single precision float conversion	11-6
Description	11-6
Settings	11-6
Tip	11-6
Command-Line Information	11-6
Recommended Settings	11-6
Detect underflow	11-7
Description	11-7
Settings	11-7
Tips	11-7
Dependency	11-7
Command-Line Information	11-7
Recommended Settings	11-7
Detect precision loss	11-9
Description	11-9
Settings	11-9
Tips	11-9
Dependency	11-9
Command-Line Information	11-9
Recommended Settings	11-9
Detect overflow	11-11
Description	11-11
Settings	11-11
Tips	11-11
Dependency	11-11
Command-Line Information	11-11
Recommended Settings	11-11

Model Referencing Parameters

12

Model Configuration Parameters: Model Referencing	12-2
Rebuild	12-4
Description	12-4
Settings	12-4
Definitions	12-5
Tips	12-6
Dependency	12-7
Command-Line Information	12-7
Recommended Settings	12-7
Compatibility Considerations	12-7
Never rebuild diagnostic	12-9
Description	12-9
Settings	12-9
Tip	12-9
Dependency	12-9
Command-Line Information	12-9

Recommended Settings	12-10
Enable parallel model reference builds	12-11
Description	12-11
Settings	12-11
Dependency	12-11
Tip	12-11
Command-Line Information	12-11
Recommended Settings	12-11
MATLAB worker initialization for builds	12-13
Description	12-13
Settings	12-13
Limitation	12-13
Dependency	12-13
Command-Line Information	12-13
Recommended Settings	12-13
Enable strict scheduling checks for referenced models	12-15
Description	12-15
Settings	12-15
Command-Line Information	12-15
Total number of instances allowed per top model	12-16
Description	12-16
Settings	12-16
Command-Line Information	12-16
Recommended Settings	12-16
Pass fixed-size scalar root inputs by value for code generation	12-18
Description	12-18
Settings	12-18
Tips	12-18
Command-Line Information	12-18
Recommended Settings	12-19
Minimize algebraic loop occurrences	12-20
Description	12-20
Settings	12-20
Tips	12-20
Command-Line Information	12-20
Recommended Settings	12-20
Propagate all signal labels out of the model	12-22
Description	12-22
Settings	12-22
Tips	12-22
Command-Line Information	12-23
Recommended Settings	12-23
Use local solver when referencing model	12-25
Description	12-25
Settings	12-25
Tips	12-25
Command-Line Information	12-25

Recommended Settings	12-25
Propagate sizes of variable-size signals	12-27
Description	12-27
Settings	12-27
Command-Line Information	12-28
Recommended Settings	12-28
Model dependencies	12-29
Description	12-29
Settings	12-29
Tips	12-30
Command-Line Information	12-30
Recommended Settings	12-30
Perform consistency check on parallel pool	12-31
Description	12-31
Settings	12-31
Command-Line Information	12-31

Simulation Target Parameters

13

Model Configuration Parameters: Simulation Target	13-2
Language	13-6
Description	13-6
Settings	13-6
Command-Line Information	13-6
Recommended Settings	13-7
GPU acceleration	13-8
Description	13-8
Settings	13-8
Command-Line Information	13-8
Recommended Settings	13-8
Enable custom code analysis	13-9
Description	13-9
Settings	13-9
Command-Line Information	13-9
Recommended Settings	13-9
Additional code	13-10
Description	13-10
Settings	13-10
Command-Line Information	13-10
Recommended Settings	13-10
Include headers	13-11
Description	13-11
Settings	13-11

Tips	13-11
Command-Line Information	13-11
Recommended Settings	13-11
Initialize code	13-12
Description	13-12
Settings	13-12
Tip	13-12
Command-Line Information	13-12
Recommended Settings	13-12
Terminate code	13-13
Description	13-13
Settings	13-13
Tip	13-13
Command-Line Information	13-13
Recommended Settings	13-13
Include directories	13-14
Description	13-14
Settings	13-14
Command-Line Information	13-14
Recommended Settings	13-14
Source files	13-16
Description	13-16
Settings	13-16
Limitation	13-16
Tip	13-16
Command-Line Information	13-16
Recommended Settings	13-16
Libraries	13-17
Description	13-17
Settings	13-17
Limitation	13-17
Tips	13-17
Command-Line Information	13-17
Recommended Settings	13-17
Defines	13-19
Description	13-19
Settings	13-19
Command-Line Information	13-19
Recommended Settings	13-19
Compiler flags	13-20
Description	13-20
Settings	13-20
Command-Line Information	13-20
Recommended Settings	13-20
Linker flags	13-21
Description	13-21
Settings	13-21

Command-Line Information	13-21
Recommended Settings	13-21
Deterministic functions	13-22
Description	13-22
Settings	13-22
Command-Line Information	13-22
Recommended Settings	13-22
Specify by function	13-24
Description	13-24
Settings	13-24
Command-Line Information	13-24
Recommended Settings	13-24
Default function array layout	13-26
Description	13-26
Settings	13-26
Command-Line Information	13-26
Recommended Settings	13-26
Undefined function and variable handling	13-28
Description	13-28
Settings	13-28
Command-Line Information	13-28
Recommended Settings	13-29
Simulate custom code in a separate process	13-30
Description	13-30
Settings	13-30
Command-Line Information	13-30
Recommended Settings	13-30
Enable global variables as function interfaces	13-32
Description	13-32
Settings	13-32
Command-Line Information	13-32
Recommended Settings	13-32
Exception by function	13-33
Description	13-33
Settings	13-33
Command-Line Information	13-33
Example	13-33
Recommended Settings	13-34

Solver Parameters

14

Solver Pane	14-2
--------------------------	------

Start time	14-6
Description	14-6
Settings	14-6
Programmatic Use	14-6
Stop time	14-7
Description	14-7
Settings	14-7
Programmatic Use	14-7
Type	14-8
Description	14-8
Settings	14-8
Dependencies	14-8
Programmatic Use	14-9
Solver	14-10
Description	14-10
Settings	14-10
Tips	14-13
Dependencies	14-13
Command-Line Information	14-15
Max step size	14-16
Description	14-16
Settings	14-16
Tips	14-16
Dependencies	14-16
Programmatic Use	14-16
Recommended Settings	14-17
Initial step size	14-18
Description	14-18
Settings	14-18
Tips	14-18
Dependencies	14-18
Programmatic Use	14-18
Recommended Settings	14-18
Min step size	14-20
Description	14-20
Settings	14-20
Tips	14-20
Dependencies	14-20
Programmatic Use	14-20
Recommended Settings	14-20
Relative tolerance	14-22
Description	14-22
Settings	14-22
Tips	14-22
Dependencies	14-22
Programmatic Use	14-22
Recommended Settings	14-23

Absolute tolerance	14-24
Description	14-24
Settings	14-24
Tips	14-24
Dependencies	14-25
Programmatic Use	14-25
Recommended Settings	14-25
Shape preservation	14-26
Description	14-26
Settings	14-26
Tips	14-26
Dependencies	14-26
Programmatic Use	14-26
Recommended Settings	14-26
Maximum order	14-28
Description	14-28
Settings	14-28
Tips	14-28
Dependencies	14-28
Programmatic Use	14-28
Recommended Settings	14-29
Solver reset method	14-30
Description	14-30
Settings	14-30
Tips	14-30
Dependencies	14-30
Programmatic Use	14-30
Recommended Settings	14-30
Number of consecutive min steps	14-32
Description	14-32
Settings	14-32
Dependencies	14-32
Programmatic Use	14-32
Recommended Settings	14-32
Solver Jacobian Method	14-34
Description	14-34
Settings	14-34
Tips	14-34
Dependencies	14-34
Programmatic Use	14-34
Recommended Settings	14-34
Daessc mode	14-36
Description	14-36
Settings	14-36
Tips	14-36
Dependencies	14-36
Programmatic Use	14-36
Recommended Settings	14-37

Enable zero-crossing detection for fixed-step solver	14-38
Description	14-38
Settings	14-38
Dependencies	14-38
Programmatic Use	14-38
Maximum number of bracketing iterations	14-39
Description	14-39
Settings	14-39
Dependencies	14-39
Programmatic Use	14-39
Maximum number of zero-crossings per step	14-40
Description	14-40
Settings	14-40
Dependencies	14-40
Programmatic Use	14-40
Allow multiple tasks to access inputs and outputs	14-41
Description	14-41
Settings	14-41
Command-Line Information	14-41
Recommended Settings	14-41
Treat each discrete rate as a separate task	14-42
Description	14-42
Settings	14-42
Tips	14-42
Dependency	14-42
Command-Line Information	14-42
Recommended Settings	14-43
Automatically handle rate transition for data transfer	14-44
Description	14-44
Settings	14-44
Tips	14-44
Programmatic Use	14-44
Recommended Settings	14-44
Deterministic data transfer	14-46
Description	14-46
Dependencies	14-46
Programmatic Use	14-46
Recommended Settings	14-47
Higher priority value indicates higher task priority	14-48
Description	14-48
Settings	14-48
Programmatic Use	14-48
Recommended Settings	14-48
Zero-crossing control	14-49
Description	14-49
Settings	14-49
Tips	14-49

Dependencies	14-49
Programmatic Use	14-50
Recommended Settings	14-50
Time tolerance	14-51
Description	14-51
Settings	14-51
Tips	14-51
Dependencies	14-51
Programmatic Use	14-52
Recommended Settings	14-52
Number of consecutive zero crossings	14-53
Description	14-53
Settings	14-53
Tips	14-53
Dependencies	14-53
Programmatic Use	14-53
Recommended Settings	14-53
Algorithm	14-55
Description	14-55
Settings	14-55
Tips	14-55
Dependencies	14-55
Programmatic Use	14-55
Recommended Settings	14-55
Signal threshold	14-57
Description	14-57
Settings	14-57
Tips	14-57
Dependencies	14-57
Programmatic Use	14-57
Recommended Settings	14-57
Periodic sample time constraint	14-59
Description	14-59
Settings	14-59
Tips	14-59
Dependencies	14-60
Programmatic Use	14-60
Recommended Settings	14-60
Fixed-step size (fundamental sample time)	14-61
Description	14-61
Settings	14-61
Dependencies	14-61
Programmatic Use	14-61
Recommended Settings	14-62
Sample time properties	14-63
Description	14-63
Settings	14-63
Tips	14-63

Dependencies	14-64
Programmatic Use	14-64
Extrapolation order	14-65
Description	14-65
Settings	14-65
Tip	14-65
Dependencies	14-65
Programmatic Use	14-65
Recommended Settings	14-65
Number of Newton's iterations	14-67
Description	14-67
Settings	14-67
Dependencies	14-67
Programmatic Use	14-67
Recommended Settings	14-67
Allow tasks to execute concurrently on target	14-69
Description	14-69
Settings	14-69
Programmatic Use	14-69
Recommended Settings	14-70
Auto scale absolute tolerance	14-71
Description	14-71
Settings	14-71
Programmatic Use	14-71
Recommended Settings	14-71
Integration method	14-73
Description	14-73
Settings	14-73
Command-Line Information	14-73
Recommended Settings	14-73

Hardware Implementation Parameters

15

Hardware Implementation Pane	15-2
Hardware board	15-5
Settings	15-5
Tips	15-5
Dependencies	15-5
Command-Line Information	15-5
Recommended Settings	15-6
See Also	15-6
Code Generation system target file	15-7

Device vendor	15-8
Settings	15-8
Tips	15-8
Dependencies	15-8
Command-Line Information	15-9
Recommended Settings	15-9
See Also	15-9
Device type	15-10
Settings	15-10
Tips	15-12
Dependencies	15-19
Command-Line Information	15-20
Recommended Settings	15-20
See Also	15-20
Number of bits: char	15-21
Description	15-21
Settings	15-21
Tip	15-21
Dependencies	15-21
Command-Line Information	15-21
Recommended Settings	15-21
See Also	15-22
Number of bits: short	15-23
Description	15-23
Settings	15-23
Tip	15-23
Dependencies	15-23
Command-Line Information	15-23
Recommended Settings	15-23
See Also	15-24
Number of bits: int	15-25
Description	15-25
Settings	15-25
Tip	15-25
Dependencies	15-25
Command-Line Information	15-25
Recommended Settings	15-25
See Also	15-26
Number of bits: long	15-27
Description	15-27
Settings	15-27
Tip	15-27
Dependencies	15-27
Command-Line Information	15-27
Recommended Settings	15-27
See Also	15-28
Number of bits: long long	15-29
Description	15-29
Settings	15-29

Tips	15-29
Dependencies	15-29
Command-Line Information	15-29
Recommended Settings	15-29
See Also	15-30
Number of bits: float	15-31
Description	15-31
Settings	15-31
Command-Line Information	15-31
Recommended Settings	15-31
See Also	15-31
Number of bits: double	15-32
Description	15-32
Settings	15-32
Command-Line Information	15-32
Recommended Settings	15-32
See Also	15-32
Number of bits: native	15-33
Description	15-33
Settings	15-33
Tip	15-33
Dependencies	15-33
Command-Line Information	15-33
Recommended Settings	15-33
See Also	15-34
Number of bits: pointer	15-35
Description	15-35
Settings	15-35
Dependencies	15-35
Command-Line Information	15-35
Recommended Settings	15-35
See Also	15-35
Number of bits: size_t	15-37
Description	15-37
Settings	15-37
Dependencies	15-37
Command-Line Information	15-37
Recommended Settings	15-37
See Also	15-38
Number of bits: ptrdiff_t	15-39
Description	15-39
Settings	15-39
Dependencies	15-39
Command-Line Information	15-39
Recommended Settings	15-39
See Also	15-40
Largest atomic size: integer	15-41
Description	15-41

Settings	15-41
Tip	15-41
Dependencies	15-41
Command-Line Information	15-41
Recommended Settings	15-42
See Also	15-42
Largest atomic size: floating-point	15-43
Description	15-43
Settings	15-43
Tip	15-43
Dependencies	15-43
Command-Line Information	15-43
Recommended Settings	15-43
See Also	15-44
Byte ordering	15-45
Description	15-45
Settings	15-45
Dependencies	15-45
Command-Line Information	15-45
Recommended Settings	15-45
See Also	15-46
Signed integer division rounds to	15-47
Description	15-47
Settings	15-47
Tips	15-47
Dependency	15-48
Command-Line Information	15-48
Recommended settings	15-48
See Also	15-48
Shift right on a signed integer as arithmetic shift	15-49
Description	15-49
Settings	15-49
Tips	15-49
Dependency	15-49
Command-Line Information	15-49
Recommended settings	15-49
See Also	15-50
Support long long	15-51
Description	15-51
Settings	15-51
Tips	15-51
Dependencies	15-51
Command-Line Information	15-51
Recommended Settings	15-51
See Also	15-52

Signal Properties Dialog Box

16

Data Transfer Options for Concurrent Execution	16-2
Specify data transfer settings	16-2
Data transfer handling option	16-2
Extrapolation method (continuous-time signals)	16-2
Initial condition	16-2

Simulink Preferences Window

17

Font Styles for Models	17-2
Font Styles Overview	17-2

Simulink Mask Editor

18

Mask Editor Overview	18-2
Parameters & Dialog Pane	18-2
Code Pane	18-16
Icon Pane	18-19
Constraints	18-27
Additional Options	18-28
Dialog Control Operations	18-30
Moving dialog controls in the Dialog box	18-30
Cut, Copy, and Paste Controls	18-30
Delete nodes	18-31
Error Display	18-31
Specify Data Types Using DataTypeStr Parameter	18-33
Associate Data Types to Edit Parameter	18-33
View DataTypeStr Programmatically	18-37
Design a Mask Dialog Box	18-39

Concurrent Execution Window

19

Concurrent Execution Window: Main Pane	19-2
Concurrent Execution Window Overview	19-2
Enable explicit model partitioning for concurrent behavior	19-3

Data Transfer Pane	19-5
Data Transfer Pane Overview	19-5
Periodic signals	19-5
Continuous signals	19-6
Extrapolation method	19-6
Automatically handle rate transition for data transfer	19-7
CPU Pane	19-8
CPU Pane Overview	19-8
Name	19-8
Hardware Node Pane	19-9
Hardware Node Pane Overview	19-9
Name	19-9
Clock Frequency [MHz]	19-9
Color	19-9
Periodic Pane	19-11
Periodic Pane Overview	19-11
Name	19-11
Periodic Trigger	19-11
Color	19-12
Template	19-12
Task Pane	19-13
Task Pane Overview	19-13
Name	19-13
Period	19-14
Color	19-14
Interrupt Pane	19-15
Interrupt Pane Overview	19-15
Name	19-15
Color	19-16
Aperiodic trigger source	19-16
Signal number [2,SIGRTMAX-SIGRTMIN-1]	19-17
Event name	19-17
System Tasks Pane	19-19
System Tasks Pane Overview	19-19
System Task Pane	19-20
System Task Pane Overview	19-20
Name	19-20
Period	19-20
Color	19-21
System Interrupt Pane	19-22
System Interrupt Pane Overview	19-22
Name	19-22
Color	19-22
Profile Report Pane	19-24
Profile Report Pane Overview	19-24
Number of time steps	19-24

20

Simulation Stepping Options	20-2
Simulation Stepping Options Overview	20-2
Enable stepping back	20-3
Maximum number of saved back steps	20-3
Interval between stored back steps	20-4
Move back/forward by	20-4

Variant Manager for Simulink

21

Variant Manager for Simulink	21-2
Variant Manager	21-2
Install Variant Manager for Simulink	21-3
Open Variant Manager	21-4
Explore Variant Manager Window	21-4
Manage Variant Elements	21-5
Reduce a Variant Model	21-9
Analyze Variant Configurations	21-9
Icons in Variant Manager	21-10
Access the Variant Manager Functionality Programmatically	21-13
Limitations	21-14

Math and Data Types

22

Math and Data Types Pane	22-2
Simulation behavior for denormal numbers	22-3
Description	22-3
Settings	22-3
Tips	22-3
Command-Line Information	22-3
Dependency	22-3
Default for underspecified data type	22-4
Description	22-4
Settings	22-4
Tips	22-4
Command-Line Information	22-4
Recommended Settings	22-4
Use division for fixed-point net slope computation	22-6
Description	22-6
Settings	22-6
Tips	22-6

Dependency	22-7
Command-Line Information	22-7
Recommended Settings	22-7
Gain parameters inherit a built-in integer type that is lossless	22-8
Description	22-8
Settings	22-8
Tips	22-8
Dependencies	22-8
Command-Line Information	22-8
Recommended Settings	22-9
Use floating-point multiplication to handle net slope corrections	22-10
Description	22-10
Settings	22-10
Tips	22-10
Dependencies	22-10
Command-Line Information	22-10
Recommended Settings	22-10
Inherit floating-point output type smaller than single precision	22-12
Description	22-12
Settings	22-12
Tips	22-13
Dependencies	22-13
Command-Line Information	22-13
Recommended Settings	22-13
Application lifespan (days)	22-14
Description	22-14
Settings	22-14
Tips	22-14
Command-Line Information	22-15
Recommended Settings	22-15
Use algorithms optimized for row-major array layout	22-16
Description	22-16
Settings	22-16
Tips	22-17
Command-Line Information	22-17
Recommended Settings	22-17

Model Parameter Configuration Dialog Box

23

Model Parameter Configuration Dialog Box	23-2
Source list	23-2
Refresh list	23-3
Add to table	23-3
New	23-3
Storage class	23-3
Storage type qualifier	23-3

Model Configuration Parameters: Model Advisor	24-2
Model Advisor Pane Overview	24-2
To get help on an option	24-2
Model Advisor configuration file	24-3
Description	24-3
Settings	24-3
Tips	24-3
Command-Line Information	24-3
Recommended Settings	24-4
Show Model Advisor edit-time checks	24-5
Description	24-5
Settings	24-5
Tip	24-5
Command-Line Information	24-5
Recommended Settings	24-6

Configuration Parameters Dialog Box

Model Configuration Pane

In this section...
“Model Configuration Overview” on page 1-2
“Name” on page 1-2
“Description” on page 1-2
“Configuration Parameters” on page 1-3

Model Configuration Overview

View or edit the name and description of your configuration set.

In the Model Explorer you can edit the name and description of your configuration sets.

In the Model Explorer or Simulink Preferences window you can edit the description of your template configuration set, Model Configuration Preferences. Go to the Model Configuration Preferences to edit the template Configuration Parameters to be used as defaults for new models.

When editing the Model Configuration preferences, you can click **Restore to Default Preferences** to restore the default configuration settings for creating new models. These underlying defaults cannot be changed.

For more information about configuration sets, see “Manage Configuration Sets for a Model”.

Name

Specify the name of your configuration set.

Settings

Default: Configuration (for Active configuration set) or Configuration Preferences (for default configuration set).

Edit the name of your configuration set.

In the Model Configuration Preferences, the name of the default configuration is always Configuration Preferences, and cannot be changed.

Description

Specify a description of your configuration set.

Settings

No Default

Enter text to describe your configuration set.

Configuration Parameters

No further help documentation is available for this parameter.

Simulink Configuration Parameters: Advanced

Test hardware is the same as production hardware

Description

Specify whether the test hardware differs from the production hardware.

Category: Hardware Implementation

Settings

Default: On

On

Specifies that the hardware used to test the code generated from the model is the same as the production hardware, or has the same characteristics.

Off

Specifies that the hardware used to test the code generated from the model has different characteristics than the production hardware.

Tip

You can generate code that runs on the test hardware but behaves as if it had been generated for and executed on the deployment hardware.

Dependency

Enables test hardware parameters.

Recommended settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	On

See Also

Related Examples

- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)
- “Hardware Implementation Pane” on page 15-2

Test device vendor and type

Description

Select the manufacturer and type of the hardware to use to test the code generated from the model.

Category: Hardware Implementation

Settings

Default: Intel, x86-64 (Windows64)

- AMD
- ARM Compatible
- Altera
- Analog Devices
- Apple
- Atmel
- Freescale
- Infineon
- Intel
- Microchip
- NXP
- Renesas
- STMicroelectronics
- Texas Instruments
- ASIC/FPGA
- Custom Processor

AMD® options:

- Athlon 64
- K5/K6/Athlon
- x86-32 (Windows 32)
- x86-64 (Linux 64)
- x86-64 (macOS)
- x86-64 (Windows64)

ARM® options:

- ARM 10
- ARM 11
- ARM 7

- ARM 8
- ARM 9
- ARM Cortex-A
- ARM Cortex-M
- ARM Cortex-R
- ARM Cortex
- ARM 64-bit (LP64)
- ARM 64-bit (LLP64)

Altera® options:

- SoC (ARM CortexA)

Analog Devices® options:

- ADSP-CM40x (ARM Cortex-M)
- Blackfin
- SHARC
- TigerSHARC

Apple options:

- ARM64

Atmel® options:

- AVR
- AVR (32-bit)
- AVR (8-bit)

Freescale™ options:

- 32-bit PowerPC
- 68332
- 68HC08
- 68HC11
- ColdFire
- DSP563xx (16-bit mode)
- HC(S)12
- MPC52xx
- MPC5500
- MPC55xx
- MPC5xx
- MPC7xxx
- MPC82xx

- MPC83xx
- MPC85xx
- MPC86xx
- MPC8xx
- S08
- S12x
- StarCore

Infineon® options:

- C16x, XC16x
- TriCore

Intel® options:

- x86–32 (Windows32)
- x86–64 (Linux 64)
- x86–64 (macOS)
- x86–64 (Windows64)

Microchip options:

- PIC18
- dsPIC

NXP options:

- Cortex–M0/M0+
- Cortex–M3
- Cortex–M4

Renesas® options:

- M16C
- M32C
- R8C/Tiny
- RH850
- RL78
- RX
- RZ
- SH-2/3/4
- V850

STMicroelectronics®:

- ST10/Super10

Texas Instruments™ options:

- C2000
- C5000
- C6000
- MSP430
- Stellaris Cortex-M3
- TMS470
- TMS570 Cortex-R4

ASIC/FPGA options:

- ASIC/FPGA

Tips

- Before you select the device type, select the device vendor.
- Selecting a device type specifies the hardware device to define system constraints:
 - Default hardware properties appear as the initial values.
 - You cannot change parameters with only one possible value.
 - Parameters with more than one possible value provide a list of valid values.

The following table lists values for each device type.

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
AMD															
Athlon 64	8	16	32	64	64	64	64	64	64	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
K5/K6/Athlon	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
x86-32 (Windows32)	8	16	32	32	64	32	32	32	32	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
x86-64 (Linux 64)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
x86-64 (macOS)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
x86-64 (Windows64)	8	16	32	32	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
ARM Compatible															
ARM 7/8/9/10	8	16	32	32	64	32	32	32	32	Long	Float	Little Endian	Zero	✓	<input type="checkbox"/>
ARM 11	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
ARM Cortex	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
ARM 64-bit (LP64)	8	16	32	64	64	64	64	64	64	Long	Double	Little Endian	Zero	✓	✓
ARM 64-bit (LLP64)	8	16	32	32	64	64	64	64	64	Long	Double	Little Endian	Zero	✓	✓
Altera															
SoC (ARM Cortex A)	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
Analog Devices															
ADSP-CM40x (ARM Cortex-M)	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
Blackfin	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
SHARC	32	32	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	<input type="checkbox"/>
TigerSHARC	32	32	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
Apple															
ARM64	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
Atmel															
AVR	8	16	16	32	64	8	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
AVR (32-bit)	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
AVR (8-bit)	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
Freescale															
32-bit PowerPC	8	16	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	<input type="checkbox"/>
68332	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>
68HC08	8	16	16	32	64	8	8	16	8	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
68HC11	8	16	16	32	64	8	8	16	16	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>
ColdFire	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>
DSP563xx (16-bit mode)	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
DSP5685x	8	16	16	32	64	16	16	16	16	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
HC(S)12	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>
MPC52xx, MPC5500, MPC55xx, MPC5xx, PC5xx, MPC7xxx, MPC82xx, MPC83xx, MPC86xx, MPC8xx	8	16	32	32	64	32	32	32	32	Long	None	Big Endian	Zero	✓	<input type="checkbox"/>
MPC85xx	8	16	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	<input type="checkbox"/>
S08	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>
S12x	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
StarCore	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
Infineon															
C16x, XC16x	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
TriCore	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
Intel															
x86-32 (Windows32)	8	16	32	32	64	32	32	32	32	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
x86-64 (Linux 64)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
x86-64 (macOS)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
x86-64 (Windows64)	8	16	32	32	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
Microchip															
PIC18	8	16	16	32	64	8	8	24	24	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
dsPIC	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
NXP															

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
Cortex-M0/M0+	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
Cortex-M3	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
Cortex-M4	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
Renesas															
M16C	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
M32C	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
R8C/Tiny	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
RH850	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
RL78	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
RX	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
RZ	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
SH-2/3/4	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
V850	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
STMicroelectronics															
ST10/Super10	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
Texas Instruments															
C2000	16	16	16	32	64	16	32	16	16	Int	None	Little Endian	Zero	✓	<input type="checkbox"/>
C5000	16	16	16	32	64	16	16	16	16	Int	None	Big Endian	Zero	✓	<input type="checkbox"/>
C6000	8	16	32	40	64	32	32	32	32	Int	None	Little Endian	Zero	✓	<input type="checkbox"/>
MSP430	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
Stellaris Cortex-M3	8	16	32	32	6	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
TMS470	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
TMS570 Cortex-R4	8	16	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	<input type="checkbox"/>
ASIC/FPGA															
ASIC/FPGA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

- If your hardware does not match one of the listed types, select Custom.

- The **Device vendor** and **Device type** fields share the command-line parameter `TargetHWDeviceType`. When specifying this parameter at the command line, separate the device vendor and device type values by using the characters `->`. For example: `'Intel->x86-64 (Linux 64)'`.
- If you have a Simulink Coder™ license and you want to add **Device vendor** and **Device type** values to the default set, see “Register New Hardware Devices” (Simulink Coder).

Dependencies

The **Device vendor** and **Device type** parameter values reflect available device support for the selected hardware board.

Menu options that are available depend on the **Device vendor** parameter setting.

With the exception of device vendor ASIC/FPGA, selecting a device type sets the following parameters:

- **Number of bits: char**
- **Number of bits: short**
- **Number of bits: int**
- **Number of bits: long**
- **Number of bits: long long**
- **Number of bits: float**
- **Number of bits: double**
- **Number of bits: native**
- **Number of bits: pointer**
- **Number of bits: size_t**
- **Number of bits: ptrdiff_t**
- **Largest atomic size: integer**
- **Largest atomic size: floating-point**
- **Byte ordering**
- **Signed integer division rounds to**
- **Shift right on a signed integer as arithmetic shift**
- **Support long long**

Whether you can modify the value of a device-specific parameter varies according to device type.

Command-Line Information

Parameter: `TargetHWDeviceType`

Type: character vector

Value: any valid value (see tips)

Default: `'Intel->x86-64 (Windows64)'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- “Hardware board” on page 15-5
- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)
- “Hardware Implementation Pane” on page 15-2

Number of bits: char

Description

Describe the character bit length for the hardware that you use to test code.

Category: Hardware Implementation

Settings

Default: 8

Minimum: 8

Maximum: 32

Enter an integer value between 8 and 32.

Tip

All values must be a multiple of 8.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: TargetBitPerChar

Type: integer

Value: any valid value

Default: 8

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- [Specifying Test Hardware Characteristics \(Simulink Coder\)](#)
- [Hardware Implementation Options \(Simulink Coder\)](#)
- [“Hardware Implementation Pane” on page 15-2](#)

Number of bits: short

Description

Describe the data bit length for the hardware that you use to test code.

Category: Hardware Implementation

Settings

Default: 16

Minimum: 8

Maximum: 32

Enter an integer value between 8 and 32.

Tip

All values must be a multiple of 8.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: TargetBitPerShort

Type: integer

Value: any valid value

Default: 16

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- [Specifying Test Hardware Characteristics \(Simulink Coder\)](#)
- [Hardware Implementation Options \(Simulink Coder\)](#)
- [“Hardware Implementation Pane” on page 15-2](#)

Number of bits: int

Description

Describe the data integer bit length of the hardware that you use to test code.

Category: Hardware Implementation

Settings

Default: 32

Minimum: 8

Maximum: 32

Enter an integer value between 8 and 32.

Tip

All values must be a multiple of 8.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: TargetBitPerInt

Type: integer

Value: any valid value

Default: 32

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- [Specifying Test Hardware Characteristics \(Simulink Coder\)](#)
- [Hardware Implementation Options \(Simulink Coder\)](#)
- [“Hardware Implementation Pane” on page 15-2](#)

Number of bits: long

Description

Describe the data bit lengths for the hardware that you use to test code.

Category: Hardware Implementation

Settings

Default: 32

Minimum: 32

Maximum: 64

Enter an integer value between 32 and 64.

Tip

All values must be a multiple of 8 and between 32 and 64.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: TargetBitPerLong

Type: integer

Value: any valid value

Default: 32

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- [Specifying Test Hardware Characteristics \(Simulink Coder\)](#)
- [Hardware Implementation Options \(Simulink Coder\)](#)
- [“Hardware Implementation Pane” on page 15-2](#)

Number of bits: long long

Description

Describe the length in bits of the C `long long` data type that the test hardware supports.

Category: Hardware Implementation

Settings

Default: 64

Minimum: 64

Maximum: 128

The number of bits that represent the C `long long` data type.

Tips

- Use the `long long` data type only if your C compiler supports `long long`.
- You can change the value for custom targets only. For custom targets, all values must be a multiple of 8 and between 64 and 128.

Dependencies

- **Enable long long** enables use of this parameter.
- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- The value of this parameter must be greater than or equal to the value of **Number of bits: long**.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: TargetBitPerLongLong

Type: integer

Value: any valid value

Default: 64

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific

Application	Setting
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)
- “Hardware Implementation Pane” on page 15-2

Number of bits: float

Description

Describe the bit length of floating-point data for the hardware that you use to test code (read only).

Category: Hardware Implementation

Settings

Default: 32

Always equals 32.

Command-Line Information

Parameter: TargetBitPerFloat

Type: integer

Value: 32 (read-only)

Default: 32

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)
- “Hardware Implementation Pane” on page 15-2

Number of bits: double

Description

Describe the bit-length of double data for the hardware that you use to test code (read only).

Category: Hardware Implementation

Settings

Default: 64

Always equals 64.

Command-Line Information

Parameter: TargetBitPerDouble

Type: integer

Value: 64 (read only)

Default: 64

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)
- “Hardware Implementation Pane” on page 15-2

Number of bits: native

Description

Describe the microprocessor native word size for the hardware that you use to test code.

Category: Hardware Implementation

Settings

Default: 64

Minimum: 8

Maximum: 64

Enter a value between 8 and 64.

Tip

All values must be a multiple of 8.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: TargetWordSize

Type: integer

Value: any valid value

Default: 64

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- [Specifying Test Hardware Characteristics \(Simulink Coder\)](#)
- [Hardware Implementation Options \(Simulink Coder\)](#)
- [“Hardware Implementation Pane” on page 15-2](#)

Number of bits: pointer

Description

Describe the bit-length of pointer data for the hardware that you use to test code.

Category: Hardware Implementation

Settings

Default: 64

Minimum: 8

Maximum: 64

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: TargetBitPerPointer

Type: integer

Value: any valid value

Default: 64

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)
- “Hardware Implementation Pane” on page 15-2

Number of bits: size_t

Description

Describe the bit-length of `size_t` data for the hardware that you use to test code.

If `ProdEqTarget` is off, an Embedded Coder® processor-in-the-loop (PIL) simulation checks this setting with reference to the target hardware. If `ProdEqTarget` is on, the PIL simulation checks the `ProdBitPerSizeT` setting.

Category: Hardware Implementation

Settings

Default: 64

Value must be 8, 16, 24, 32, 40, 64, or 128 *and* greater or equal to the value of `int`.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: TargetBitPerSizeT

Type: integer

Value: any valid value

Default: 64

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)

- “Hardware Implementation Pane” on page 15-2
- “Verification of Code Generation Assumptions” (Embedded Coder)

Number of bits: ptrdiff_t

Description

Describe the bit-length of ptrdiff_t data for the hardware that you use to test code.

If ProdEqTarget is off, an Embedded Coder processor-in-the-loop (PIL) simulation checks this setting with reference to the target hardware. If ProdEqTarget is on, the PIL simulation checks the ProdBitPerPtrDiffT setting.

Category: Hardware Implementation

Settings

Default: 64

Value must be 8, 16, 24, 32, 40, 64, or 128 *and* greater or equal to the value of int.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: TargetBitPerPtrDiffT

Type: integer

Value: any valid value

Default: 64

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)

- “Hardware Implementation Pane” on page 15-2
- “Verification of Code Generation Assumptions” (Embedded Coder)

Largest atomic size: integer

Description

Specify the largest integer data type that can be atomically loaded and stored on the hardware that you use to test code.

Category: Hardware Implementation

Settings

Default: Char

Char

Specifies that `char` is the largest integer data type that can be atomically loaded and stored on the hardware that you use to test code.

Short

Specifies that `short` is the largest integer data type that can be atomically loaded and stored on the hardware that you use to test code.

Int

Specifies that `int` is the largest integer data type that can be atomically loaded and stored on the hardware that you use to test code.

Long

Specifies that `long` is the largest integer data type that can be atomically loaded and stored on the hardware that you use to test code.

LongLong

Specifies that `long long` is the largest integer data type that can be atomically loaded and stored on the hardware that you use to test code.

Tip

Use this parameter, where possible, to remove unnecessary double-buffering or unnecessary semaphore protection, based on data size, in generated multirate code.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.
- You can set this parameter to `LongLong` only if the hardware used to test the code supports the C long long data type and you have selected **Enable long long**.

Command-Line Information

Parameter: TargetLargestAtomicInteger

Value: 'Char' | 'Short' | 'Int' | 'Long' | 'LongLong'

Default: 'Char'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)
- “Support long long” on page 2-44
- “Hardware Implementation Pane” on page 15-2

Largest atomic size: floating-point

Description

Specify the largest floating-point data type that can be atomically loaded and stored on the hardware that you use to test code.

Category: Hardware Implementation

Settings

Default: Float

Float

Specifies that `float` is the largest floating-point data type that can be atomically loaded and stored on the hardware that you use to test code.

Double

Specifies that `double` is the largest floating-point data type that can be atomically loaded and stored on the hardware that you use to test code.

None

Specifies that there is no applicable setting or not to use this parameter in generating multirate code.

Tip

Use this parameter, where possible, to remove unnecessary double-buffering or unnecessary semaphore protection, based on data size, in generated multirate code.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: TargetLargestAtomicFloat

Value: 'Float' | 'Double' | 'None'

Default: 'Float'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific

Application	Setting
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)
- “Hardware Implementation Pane” on page 15-2

Byte ordering

Description

Describe the byte ordering for the hardware that you use to test code.

Category: Hardware Implementation

Settings

Default: Little Endian

Unspecified

Specifies that the code determines the endianness of the hardware. This choice is the least efficient.

Big Endian

The most significant byte comes first.

Little Endian

The least significant byte comes first.

Note For guidelines about configuring **Production hardware** controls for code generation, see Hardware Implementation Options (Simulink Coder).

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: TargetEndianness

Value: 'Unspecified' | 'LittleEndian' | 'BigEndian'

Default: 'LittleEndian'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)
- “Hardware Implementation Pane” on page 15-2

Signed integer division rounds to

Description

Describe how your compiler for the test hardware rounds the result of dividing two signed integers.

Category: Hardware Implementation

Settings

Default: Zero

Undefined

Choose this option if neither Zero nor Floor describes the compiler behavior, or if that behavior is unknown.

Zero

If the quotient is between two integers, the compiler chooses the integer that is closer to zero as the result.

Floor

If the quotient is between two integers, the compiler chooses the integer that is closer to negative infinity.

Tips

- Use the **Integer rounding mode** parameter on your model's blocks to simulate the rounding behavior of the C compiler that you use to compile code generated from the model. This setting appears on the **Signal Attributes** pane of the parameter dialog boxes of blocks that can perform signed integer arithmetic, such as the Product block.
- For most blocks, the value of **Integer rounding mode** completely defines rounding behavior. For blocks that support fixed-point data and the **Simplest** rounding mode, the value of **Signed integer division rounds to** also affects rounding. For details, see "Rounding" (Fixed-Point Designer).
- For information on how this option affects code generation, see Hardware Implementation Options (Simulink Coder).
- This table illustrates the compiler behavior described by the options for this parameter.

N	D	Ideal N/D	Zero	Floor	Undefined
33	4	8.25	8	8	8
-33	4	-8.25	-8	-9	-8 or -9
33	-4	-8.25	-8	-9	-8 or -9
-33	-4	8.25	8	8	8 or 9

Dependency

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: TargetIntDivRoundTo

Value: 'Floor' | 'Zero' | 'Undefined'

Default: 'Zero'

Recommended settings

Application	Setting
Debugging	No impact for simulation or during development. Undefined for production code generation.
Traceability	No impact for simulation or during development. Zero or Floor for production code generation.
Efficiency	No impact for simulation or during development. Zero for production code generation.
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)
- “Hardware Implementation Pane” on page 15-2

Shift right on a signed integer as arithmetic shift

Description

Describe how your compiler for the test hardware fills the sign bit in a right shift of a signed integer.

Category: Hardware Implementation

Settings

Default: On

On

Generates simple, efficient code whenever the Simulink model performs arithmetic shifts on signed integers.

Off

Generates fully portable but less efficient code to implement right arithmetic shifts.

Tips

- Select this parameter if your C compiler implements a signed integer right shift as an arithmetic right shift.
- An arithmetic right shift fills bits vacated by the right shift with the value of the most significant bit, which indicates the sign of the number in twos complement notation. It is equivalent to dividing the number by 2.
- This setting affects only code generation.

Dependency

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: TargetShiftRightIntArith

Value: 'on' | 'off'

Default: 'on'

Recommended settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	On

Application	Setting
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)
- “Hardware Implementation Pane” on page 15-2

Support long long

Description

Specify that your C compiler supports the C `long long` data type. Most C99 compilers support `long long`.

Category: Hardware Implementation

Settings

Default: Off

On

Enables use of C `long long` data type on the test hardware.

Off

Disables use of C `long long` data type on the test hardware.

Tips

- This parameter is enabled only if the selected test hardware supports the C `long long` data type.
- If your compiler does not support C `long long`, do not select this parameter.

Dependencies

This parameter enables **Number of bits: long long**.

Command-Line Information

Parameter: TargetLongLongMode

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific
Safety precaution	No impact when Test hardware is the same as production hardware is selected. If it is not selected, no recommendation.

See Also

Related Examples

- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)
- “Number of bits: long long” on page 2-23
- “Hardware Implementation Pane” on page 15-2

Allowed unit systems

Description

Specify unit systems allowed in the model.

Category: Diagnostics

Settings

Default: all

all or comma-separated list of one or more of:

SI

International System of Units.

SI (extended)

International System of Units (extended).

English

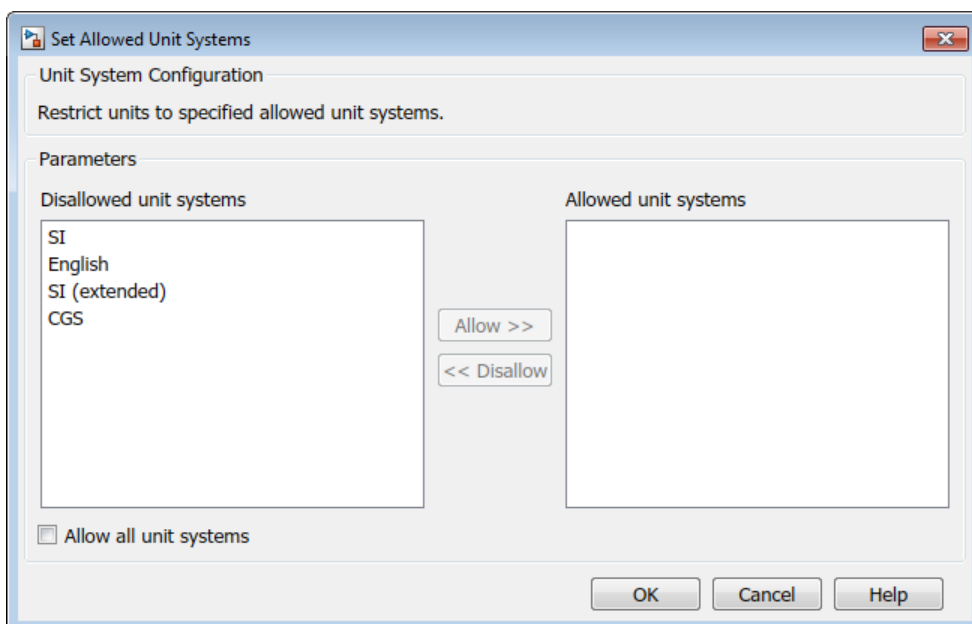
English units.

CGS

Centimetre-gram-second system of units.

Tip

As an alternative to the text box, click the **Set Allowed Unit Systems** button.



- To allow all unit systems, select the **Allow all unit systems** check box.
- Use the **Allow** and **Disallow** buttons to allow or disallow selected unit systems.

Command-Line Information

Parameter: AllowedUnitSystems

Type: character vector

Value: SI | SI (extended) | English | CGS in a comma-delimited list or all, without quotation marks

Default: all

See Also

Related Examples

- “Unit Specification in Simulink Models”
- Solver Diagnostics on page 9-2

Units inconsistency messages

Description

Specify if unit inconsistencies should be reported as warnings. Select the diagnostic action to take when the Simulink software detects unit inconsistencies.

Category: Diagnostics

Settings

Default: warning

warning

Display unit inconsistencies as warnings.

none

Display nothing for unit inconsistencies (do not report unit inconsistencies),

Command-Line Information

Parameter: UnitsInconsistencyMsg

Value: 'warning' | 'none'

Default: 'warning'

See Also

Related Examples

- “Unit Specification in Simulink Models”
- Solver Diagnostics on page 9-2

Allow automatic unit conversions

Description

Allow automatic unit conversions in the model.

Category: Diagnostics

Settings

Default: On

On

Enables automatic unit conversions in cases where units have a known mathematical relationship. For more information, see “Converting Units”.

Off

Disables automatic unit conversions in cases where units where units have a known mathematical relationship. To convert, you must insert a Unit Conversion block between the differing ports.

Command-Line Information

Parameter: AllowAutomaticUnitConversions

Value: 'on' | 'off'

Default: 'on'

See Also

Related Examples

- “Unit Specification in Simulink Models”
- “Converting Units”
- Solver Diagnostics on page 9-2

Dataset signal format

Description

Format for logged Dataset leaf elements.

Category: Data Import/Export

Settings

Default: timeseries

timeseries

Save Dataset element values in MATLAB® timeseries format.

timetable

Save Dataset element values in MATLAB timetable format.

Comparison of Formats

The timetable format enables easier merging of logged data from multiple simulations.

Property Display

The timeseries format displays one field for time properties (TimeInfo) and a second field for data properties (DataInfo). For example, here are the properties of a timeseries object for a nonscalar signal.

```
ts
```

```
timeseries
```

```
Common Properties:
  Name: ''
  Time: [1001x1 double]
  TimeInfo: [1x1 tsdata.timemetadata]
  Data: [1001x1 double]
  DataInfo: [1x1 tsdata.datametadata]
```

When you enter the name of a timetable object (for example, tt) and query the properties, you see all of the properties.

```
tt.Properties
```

```
ans =
```

```
struct with fields:
  Description: ''
  UserData: []
  DimensionNames: {'Time' 'Variables'}
  VariableDescriptions: {}
```

```

    VariableNames: ['temperature' 'WindSpeed' 'WindDirection']
    VariableUnits: {}
    VariableContinuity: ['continuous']
    RowTimes: [64x1 duration]

```

Data Access

To access data logged in the `timeseries` format, use the `Data` property for a signal. For example, for a `timeseries` object `ts` (only first five values shown):

```

ts = yout{1}.Values;
ts.Data

ans =

    0
 -0.0002
 -0.0012
 -0.0062
 -0.0306

```

The `timetable` format for logged Dataset data produces a table with one time column, called `Time`, and one data column, called `Data`. The `Time` column is the simulation time vector for a given signal, stored as a duration type, with the setting of seconds to match the units of simulation time, starting with the simulation start time (typically set to 0 sec). The Simulink signal dimensions of `[n]` and `[nx1]` are treated equivalently in the `timetable` representation. For example, for a `timetable` object `tt` (only first five values shown):

```

tt = yout{1}.Values;
tt.Data

```

Time	Data
0 sec	[1x3x2 double]
0.1 sec	[1x3x2 double]
0.2 sec	[1x3x2 double]
0.3 sec	[1x3x2 double]
0.4 sec	[1x3x2 double]

The number of samples is the first dimension in the `Data` column of the `timetable` object, but it is the last dimension in the data field of logged `timeseries` data that is nonscalar. Therefore, when you access data in `timetable` format, you may need to reshape the data when each sample is a nonscalar array. One option is to use the `squeeze` function. For example, to access the first data row in the dataset, you can use a command like this:

```

squeeze(tt.Data{1,1})

ans =
     1     2
     3     4
     5     6

```

If a signal is a bus or array of buses, the signal values are logged as a structure of `timetable` objects, with each leaf of the structure corresponding to the logged result of each leaf signal in the bus.

Units

For data logged in Simulink, the `timeseries` format displays units for time values in the `Units` property. Units can be specified as any value of any class. Timeseries logging sets the units to a `Simulink.SimulationData.Unit` object, if the logged signal has units specified. For loading, units are honored only if they are of type `Simulink.SimulationData.Unit`; otherwise, they are ignored.

For the `timetable` format, Simulink does not support units for logged data.

Data Interpolation

The `timeseries` format `Interpolation` property displays whether the interpolation method is `linear` (default) or `zoh`.

The `timetable` format `VariableContinuity` property characterizes variables as continuous or discrete. The possible values for simulation data are:

- `continuous` - Corresponds to the `timeseries` property `Interpolation` setting of `linear`. Simulink uses this setting for filling continuous sample times.
- `step` - Corresponds to the `timeseries` property `Interpolation` setting of `zoh`.

Simulink uses this setting for filling discrete sample times.

Uniform and Nonuniform Time

The `timeseries` format displays whether the time data is uniform or nonuniform. For data logged for continuous sample times (linear interpolation), the `TimeInfo` property indicates that the time is nonuniform and gives the length. For a discrete sample times (zero-order hold interpolation), the `TimeInfo` property indicates that the time is uniform and gives the length and increment.

The `timetable` format does not have a property for uniform and nonuniform time data.

For data in `timeseries` or `timetable` format, you can use the MATLAB `isregular` function to get this time information.

Signal Name

The `timeseries` format stores the name of a logged signal in a `Simulink.SimulationData.Element` wrapper object, as well as in the `timeseries` object itself.

The `timetable` format stores the name of a logged signal in a `Simulink.SimulationData.Element` wrapper object, but not in the `timetable` object itself.

Tips

- The **Dataset signal format** parameter has no effect when using Scope blocks to log data.

Programmatic Use

Parameter: `DatasetSignalFormat`

Value: `'timeseries' | 'timetable'`

Default: `'timeseries'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Model Configuration Parameters: Data Import/Export” on page 3-2
- “Log Data to Persistent Storage”

Stream To Workspace blocks

Description

Whether data logged using To Workspace blocks streams to the Simulation Data Inspector.

Category: Data Import/Export

Settings

Default: On

On

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

Off

Data logged using To Workspace blocks does not stream to the Simulation Data Inspector.

When you disable this parameter, To Workspace blocks do not support:

- Logging arrays of buses.
- Logging signals with `string` or `half` data types.
- Logging signals with `int64` or `uint64` using the built-in data types.

When a license for Fixed-Point Designer™ is available, `int64` and `uint64` data is logged as a `fi` object. When a license for Fixed-Point Designer is not available, data is logged as `double` data.

- Logging data inside for-each subsystems.
- Using the `Timeseries` format in rapid accelerator simulations.

Tips

When you run multiple simulations in a single MATLAB session, the Simulation Data Inspector retains results from each simulation so you can analyze the results together. To control the amount of data retained in the Simulation Data Inspector, do not use this parameter. See “Limit the Size of Logged Data”.

Programmatic Use

Parameter: `StreamToWks`

Value: `'on'` | `'off'`

Default: `'on'`

Recommended Settings

Application	Setting
Debugging	On
Traceability	No recommendation
Efficiency	On
Safety precaution	No recommendation

See Also

Related Examples

- “Model Configuration Parameters: Data Import/Export” on page 3-2
- “Limit the Size of Logged Data”

Array bounds exceeded

Description

Ensure that Simulink-allocated memory used in S-functions does not write beyond its assigned array bounds when writing to its outputs, states, or work vectors.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- Use this option to check whether execution of each instance of a block during model simulation writes data to memory locations not allocated to the block. This can happen only if your model includes a user-written S-function that has a bug.
- Enabling this option slows down model execution considerably. Thus, you should enable it only if you suspect that your model contains a user-written S-function that has a bug.
- This option causes Simulink software to check whether a block writes outside the memory allocated to it during simulation. Typically this can happen only if your model includes a user-written S-function that has a bug.
- See Checking Array Bounds in “Handle Errors in S-Functions” for more information on using this option.
- For models referenced in Accelerator mode, Simulink ignores the **Array bounds exceeded** parameter setting if you set it to a value other than **None**.

You can use the Model Advisor to identify referenced models for which Simulink changes configuration parameter settings during accelerated simulation.

- 1 In the Simulink Editor, in the **Modeling** tab, click **Model Advisor**, then click **OK**.
- 2 Select **By Task**.
- 3 Run the **Check diagnostic settings ignored during accelerated model reference simulation** check.

Command-Line Information

Parameter: ArrayBoundsChecking

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	warning
Traceability	No impact
Efficiency	none
Safety precaution	No impact

See Also

Related Examples

- Diagnosing Simulation Errors
- Data Validity Diagnostics on page 6-2

Model Verification block enabling

Description

Enable model verification blocks in the current model either globally or locally.

Category: Diagnostics

Settings

Default: Use local settings

Use local settings

Enables or disables blocks based on the value of the **Enable assertion** parameter of each block. If a block's **Enable assertion** parameter is on, the block is enabled; otherwise, the block is disabled.

Enable All

Enables all model verification blocks in the model regardless of the settings of their **Enable assertion** parameters.

Disable All

Disables all model verification blocks in the model regardless of the settings of their **Enable assertion** parameters.

Dependency

Simulation and code generation ignore the **Model Verification block enabling** parameter when model verification blocks are inside a S-function.

Command-Line Information

Parameter: AssertControl

Value: 'UseLocalSettings' | 'EnableAll' | 'DisableAll'

Default: 'UseLocalSettings'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	EnableAll for simulation or during development DisableAll for production code generation

See Also

Related Examples

- Diagnosing Simulation Errors
- Data Validity Diagnostics on page 6-2

Check undefined subsystem initial output

Description

Specify whether to display a warning if the model contains a conditionally executed subsystem in which a block with a specified initial condition drives an Output port block with an undefined initial condition

Category: Diagnostics

Settings

Default: On

On

Displays a warning if the model contains a conditionally executed subsystem in which a block with a specified initial condition drives an Output port block with an undefined initial condition.


Off

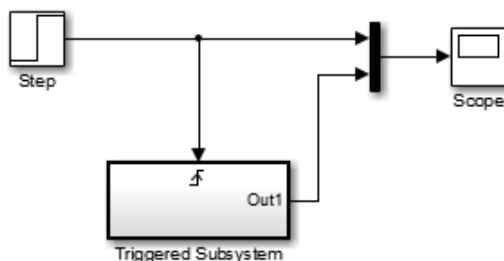
Does not display a warning.

Tips

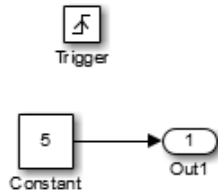
- This situation occurs when a block with a specified initial condition, such as a Constant, Initial Condition, or Delay block, drives an Output port block with an undefined initial condition (**Initial output** parameter is set to []).
- Models with such subsystems can produce initial results (i.e., before initial activation of the conditionally executed subsystem) in the current release that differ from initial results produced in Release 13 or earlier releases.

Consider for example the following model.

 ex_check_undefined_subsys_initial_output ▶

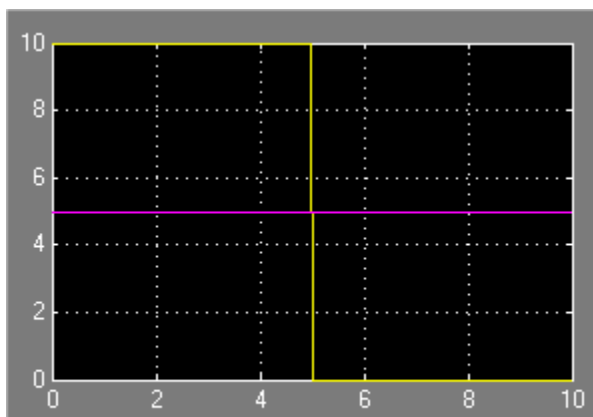


ex_check_undefined_subsys_initial_output ▶ Triggered Subsystem

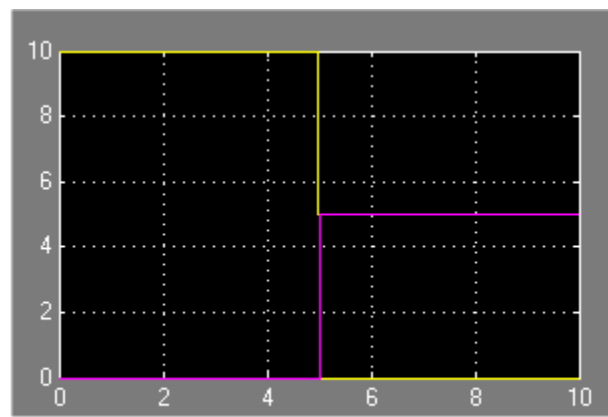


This model does not define the initial condition of the triggered subsystem's output port.

The following figure compares the superimposed output of this model's Step block and the triggered subsystem in Release 13 and the current release.



Release 13



Current Release

Notice that the initial output of the triggered subsystem differs between the two releases. This is because Release 13 and earlier releases use the initial output of the block connected to the output port (i.e., the Constant block) as the triggered subsystem's initial output. By contrast, this release outputs 0 as the initial output of the triggered subsystem because the model does not specify the port's initial output.

Dependency

This parameter is enabled only if **Underspecified initialization detection** is set to **Classic**.

Command-Line Information

Parameter: CheckSSInitialOutputMsg

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	On

See Also

Related Examples

- Diagnosing Simulation Errors
- “Conditionally Executed Subsystems and Models”
- “Underspecified initialization detection” on page 2-65
- “Model Configuration Parameters: Diagnostics” on page 9-2

Detect multiple driving blocks executing at the same time step

Description

Select the diagnostic action to take when the software detects a Merge block with more than one driving block executing at the same time step.

Category: Diagnostics

Settings

Default: error

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink terminates the simulation and displays an error message only if the execution order of the driving blocks is not explicitly defined.

Tips

- Connecting the inputs of a Merge block to multiple driving blocks that execute at the same time step can lead to inconsistent results for both simulation and generated code. Set **Detect multiple driving blocks executing at the same time step** to **error** to avoid such situations.
- This diagnostic action does not cross into referenced models. For example, a test harness does not detect when the model under test contains a Merge block with more than one driving block executing at the same time step. The model under test is referenced by a Model block in the test harness.
- This diagnostic does not apply when one function-call initiator block, such as a Stateflow® Chart or MATLAB Function block, determines the execution order of the driving blocks that connect to the Merge block. In this case, the Merge block results are consistent despite having multiple driving blocks that execute at the same time step.

Command-Line Information

Parameter: MergeDetectMultiDrivingBlocksExec

Value: 'none' | 'warning' | 'error'

Default: 'error'

Recommended Settings

Application	Setting
Debugging	error
Traceability	error

Application	Setting
Efficiency	No impact
Safety precaution	error

See Also

Merge

Related Examples

- Diagnosing Simulation Errors
- “Check usage of Merge blocks”
- “Underspecified initialization detection” on page 2-65
- Data Validity Diagnostics on page 6-2

Underspecified initialization detection

Description

Select how Simulink software handles initialization of initial conditions for conditionally executed subsystems, Merge blocks, subsystem elapsed time, and Discrete-Time Integrator blocks.

Category: Diagnostics

Settings

Default: Simplified

Classic

Initial conditions are initialized the same way they were prior to R2008b.

Simplified

Initial conditions are initialized using the enhanced behavior, which can improve the consistency of simulation results.

Tips

- Use **Classic** to ensure compatibility with previous releases of Simulink. Use **Simplified** to improve 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. For more information, see “Simplified Initialization Mode” and “Classic Initialization Mode”.
- For existing models, MathWorks® recommends using the Model Advisor to migrate your model to the new settings. To migrate your model to simplified initialization mode, run the following Model Advisor checks:
 - “Check usage of Merge blocks”
 - “Check usage of Outport blocks”
 - “Check usage of Discrete-Time Integrator blocks”
 - “Check model settings for migration to simplified initialization mode”

For more information, see “Convert from Classic to Simplified Initialization Mode”.

- When using **Simplified** initialization mode, you must set “Bus signal treated as vector” on page 5-12 to error on the Connectivity Diagnostics pane.

Dependencies

Selecting **Classic** enables the following parameter:

- **Check undefined subsystem initial output**

Selecting **Simplified** disables this parameter.

Command-Line Information

Parameter: UnderspecifiedInitializationDetection

Value: 'Classic' | 'Simplified'

Default: 'Simplified'

Recommended Settings

Application	Setting
Debugging	Simplified
Traceability	Simplified
Efficiency	Simplified
Safety precaution	Simplified

See Also

Merge | Discrete-Time Integrator

Related Examples

- “Convert from Classic to Simplified Initialization Mode”
- “Conditional Subsystem Initial Output Values”
- “Conditionally Executed Subsystems and Models”
- “Simplified Initialization Mode”
- “Classic Initialization Mode”
- “Conditional Subsystem Output Values When Disabled”
- Diagnosing Simulation Errors
- Data Validity Diagnostics on page 6-2

Solver data inconsistency

Description

Select the diagnostic action to take if Simulink software detects S-functions that have continuous sample times, but do not produce consistent results when executed multiple times.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- Consistency checking can cause a significant decrease in performance (up to 40%).
- Consistency checking is a debugging tool that validates certain assumptions made by Simulink ODE solvers. Use this option to:
 - Validate your S-functions and ensure that they adhere to the same rules as Simulink built-in blocks.
 - Determine the cause of unexpected simulation results.
 - Ensure that blocks produce constant output when called with a given value of t (time).
- Simulink software saves (caches) output, the zero-crossing, the derivative, and state values from one time step for use in the next time step. The value at the end of a time step can generally be reused at the start of the next time step. Solvers, particularly stiff solvers such as `ode23s` and `ode15s`, take advantage of this to avoid redundant calculations. While calculating the Jacobian matrix, a stiff solver can call a block's output functions many times at the same value of t .
- When consistency checking is enabled, Simulink software recomputes the appropriate values and compares them to the cached values. If the values are not the same, a consistency error occurs. Simulink software compares computed values for these quantities:
 - Outputs
 - Zero crossings
 - Derivatives
 - States

Command-Line Information

Parameter: ConsistencyChecking

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	warning
Traceability	No impact
Efficiency	none
Safety precaution	No impact

See Also

Related Examples

- Diagnosing Simulation Errors
- Choosing a Solver
- Solver Diagnostics on page 9-2

Ignored zero crossings

Description

Select the diagnostic action to take if Simulink detects zero-crossings that are being ignored.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- Consistency checking can cause a significant decrease in performance (up to 40%).
- Consistency checking is a debugging tool that validates certain assumptions made by Simulink ODE solvers. Use this option to:
 - Validate your S-functions and ensure that they adhere to the same rules as Simulink built-in blocks.
 - Determine the cause of unexpected simulation results.
 - Ensure that blocks produce constant output when called with a given value of t (time).
- Simulink software saves (caches) output, the zero-crossing, the derivative, and state values from one time step for use in the next time step. The value at the end of a time step can generally be reused at the start of the next time step. Solvers, particularly stiff solvers such as `ode23s` and `ode15s`, take advantage of this to avoid redundant calculations. While calculating the Jacobian matrix, a stiff solver can call a block's output functions many times at the same value of t .
- When consistency checking is enabled, Simulink software recomputes the appropriate values and compares them to the cached values. If the values are not the same, a consistency error occurs. Simulink software compares computed values for these quantities:
 - Outputs
 - Zero crossings
 - Derivatives
 - States

Command-Line Information

Parameter: IgnoredZcDiagnostic

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	warning
Traceability	No impact
Efficiency	none
Safety precaution	No impact

See Also

Related Examples

- Solver Diagnostics on page 9-2

Masked zero crossings

Description

Select the diagnostic action to take if Simulink detects zero-crossings that are being masked.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

This parameter is enabled only if **Type** is set to Variable-step.

Command-Line Information

Parameter: MaskedZcDiagnostic

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	warning
Traceability	No impact
Efficiency	none
Safety precaution	No impact

See Also

Related Examples

- Solver Diagnostics on page 9-2

Block diagram contains disabled library links

Description

Select the diagnostic action to take when saving a model containing disabled library links.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning and saves the block diagram. The diagram may not contain the information you had intended.

error

Simulink software displays an error message. The model is not saved.

Tip

Use the Model Advisor Identify disabled library links check to find disabled library links.

Command-Line Information

Parameter: SaveWithDisabledLinksMsg

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Disable or Break Links to Library Blocks”
- “Identify disabled library links”
- Saving a Model

- Solver Diagnostics on page 9-2

Block diagram contains parameterized library links

Description

Select the diagnostic action to take when saving a model containing parameterized library links.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning and saves the block diagram. The diagram may not contain the information you had intended.

error

Simulink software displays an error message. The model is not saved.

Tips

- Use the Model Advisor Identify parameterized library links check to find parameterized library links.

Command-Line Information

Parameter: SaveWithParameterizedLinksMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Identify parameterized library links”
- Solver Diagnostics on page 9-2

Initial state is array

Description

Message behavior when the initial state is an array

Category: Diagnostics

Settings

Default: warning

warning

Simulink software displays a warning when the initial state is an array. If the order of the elements in the array do not match the order in which blocks initialize, the simulation can produce unexpected results.

error

Simulink software displays an error message when the initial state is an array.

none

Simulink software does not display a message when the initial state is an array.

Tips

- Avoid using an array for the initial state. If the order of the elements in the array does not match the order in which blocks initialize, the simulation can produce unexpected results. To promote deterministic simulation results, use the default setting or set the diagnostic to `error`.
- Instead of using array format for the initial state, use a format such as structure, structure with time, or Dataset.

Command-Line Information

Parameter: InitInArrayFormatMsg

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	Use the default setting of warning.
Traceability	Use the default setting of warning.
Efficiency	Use the default setting of warning.
Safety precaution	Use the default setting of warning.

See Also

Related Examples

- “Initial state” on page 3-6
- “Save Block States and Simulation Operating Points”
- “Dataset Conversion for Logged Data”
- “Model Configuration Parameters: Diagnostics” on page 9-2

Insufficient maximum identifier length

Description

For referenced models, specify diagnostic action when the configuration parameter **Maximum identifier length** does not provide enough character length to make global identifiers unique across models.

Category: Diagnostics

Settings

Default: warning

warning

The code generator displays a warning message when the configuration parameter **Maximum identifier length** does not provide enough character length to make global identifiers unique across models. The code generator truncates the identifier to fit the specified value in the configuration parameter **Maximum identifier length** in the generated code.

error

The code generator displays an error message when the configuration parameter **Maximum identifier length** does not provide enough character length to make global identifiers unique across models.

none

The code generator does not display a message when the configuration parameter **Maximum identifier length** does not provide enough character length to make global identifiers unique across models. The code generator truncates the identifier to fit the specified value in the configuration parameter **Maximum identifier length** in the generated code.

Command-Line Information

Parameter: ModelReferenceSymbolNameMessage

Type: character vector

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	Use the default setting of warning.
Traceability	Use the default setting of warning.
Efficiency	Use the default setting of warning.
Safety precaution	Use the default setting of warning.

See Also

Related Examples

- Maximum identifier length (Simulink Coder)
- “Model Configuration Parameters: Diagnostics” on page 9-2
- “Specify Identifier Length to Avoid Naming Collisions” (Simulink Coder)
- “Set Configuration Parameters for Code Generation of Model Hierarchies” (Simulink Coder)

Import custom code

Description

Specify whether or not to parse available custom code variables and functions and compile custom code into its own simulation target. This option affects the C Caller block, the C Function block, the MATLAB Function block, the MATLAB System block, and Stateflow charts.

Category: Simulation Target

Settings

Default: On

On

When this option is on, Simulink:

- Uses the same custom code for simulation with the C Caller block, the C Function block, the MATLAB Function block, the MATLAB System block, and Stateflow charts. When using the C Caller block or the C Function block, this option must be turned on.
- Automatically rebuilds the custom code simulation target when specified custom code dependencies change.
- Automatically rebuilds simulation targets for blocks using custom code when custom code changes.
- Enables Just-In-Time (JIT) compilation of the C Caller block, the C Function block, the MATLAB Function block, the MATLAB System block, and Stateflow charts.
- Allows the **Enable custom code analysis** option to enable Simulink Coverage™ and Simulink Design Verifier™ support for custom code.
- Allows edit and compile time error detection for C interface errors from your model.
- Calls specified initialize code and terminate code at the start and end of model simulation, respectively, regardless of whether any block in the model calls external custom code. See **Initialize function** and **Terminate function**.
- Enables parsing of custom code to report unresolved symbols in Stateflow charts in your model.
- Requires that your custom code be complete and not dependent on any other files in order to be built.

Off

When this option is off, Simulink:

- Combines Simulation Target custom code dependencies with those specified by other means (`coder.cinclude`, `coder.updateBuildInfo`, and `coder.ExternalDependency`) in Stateflow charts that use MATLAB as the action language, MATLAB Function blocks, and MATLAB System blocks.
- Calls specified initialize code and terminate code only if necessary. If no block in the model calls external custom code, Simulink does not call the initialize code or the terminate code. See **Initialize function** and **Terminate function**.

Note When this option is on, if you use the same custom code across different unique configurations, each unique configuration is treated as a separate unit even if the configurations refer to the same custom code. For instance, if you have a root-level model and a library subsystem that refer to the same custom code, then the custom code global variables accessed by the library block and the root-level model are different.

Note In most cases, **Import custom code** should be selected. Clear the **Import custom code** parameter only if your custom code is incompatible with this parameter.

Command-Line Information

Parameter: SimParseCustomCode

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	On
Traceability	No impact
Efficiency	No impact
Safety precaution	On

See Also

Related Examples

- “Reuse Custom Code in Stateflow Charts” (Stateflow)
- “Manage Symbols in the Stateflow Editor” (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Compiler optimization level

Description

Sets the degree of optimization used by the compiler when generating code for acceleration, Stateflow charts, MATLAB Function block, and MATLAB System block.

Category: Simulation Target

Settings

Default: Optimizations off (faster builds)

Optimizations off (faster builds)

Specifies the compiler not to optimize code. This results in faster build times.

Optimizations on (faster runs)

Specifies the compiler to generate optimized code. The generated code will run faster, but the model build will take longer than if optimizations are off.

Tips

- The default `Optimizations off` is a good choice for most models. This quickly produces code that can be used with acceleration.
- Set `Optimizations on` to optimize your code. The fast running code produced by optimization can be advantageous if you will repeatedly run your model with the accelerator.

Command-Line Information

Parameter: SimCompilerOptimization

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Acceleration”

- “Interact with the Acceleration Modes Programmatically”
- “Customize the Acceleration Build Process”
- “Model Configuration Parameters: Code Generation Optimization” (Simulink Coder)

Verbose accelerator builds

Description

Select the amount of information displayed during code generation for Simulink Accelerator mode, referenced model Accelerator mode, and Rapid Accelerator mode.

Category: Simulation Target

Settings

Default: Off

Off

Display limited amount of information during the code generation process.

On

Display progress information during code generation, and show the compiler options in use.

Command-Line Information

Parameter: AccelVerboseBuild

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Controlling Verbosity During Code Generation”
- “Model Configuration Parameters: Code Generation Optimization” (Simulink Coder)

Implement logic signals as Boolean data (vs. double)

Description

Controls the output data type of blocks that generate logic signals.

Category: Simulation Target

Settings

Default: On

On

Blocks that generate logic signals output a signal of `boolean` data type. This reduces the memory requirements of generated code.

Off

Blocks that generate logic signals output a signal of `double` data type. This ensures compatibility with models created by earlier versions of Simulink software.

Tips

- Setting this option **on** reduces the memory requirements of generated code, because a Boolean signal typically requires one byte of storage compared to eight bytes for a `double` signal.
- Setting this option **off** allows the current version of Simulink software to run models that were created by earlier versions of Simulink software that supported only signals of type `double`.
- This optimization affects the following blocks:
 - **Logical Operator block** - This parameter affects only those Logical Operator blocks whose **Output data type** parameter specifies `Inherit: Logical` (see *Configuration Parameters: Optimization*). If this parameter is selected, such blocks output a signal of `boolean` data type; otherwise, such blocks output a signal of `double` data type.
 - **Relational Operator block** - This parameter affects only those Relational Operator blocks whose **Output data type** parameter specifies `Inherit: Logical` (see *Configuration Parameters: Optimization*). If this parameter is selected, such blocks output a signal of `boolean` data type; otherwise, such blocks output a signal of `double` data type.
 - **Combinatorial Logic block** - If this parameter is selected, Combinatorial Logic blocks output a signal of `boolean` data type; otherwise, they output a signal of `double` data type. See *Combinatorial Logic* in the *Simulink Reference* for an exception to this rule.
 - **Hit Crossing block** - If this parameter is selected, Hit Crossing blocks output a signal of `boolean` data type; otherwise, they output a signal of `double` data type.

Dependencies

- This parameter is disabled for models created with a version of Simulink software that supports only signals of type `double`.

Command-Line Information

Parameter: BooleanDataType

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	On
Safety precaution	On

See Also

Related Examples

- “Optimize Generated Code Using Boolean Data for Logical Signals” (Simulink Coder)
- “Math and Data Types Pane” on page 22-2

Block reduction

Description

Reduce execution time by collapsing or removing groups of blocks.

Category: Simulation Target

Settings

Default: On

On

Simulink software searches for and reduces these block patterns:

- **Redundant type conversions** — Unnecessary type conversion blocks, such as an `int` type conversion block with an input and output of type `int`
- **Dead code** — Blocks or signals in an unused code path
- **Fast-to-slow Rate Transition block in a single-tasking system** — Rate Transition blocks with an input frequency faster than its output frequency

Off

Simulink software does not search for block patterns that can be optimized. Simulation and generated code are not optimized.

Tips

- When you select **Block reduction**, Simulink software collapses certain groups of blocks into a single, more efficient block, or removes them entirely. This reduction results in faster execution during model simulation and in generated code.
- Block reduction does not change the appearance of the source model.
- Tunable parameters do not prevent a block from being reduced by dead code elimination.
- Once block reduction takes place, Simulink software does not display the sorted order for blocks that have been removed.
- You can determine programmatically which blocks are reduced in a model by querying the `ReducedNonVirtualBlockList` parameter of the model to obtain a vector of the block handles of the reduced blocks.

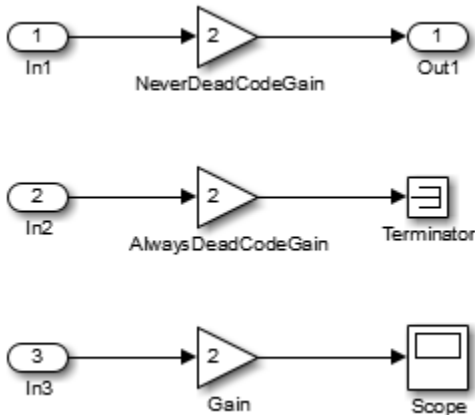
```
ReducedBlockHandlesVector = get_param(ModelName, 'ReducedNonVirtualBlockList');
```

- If you have a Simulink Coder license, block reduction is intended to remove only the generated code that represents execution of a block. Other supporting data, such as definitions for sample time and data types might remain in the generated code.

Dead Code Elimination

Any blocks or signals in an *unused code path* are eliminated from generated code.

- The following conditions need to be met for a block to be considered part of an unused code path:
 - All signal paths for the block end with a block that does not execute. Examples of blocks that do not execute include Terminator blocks, disabled Assertion blocks, S-Function blocks configured for block reduction, and To Workspace blocks when MAT-file logging is disabled for code generation.
 - No signal paths for the block include global signal storage downstream from the block.
- Tunable parameters do not prevent a block from being reduced by dead code elimination.



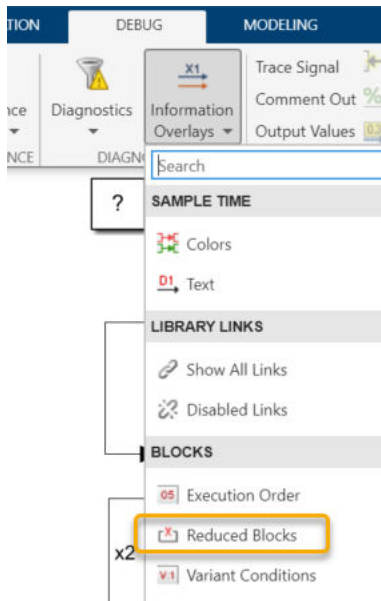
- Consider the signal paths in the following block diagram.

If you check **Block reduction**, Simulink Coder software responds to each signal path as follows:

For Signal Path...	Simulink Coder Software...
In1 to Out1	Generates code because dead code elimination conditions are not met.
In2 to Terminator	Does not generate code because dead code elimination conditions are met.
In3 to Scope	Generates code if MAT-file logging is enabled and eliminates code if MAT-file logging is disabled.

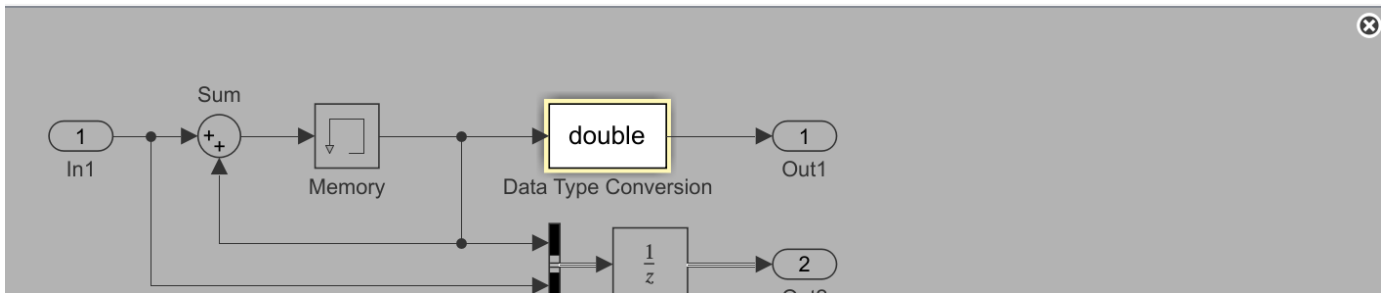
Highlight Reduced Blocks


When **Block reduction** is selected, you can highlight nonvirtual blocks that are removed to reduce execution time during model simulation and code generation. To highlight such blocks, on the Simulink toolstrip, go to the **Debug** tab. From the **Information Overlays** menu, select **Reduced Blocks**.



Note If there are no reduced blocks to highlight, the Reduced Blocks option is disabled.

Reduced blocks appear highlighted on the canvas. After you update or simulate your model, blocks that are reduced during normal simulation are highlighted. After you build your model, blocks that are reduced during code generation are highlighted.



To remove the highlighting, click  in the upper-right corner of the canvas, or clear the Reduced Blocks selection from the **Information Overlays** menu.

Command-Line Information

Parameter: BlockReduction

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	Off for simulation or during development No impact for production code generation
Traceability	Off
Efficiency	On
Safety precaution	No impact

See Also

Related Examples

- “Remove Code for Blocks That Have No Effect on Computational Results” (Simulink Coder)
- “Eliminate Dead Code Paths in Generated Code” (Simulink Coder)
- “Time-Based Scheduling” (Simulink Coder)
- “Performance” (Simulink Coder)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Conditional input branch execution

Description

Improve model execution when the model contains Switch and Multiport Switch blocks.

Category: Simulation Target

Settings

Default: On

On

Executes only the blocks required to compute the control input and the data input selected by the control input. This optimization speeds execution of code generated from the model. Limits to Switch and Multiport Switch block optimization:

- Only blocks with -1 (inherited) or inf (Constant) sample time can participate.
- Blocks with outputs flagged as test points cannot participate.
- No multirate block can participate.
- Blocks with states cannot participate.
- Model blocks cannot participate.
- Only S-functions with option `SS_OPTION_CAN_BE_CALLED_CONDITIONALLY` set can participate.

Off

Executes all blocks driving the Switch block input ports at each time step.

Command-Line Information

Parameter: ConditionallyExecuteInputs

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	On
Efficiency	On (execution), No impact (ROM, RAM)
Safety precaution	No impact

See Also

Related Examples

- “Use Conditional Input Branch Execution” (Simulink Coder)
- “Conditionally Executed Subsystems Overview”
- “Performance” (Simulink Coder)
- “Model Configuration Parameters: Simulation Target” on page 13-2
- “Simulink Optimizations and Model Coverage” (Simulink Coverage)

Break on Ctrl+C

Description

Enables responsiveness checks in code generated for MATLAB Function blocks, Stateflow charts, and dataflow domains. This parameter applies to the model during simulation and code generation.

Category: Simulation Target

Settings

Default: On

On

Enables periodic checks for Ctrl+C breaks in code generated for MATLAB Function blocks, Stateflow charts, and dataflow domains. Also allows graphics refreshing.

Off

Disables periodic checks for Ctrl+C breaks in code generated for MATLAB Function blocks, Stateflow charts, and dataflow domains. Also disables graphics refreshing.

Caution Without these checks, the only way to end a long-running execution might be to terminate the MATLAB session.

Command-Line Information

Parameter: SimCtrlC

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	On
Traceability	No recommendation
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Control Run-Time Checks”
- “Model Configuration Parameters: Simulation Target” on page 13-2

Compile-time recursion limit for MATLAB functions

Description

For compile-time recursion, control the number of copies of a function that are allowed in the generated code. This parameter applies to MATLAB code in a MATLAB Function block, a Stateflow chart, or a System object associated with a MATLAB System block. This parameter applies to the model during simulation and code generation.

Category: Simulation Target > Advanced parameters

Settings

Default: 50

- To disallow recursion in the MATLAB code, set this parameter to 0.
- The default compile-time recursion limit is high enough for most recursive functions that require compile-time recursion. If code generation fails because of the recursion limit, and you want compile-time recursion, increase the limit. Alternatively, you can change your MATLAB code so that the code generator uses run-time recursion.

Command-Line Information

Parameter: CompileTimeRecursionLimit

Type: integer

Value: valid value

Default: 50

See Also

More About

- “Code Generation for Recursive Functions”
- “Compile-Time Recursion Limit Reached”
- “Model Configuration Parameters: Simulation Target” on page 13-2

Enable implicit expansion in MATLAB functions

Description

Enable implicit expansion in code that is generated for MATLAB code that contains binary operations and functions. This parameter applies to MATLAB code in a MATLAB Function block, a Stateflow chart, or a System object associated with a MATLAB System block. This parameter also applies to the model during simulation. Implicit expansion can change the output size of the binary operations and functions. This option might cause extra code to be generated to accomplish implicit expansion.

Category: Simulation Target > Advanced parameters

Settings

Default: On

On

Enables implicit expansion for code generation of MATLAB code that contains binary operations and functions.

Off

Disables implicit expansion for code generation and model simulation of MATLAB code that contains binary operations and functions. If implicit expansion is disabled, and the MATLAB code requires implicit expansion, code generation and model simulation might generate errors.

Command-Line Information

Parameter: EnableImplicitExpansion

Value: 'on' | 'off'

Default: 'on'

See Also

Related Examples

- “Compatible Array Sizes for Basic Operations”
- “Generate Code With Implicit Expansion Enabled” (MATLAB Coder)
- “Optimize Implicit Expansion in Generated Code” (MATLAB Coder)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Enable run-time recursion for MATLAB functions

Description

Allow recursive functions in code that is generated for MATLAB code that contains recursive functions. This parameter applies to MATLAB code in a MATLAB Function block, a Stateflow chart, or a System object associated with a MATLAB System block. This parameter also applies to the model during simulation. Some coding standards, such as MISRA[®], do not allow recursion. To increase the likelihood of generating code that is compliant with MISRA C[®], clear this option.

Category: Simulation Target > Advanced parameters

Settings

Default: On



On

Enables run-time recursion for code generation of MATLAB code that contains recursive functions.



Off

Disables run-time recursion for code generation of MATLAB code that contains recursive functions. If run-time recursion is disabled, and the MATLAB code requires run-time recursion, code generation fails.

Command-Line Information

Parameter: EnableRuntimeRecursion

Value: 'on' | 'off'

Default: 'on'

See Also

More About

- “Code Generation for Recursive Functions”
- “Compile-Time Recursion Limit Reached”
- “Model Configuration Parameters: Simulation Target” on page 13-2

Dynamic memory allocation in MATLAB functions

Description

Use dynamic memory allocation (malloc) for variable-size arrays whose size (in bytes) is greater than or equal to the dynamic memory allocation threshold. This parameter applies to MATLAB code in a MATLAB Function block, a Stateflow chart, or a System object™ associated with a MATLAB System block. This parameter applies to the model during simulation and code generation. This parameter does not apply to:

- Input or output signals
- Parameters
- Global variables
- Discrete state properties of System objects associated with a MATLAB System block

Category: Simulation Target > Advanced parameters

Settings

Default: On (for GRT-based targets) | Off (for ERT-based targets)

On

Enables dynamic memory allocation.

Off

Disables dynamic memory allocation.

Dependency

Enables the **Dynamic memory allocation threshold in MATLAB functions** parameter.

Tips

- Code that uses dynamic memory allocation can be less efficient than code that uses static memory allocation. Unless your model requires dynamic memory allocation, consider clearing this check box.
- If sufficient memory is not available to satisfy a memory allocation request, dynamic memory allocation can fail. The code generator does not check memory allocation requirements. For safety-critical systems, the recommended setting for this parameter is `Off`.

Command-Line Information

Parameter: MATLABDynamicMemAlloc

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Off
Safety precaution	Off

See Also

Related Examples

- “Control Memory Allocation for Variable-Size Arrays in a MATLAB Function Block”
- “Model Configuration Parameters: Simulation Target” on page 13-2

Dynamic memory allocation threshold in MATLAB functions

Description

Specify a threshold for dynamic memory allocation. The code generator uses dynamic memory allocation for variable-size arrays whose size (in bytes) is greater than or equal to the threshold. This parameter applies to MATLAB code in a MATLAB Function block, a Stateflow chart, or a System object associated with a MATLAB System block. This parameter applies to the model during simulation and code generation. This parameter does not apply to:

- Input or output signals
- Parameters
- Global variables
- Discrete state properties of System objects associated with a MATLAB System block

Category: Simulation Target > Advanced parameters

Settings

Default: 65536

- To specify the threshold, set this parameter to a positive integer.
- To use dynamic memory allocation for all variable-size arrays, set this parameter to 0.

Dependency

Dynamic memory allocation in MATLAB functions enables this parameter.

Command-Line Information

Parameter: MATLABDynamicMemAllocThreshold

Type: integer

Value: integer value

Default: 65536

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- “Control Memory Allocation for Variable-Size Arrays in a MATLAB Function Block”
- “Model Configuration Parameters: Simulation Target” on page 13-2

Echo expressions without semicolons

Description

Enable run-time output in the MATLAB Command Window, such as actions that do not terminate with a semicolon. This behavior applies to a model that contains MATLAB Function blocks, Stateflow charts, or Truth Table blocks.

Category: Simulation Target

Settings

Default: On

On

Enables run-time output to appear in the MATLAB Command Window during simulation.

Off

Disables run-time output from appearing in the MATLAB Command Window during simulation.

Tip

- If you disable run-time output, faster model simulation occurs.

Command-Line Information

Parameter: SFSimEcho

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	On
Traceability	No impact
Efficiency	Off
Safety precaution	No impact

See Also

Related Examples

- “Speed Up Simulation” (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Enable continuous-time MATLAB functions to write to initialized persistent variables

Description

Enables continuous-time MATLAB functions to write to initialized persistent variables. If disabled, continuous-time MATLAB functions can only initialize and read persistent variables. To initialize a persistent variable, check that it is empty before assigning a value. For more information, see “Initialize Persistent Variables in MATLAB Functions”.

Category: Simulation Target

Settings

Default: Off

Off

Continuous-time MATLAB functions can only initialize and read persistent variables.

On

Continuous-time MATLAB functions can write to initialized persistent variables.

Tips

- Enable this configuration to ensure legacy functionality in models designed in releases older than R2017b.

Command-Line Information

Parameter: LegacyBehaviorForPersistentVarInContinuousTime

Value: "on" | "off" |

Default: "off"

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Initialize Persistent Variables in MATLAB Functions”

- “Model Configuration Parameters: Simulation Target” on page 13-2

Allow setting breakpoints during simulation

Description

Enables adding breakpoints in MATLAB Function blocks, Stateflow charts, State Transition blocks, and Truth Table blocks during simulation.

Category: Simulation Target

Settings

Default: Off

Off

Model does not support adding breakpoints in MATLAB Function blocks, Stateflow charts, State Transition blocks, or Truth Table blocks during simulation. The debugger for these blocks is enabled only when you add breakpoints before running your model. If there are no breakpoints when simulation begins, the running time is optimized but the model does not check for new breakpoints that you add during simulation.

On

Model supports adding breakpoints in MATLAB Function blocks, Stateflow charts, State Transition blocks, and Truth Table blocks during simulation. The debugger for these blocks is always enabled. You can pause the simulation, add breakpoints, and resume the simulation, but the model runs slower than when debugging is disabled.

Tips

- Simulink does not save the setting for this configuration parameter with your model. You must re-enable this configuration parameter every time you open your model.
- Enabling this configuration parameter has significant performance impact on models that contain multiple MATLAB Function blocks, Stateflow charts, State Transition blocks, or Truth Table blocks.

Command-Line Information

Parameter: SFSimEnableDebug

Value: 'on' | 'off' |

Default: 'off'

Recommended Settings

Application	Setting
Debugging	On
Traceability	No impact
Efficiency	Off

Application	Setting
Safety precaution	No impact

See Also

Related Examples

- “Debug MATLAB Function Blocks”
- “Set Breakpoints to Debug Charts” (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Reserved names

Description

Enter the names of variables or functions in the generated code that match the names of variables or functions specified in custom code for a model that contains MATLAB Function blocks, Stateflow charts, or Truth Table blocks.

Category: Simulation Target

Settings

Default: {}

This action changes the names of variables or functions in the generated code to avoid name conflicts with identifiers in custom code. Reserved names must be shorter than 256 characters.

Tips

- Start each reserved name with a letter or an underscore to prevent error messages.
- Each reserved name must contain only letters, numbers, or underscores.
- Separate the reserved names using commas or spaces.
- You can also specify reserved names by using the command line:

```
config_param_object.set_param('SimReservedNameArray', {'abc', 'xyz'})
```

where `config_param_object` is the object handle to the model settings in the Configuration Parameters dialog box.

Command-Line Information

Parameter: SimReservedNameArray

Type: cell array of character vectors or string array

Value: any reserved names shorter than 256 characters

Default: {}

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- “Model Configuration Parameters: Simulation Target” on page 13-2

Enable memory integrity checks

Description

Detects violations of memory integrity while building MATLAB Function blocks. Stops simulation with a diagnostic message.

Category: Simulation Target

Settings

Default: On for simulation

On for simulation

Detect violations of memory integrity while simulating MATLAB Function blocks in normal and accelerator modes. Stops simulation and displays a diagnostic message.

Off

Does not detect violations of memory integrity while building MATLAB Function blocks.

Always on

Detect violations of memory integrity while building MATLAB Function blocks for all simulation modes. Stops simulation and displays a diagnostic message.

Caution Without these checks, violations result in unpredictable behavior.

Tips

- The most likely cause of memory integrity issues is accessing an array out of bounds.
- Disable these checks only if you are sure that all array bounds and dimension checking is unnecessary.

Command-Line Information

Parameter: SimIntegrity

Value: 'on' | 'off' | 'alwaysOn'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	On
Traceability	No impact
Efficiency	No recommendation
Safety precaution	Always on

See Also

Related Examples

- “Control Run-Time Checks”
- “Model Configuration Parameters: Simulation Target” on page 13-2

Generate typedefs for imported bus and enumeration types

Description

Determines typedef handling and generation for imported bus and enumeration data types in Stateflow and MATLAB Function blocks.

Category: Simulation Target

Settings

Default: Off

On

The software will generate its own typedefs for imported bus and enumeration types.

Off

The software will not generate its own typedefs for imported bus and enumeration types, and will use definitions in the included header file. This setting requires you to include header files in Configuration Parameters, which can be done by navigating to the **Simulation Target** pane.

Tips

- This selection applies if you are using imported bus or enumeration data types in Stateflow and MATLAB Function blocks.

Command-Line Information

Parameter: SimGenImportedTypeDefs

Value: 'on' | 'off'

Default: 'off'

See Also

Related Examples

- “Model Configuration Parameters: Simulation Target” on page 13-2

Use local custom code settings (do not inherit from main model)

Description

Specify if a library model can use custom code settings that are unique from the main model.

Category: Simulation Target

Settings

Default: On

On

Enables a library model to use custom code settings that are unique from the main model.

Off

Disables a library model from using custom code settings that are unique from the main model.

Dependency

This parameter only applies to MATLAB Function blocks, Stateflow charts, or Truth Table blocks in the library model.

Command-Line Information

Parameter: SimUseLocalCustomCode

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- Including Custom C Code (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Allow symbolic dimension specification

Description

Specify whether Simulink propagates dimension symbols throughout the model and preserves these symbols in the propagated signal dimensions.

Category: Diagnostics

Settings

Default: On

On

Simulink propagates symbolic dimensions throughout the model and preserves these symbols in the propagated signal dimensions. If you have an Embedded Coder license, these symbols go into the generated code.

Off

Simulink does not propagate symbolic dimensions throughout the model nor preserve these symbols in propagated signal dimensions.

Command-Line Information

Parameter: AllowSymbolicDim

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- “Implement Symbolic Dimensions for Array Sizes in Generated Code” (Embedded Coder)
- “Model Configuration Parameters: Diagnostics” on page 9-2

Enable decoupled continuous integration

Description

Removes the coupling between continuous and discrete rates. In some cases, unnecessary coupling between the two can cause the integration to be limited by the fastest discrete rate in the model. This coupling might slow down the model.

Category: Solver

Settings

Default: Off

On

Enable the solver to decouple continuous integration from discrete rates.

Off

Preserve the coupling between continuous integration and discrete rates.

Tip

Enabling this parameter can improve simulation speed when:

- A variable step solver is used.
- The model has both continuous and discrete rates.
- The fastest discrete rate is relatively smaller than the maximum step size set by the solver.

Command-Line Information

Parameter: DecoupledContinuousIntegration

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- “Solver Pane” on page 14-2

Enable minimal zero-crossing impact integration

Reduces the impact of zero-crossing on the integration of continuous states.

Settings

Default: Off

On

The solver tries to reduce the impact of zero-crossings on the integration of continuous states

Off

The solver won't try to reduce the impact of zero-crossings on the integration of continuous states

Dependencies

- Solver **Type** must be set to Variable-Step.

Tips

Command-Line Information

Parameter: MinimalZcImpactIntegration

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

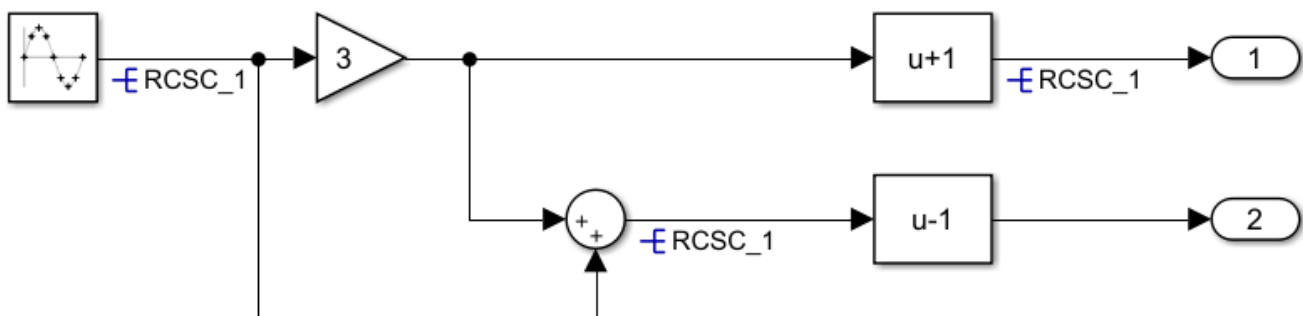
Application	Setting
Debugging	off
Traceability	off
Efficiency	on
Safety precaution	No impact

Detect ambiguous custom storage class final values

Description

Select the diagnostic action to take when your model contains a Reusable custom storage class that has more than one endpoint. An endpoint is a usage of a Reusable custom storage class with no other downstream usages.

If your model contains a Reusable custom storage class that does not have a unique endpoint, the run-time environment must not use the variable value because the value is ambiguous. For example, in this model, the final value of RCSC_1 is ambiguous because it has two endpoints. If you remove the specification from the signal line that leaves the Sum block or from the signal line that leaves the top Bias block, the Reusable custom storage class has one endpoint.



Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tip

If the run-time environment must use the final value of the signal with the Reusable custom storage class specification, set this parameter to error. Remove one of the Reusable custom storage classes so that the Reusable custom storage class has a unique endpoint.

Command-Line Information

Parameter: RCSCObservableMsg

Value: 'none' | 'warning' | 'error'

Default: 'none '

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

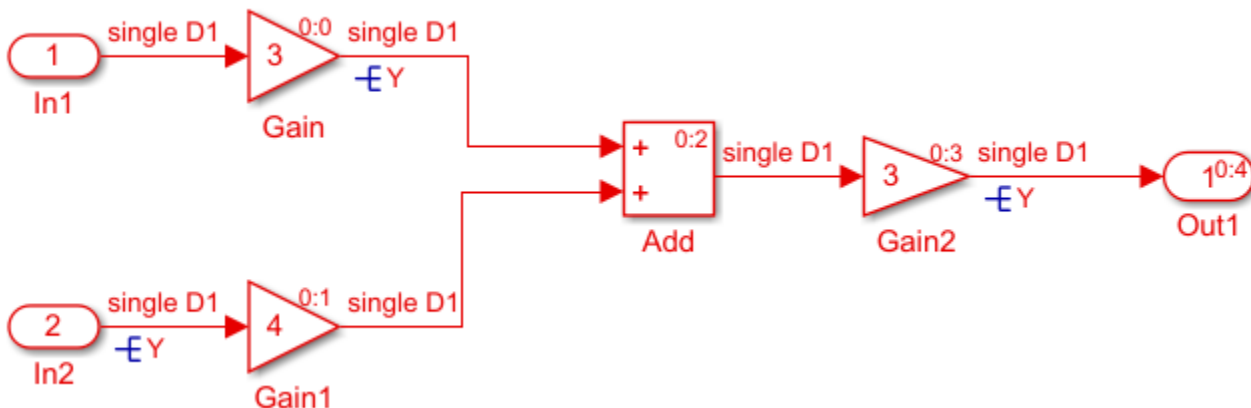
- Data Validity Diagnostics on page 6-2
- “Specify Buffer Reuse for Signals in a Path” (Embedded Coder)
- “Model Configuration Parameters: Code Generation Optimization” (Embedded Coder)

Detect non-reused custom storage classes

Description

Select the diagnostic action to take when your model contains a `Reusable` custom storage class that the code generator cannot reuse with other uses of the same `Reusable` custom storage class. The default behavior of the parameter settings varies with the presence of `Reusable` custom storage classes and referenced models. If the code generator cannot change the block execution order to enable reuse or the conditional execution of some blocks is incompatible with reuse, the code generator might not implement the reuse specification. The generated code will likely contain additional global variables.

For example, in this model, the code generator cannot reuse the variable `Y` to hold the outputs of `In2`, `Gain`, and `Gain2` because `Gain` executes before `Gain2`. The generated code contains an extra variable to hold the `Gain` output. The red numbers to the top right of the blocks indicate the execution order.



Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

When there are `Reusable` custom storage classes and referenced models present, the parameter settings are:

None

Simulink software generates a message for you to set the parameter to `Error`.

Warning

Simulink software generates a message for you to set the parameter to `Error`

Error

If `Reusable` custom storage classes can be combined Simulink software generates code. If not, it generates an error.

Tip

If you do not want the generated code to contain additional global variables because of a `Reusable` custom storage class specification that the code generator cannot honor, set this parameter to `error`. Remove the `Reusable` custom storage classes from the signal lines in the error message.

Command-Line Information

Parameter: `RCSCRenamedMsg`

Value: `'none' | 'warning' | 'error'`

Default: `'none'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also**Related Examples**

- Data Validity Diagnostics on page 6-2
- “Specify Buffer Reuse for Signals in a Path” (Embedded Coder)
- “Model Configuration Parameters: Code Generation Optimization” (Embedded Coder)

Combine output and update methods for code generation and simulation

Description

When output and update code is in one function in the generated code, force the simulation execution order to be the same as the code generation order. For certain modeling patterns, setting this parameter prevents a potential simulation and code generation mismatch. Setting this parameter might cause artificial algebraic loops. If your model requires this parameter, Simulink generates a warning of a potential simulation and code generation mismatch during the model build. The warning states that your model

```
...references a model that has an inport that is used during update only but the model combines output and update methods. This may result in a mismatch between simulation and code generation results.
```

Settings

Default: Off

On

Forces simulation execution order to be the same as code generation order when output and update code are in one function. You might get the preceding warning if your model meets these conditions:

- The referenced model has a single output/update function, uses function prototype control, or generates C++ code.
- A referenced model input connects only to blocks that do not use their input values to calculate their output values during the same time step, such as Delay or Integrator blocks. The input port is not associated with a Function-Call Subsystem port in the referenced model.
- The referenced model uses a shared global resource such as a global data store.

Off

For the preceding modeling pattern, the simulation execution order might be different than the code generation order. If the execution order is different, an answer mismatch between simulation and code generation might occur.

Tips

Selecting this parameter might cause artificial algebraic loops in simulation. Select it only if you get a warning about a possible simulation versus code generation mismatch, and you plan to generate code.

Command-Line Information

Parameter: ForceCombineOutputUpdateInSim

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Artificial Algebraic Loops”
- “Model Configuration Parameters: Diagnostics” on page 9-2

Include custom code for referenced models

Description

Use custom code for referenced model simulation (SIM) target build for accelerator mode.

Category: Model Referencing

Settings

Default: Off

Off

Ignore custom code for model reference accelerator simulation.

On

Use custom code with Stateflow or with MATLAB Function blocks during model reference accelerator simulation.

Tips

- **Caution** Using custom code for referenced models in accelerator mode can produce different results than if you simulate the model without using the custom code. If the custom code includes declarations of structures for buses or enumerations, the SIM target generation fails if the build results in duplicate declarations of those structures. Also, if the custom code uses a structure that represents a bus or enumeration, you might get unexpected simulation results.
- Use the **Configuration Parameters > Simulation Target** pane to specify the custom code file.

Command-Line Information

Parameter: SupportModelReferenceSimTargetCustomCode

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	Off

Hardware acceleration

Description

Turn hardware acceleration off or select the level of hardware acceleration in simulation. Hardware acceleration allows you to leverage SIMD instructions to improve simulation performance.

Hardware acceleration does not leverage SIMD instructions when LCC is configured as the MEX compiler.

Settings

Default: Leverage generic hardware (Faster, no rebuild)

Off

Turn off hardware acceleration.

Leverage generic hardware (Faster, no rebuild)

Leverage SIMD instructions for hardware generic to Simulink system requirements. This option does not require you to rebuild the model for simulation when the host computer changes.

Leverage native hardware (Fastest, rebuild allowed)

Leverage SIMD instructions for hardware native to the host computer for best simulation performance. This option may require rebuilding the model for simulation when the host computer changes.

Command-Line Information

Parameter: SimHardwareAcceleration

Value: 'off' | 'generic' | 'native'

Default: 'generic'

Recommended Settings

Application	Setting
Debugging	On
Traceability	No impact
Efficiency	No impact
Safety precaution	On

See Also

Related Examples

- “Model Configuration Parameters: Simulation Target” on page 13-2

Behavior when pregenerated library subsystem code is missing

Description

When you generate code for a model that contains an instance of a reusable library subsystem with a function interface, specify whether or not to display a warning or an error when the model cannot use pregenerated library code or pregenerated library code is missing.

Settings

Default: warning

none

If pregenerated library code is missing, Simulink software does not display a warning or an error. The code generator generates code for a reusable library subsystem that does not contain function interfaces. The generated code for the reusable library subsystem is in the `slprj/target/_sharedutils` folder.

warning

If pregenerated library code is missing, Simulink software displays a warning. The code generator generates code for a reusable library subsystem that does not contain function interfaces. The generated code for the reusable library subsystem is in the `slprj/target/_sharedutils` folder.

error

Simulink software displays an error message. The code generator does not generate code.

Tip

To use pregenerated library code, before generating code for the model, generate code for the library.

Command-Line Information

Parameter: `PregeneratedLibrarySubsystemCodeDiagnostic`

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	error, warning
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Library-Based Code Generation for Reusable Library Subsystems” (Embedded Coder)

Arithmetic operations in variant conditions

Description

Select the diagnostic action to take if Simulink software detects arithmetic operations (+, -, *, idivide, rem) in variant conditions specified within variant blocks.

Category: Diagnostics

Settings

Default: error

Note For models created prior to R2019a, the default value is warning.

none

When Simulink software detects arithmetic operations in variant conditions of a Variant block with the **Variant activation time** option set to `code compile`, the software takes no action.

warning

When Simulink software detects arithmetic operations in variant conditions of a Variant block with the **Variant activation time** option set to `code compile`, the software displays a warning.

error

When Simulink software detects arithmetic operations in variant conditions of a Variant block with the **Variant activation time** option set to `code compile`, the software displays a warning and terminates the simulation.

Note It is recommended to use the default value `error`, as there could be a difference in behavior between simulation and code generation. For example, if you use the condition `V * W == 10` in a Variant Source block and request that the block produces preprocessor conditions in the Simulink Coder generated code. This results in generated C code containing `#if V*W == 10`. Simulink uses `int32` types for `V` and `W`, whereas the integer types used by the compiler are implementation dependent. So, for large values of `V` and `W`, there could be a difference in behavior between simulation and code generation. If the model uses arithmetic operations, you must consider removing their usage rather than relaxing the diagnostic.

Command-Line Information

Parameter: ArithmeticOperatorsInVariantConditions

Value: 'none' | 'warning' | 'error'

Default: 'error'

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- Solver Diagnostics on page 9-2

Variant activation time inherited from Simulink.VariantControl

Description

Select the diagnostic action to take if Simulink software detects variant blocks with activation time set to `inherit` from `Simulink.VariantControl` but no variant control variables of type `Simulink.VariantControl`.

Category: Diagnostics

Settings

Default: warning

none

When Simulink software detects variant blocks with activation time set to `inherit` from `Simulink.VariantControl` but no variant control variables of type `Simulink.VariantControl`, the software takes no action.

warning

When Simulink software detects variant blocks with activation time set to `inherit` from `Simulink.VariantControl` but no variant control variables of type `Simulink.VariantControl`, the software displays a warning.

error

When Simulink software detects variant blocks with activation time set to `inherit` from `Simulink.VariantControl` but no variant control variables of type `Simulink.VariantControl`, the software terminates the simulation and displays an error message.

Command-Line Information

Parameter: `InheritVATfromSVC`

Value: `'none' | 'warning' | 'error'`

Default: `'warning'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- Solver Diagnostics on page 9-2

FMU Import blocks

Description

When the debug execution mode is enabled, FMU binaries are executed in a separate process. Executing FMU binaries in a separate process protects MATLAB processes. For example, the debug execution mode can prevent a segmentation violation in third-party FMU binaries from crashing MATLAB.

Settings

Default: Off

On

Enable debug execution mode for FMU Import blocks. This setting executes FMU binaries in a separate process.

Off

Disable debug execution mode for FMU Import blocks. This setting executes FMU binaries in the same process as the MATLAB process.

Command-Line Information

Parameter: DebugExecutionForFMUViaOutOfProcess

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

Variant condition mismatch at signal source and destination

Description

When you generate code using Embedded Coder, select the diagnostic action to take if the software detects variant-related modeling issues that may result in unused Simulink variables in the generated code. Unused variables are created when there is discrepancy in variant conditions that propagate between the source and the destination blocks while compiling the model. For more on discrepancies, see “Prevent Creation of Unused Variables for Lenient Variant Choices” on page 2-132, or “Prevent Creation of Unused Variables for Unconditional and Conditional Variant Choices” on page 2-135.

Category: Diagnostics

Settings

Default: none

none

When Simulink software detects a discrepancy in variant conditions that propagate between the source and the destination blocks with the **Variant activation time** option set to `code compile`, the software takes no action.

warning

When Simulink software detects a discrepancy in variant conditions that propagate between the source and the destination blocks with the **Variant activation time** option set to `code compile`, the software displays a warning and continues with the simulation. To suppress the warning and continue with simulation, click **Suppress**.

error

When Simulink software detects a discrepancy in variant conditions that propagate between the source and the destination blocks with the **Variant activation time** option set to `code compile`, the software displays an error and terminates the simulation.

Command-Line Information

Parameter: VariantConditionMismatch

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

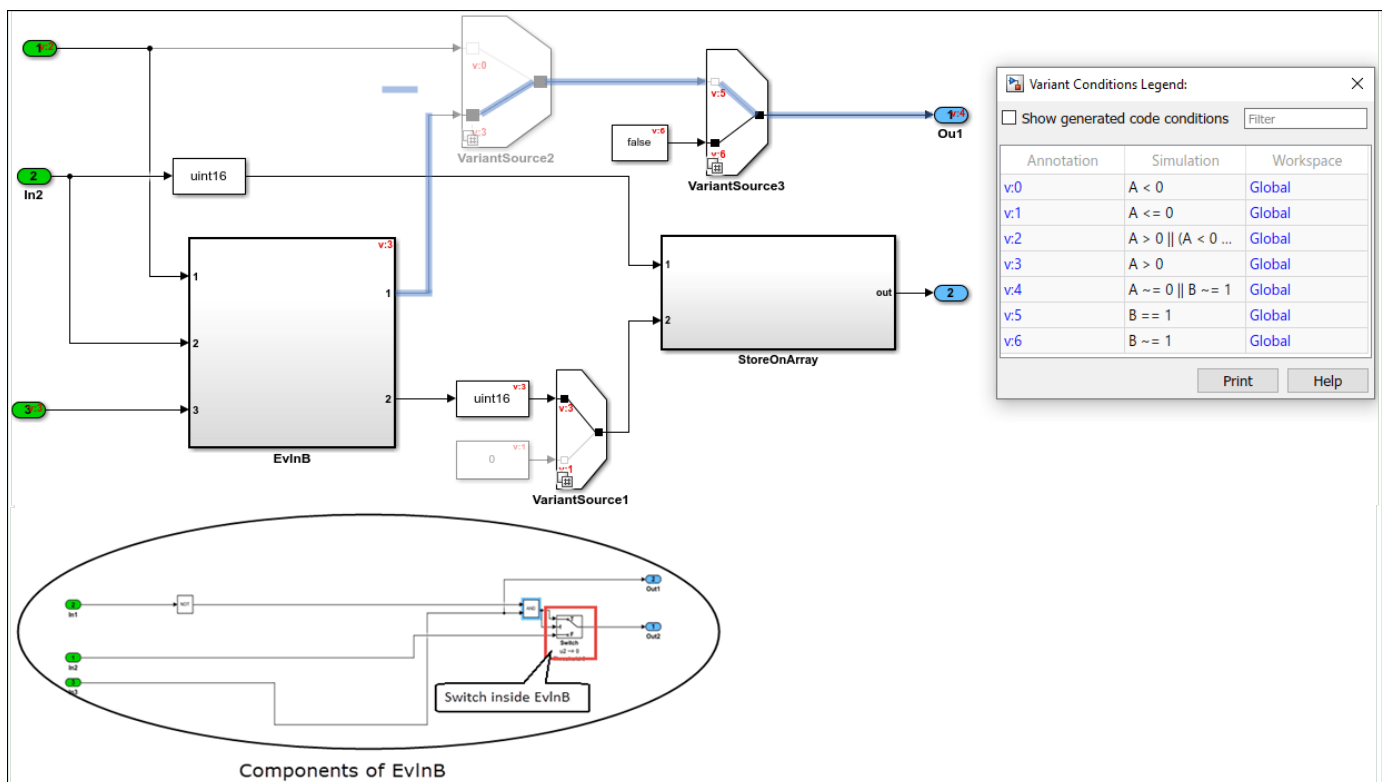
- “Prevent Creation of Unused Variables for Lenient Variant Choices” on page 2-132
- “Prevent Creation of Unused Variables for Unconditional and Conditional Variant Choices” on page 2-135

Prevent Creation of Unused Variables for Lenient Variant Choices

This example shows how to prevent Simulink® models from creating unused variables in generated code when the variant condition of the source block is more lenient than the variant condition of the destination block. Preventing the creation of unused variables in generated code increases the likelihood of generating C code that is compliant with Rule 2.2 of the MISRA C:2012 guidelines.

Model Description

In this model, the source of the highlighted signal is the EvlnB block. The destination of the signal is VariantSource3. The variant condition of EvlnB is $A > 0$, which is more lenient than the variant condition $A > 0 \ \&\& \ B == 1$ of VariantSource3.



Generate C Code Using Embedded Coder

Suppose that the value of A is 1, and the value of B is 0.

When you generate code using Embedded Coder™, the variant condition of EvlnB, $A > 0$, evaluates to true, and the condition of VariantSource3, $A > 0 \ \&\& \ B == 1$, evaluates to false. However, code is generated for all variant choices. In the generated code, the variable `rtb_Switch` is created. This variable corresponds to the Switch block that is located inside the EvlnB block. `rtb_Switch` is used only when $A > 0 \ \&\& \ B == 1$ evaluates to true. In this example, since $A > 0 \ \&\& \ B == 1$ evaluates to false, `rtb_Switch` remains unused.

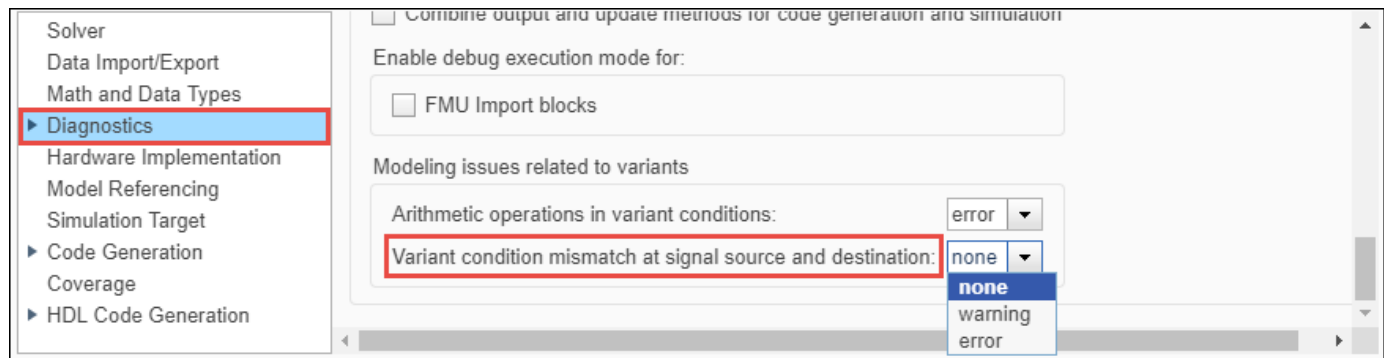
```

35 #if A != 0 || B != 1
36
37     boolean_T rtb_VariantMerge_For_Variant_So;
38
39 #endif
40
41 #if A > 0
42
43     boolean_T rtb_Switch;
44
45 #endif
46
47 /* SignalConversion generated from: '<Root>/VariantSource' incorporates:
48  * Inport: '<Root>/In1'
49  */
50 #if A < 0 && B == 1
51
52     rtb_VariantMerge_For_Variant_So = rtU.In1;
53
54 #elif A > 0
55
56 /* Outputs for Atomic SubSystem: '<Root>/EvInB2' */
57 /* Switch: '<SI>/Switch' incorporates:
58  * Inport: '<Root>/In1'
59  * Inport: '<Root>/In2'
60  * Inport: '<Root>/In3'
61  * Logic: '<SI>/LogicalOperator6'
62  * Logic: '<SI>/LogicalOperator8'
63  */
64     rtb_Switch = (((!rtU.In2) && rtU.In3) || rtU.In1);
65
66 /* End of Outputs for SubSystem: '<Root>/EvInB2' */
67 #endif
68
69 /* End of SignalConversion generated from: '<Root>/VariantSource' */
70
71 /* SignalConversion generated from: '<Root>/VariantSource' incorporates:
72  * SignalConversion generated from: '<Root>/VariantSource3'
73  */
74 #if A > 0 && B == 1
75
76     rtb_VariantMerge_For_Variant_So = rtb_Switch;
77
78 #elif B != 1
79
80 /* SignalConversion generated from: '<Root>/VariantSource' incorporates:
81  * Constant: '<Root>/Constant'
82  */
83     rtb_VariantMerge_For_Variant_So = false;
84
85 #endif

```

rtb_Switch is used only when A>0 && B==1 evaluates to true.

To avoid modeling issues that may create unused variables in generated code, select **Model Settings** > **Diagnostics**. Expand the **Advanced** parameters section and scroll to the bottom of the dialog box. In the **Modeling issues related to variants** section, set the **Variant condition mismatch at signal source and destination** parameter to warning or error. Setting this parameter warns you about unused variables during code generation.



See Also

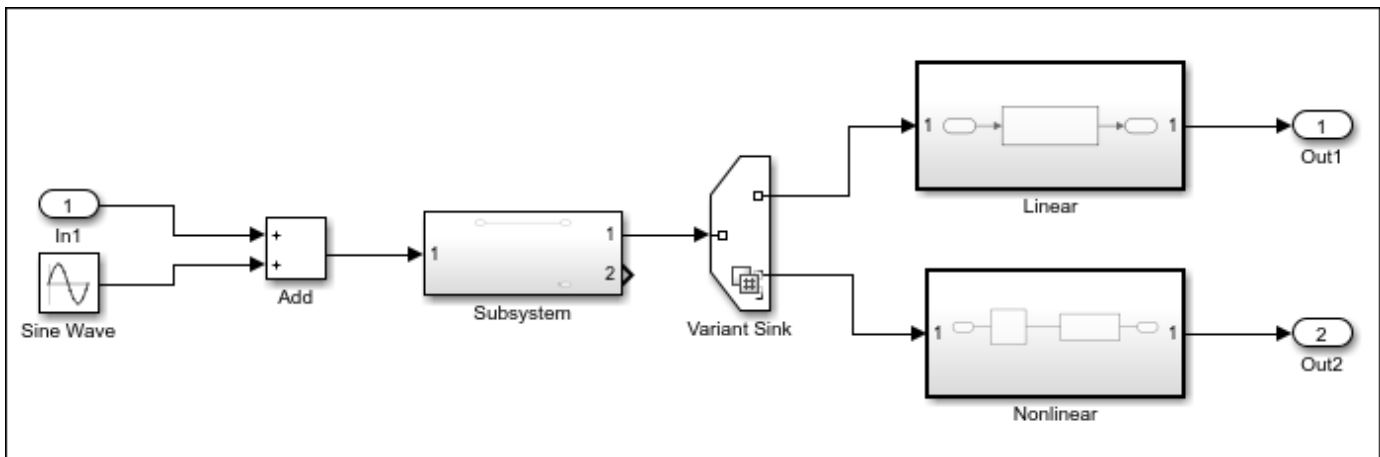
- “Variant condition mismatch at signal source and destination” on page 2-130
- “Prevent Creation of Unused Variables for Unconditional and Conditional Variant Choices” on page 2-135

Prevent Creation of Unused Variables for Unconditional and Conditional Variant Choices

This example shows how to prevent Simulink® models from creating unused variables in generated code when the variant condition of the source block is unconditional and the variant condition of the destination block is conditional. Preventing the creation of unused variables in generated code increases the likelihood of generating C code that is compliant with Rule 2.2 of the MISRA C:2012 guidelines.

Model Description

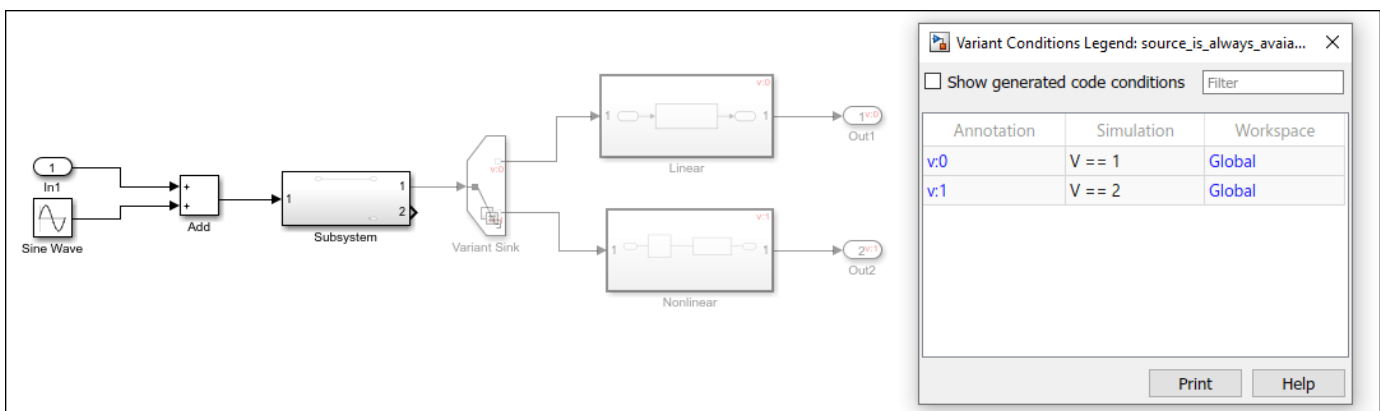
In this model, the Variant Sink block has the variant condition expressions $V == 1$ and $V == 2$.



Generate C Code Using Embedded Coder

Suppose that the value of V is set to 3.

When you generate code using Embedded Coder™, the variant condition expressions $V == 1$ and $V == 2$ evaluate to `false`. Simulink disables all the blocks connected to the input and output stream of the Variant Sink block. However, code is generated for all the variant choices in the model.



A variable `Add` is created in the generated code. During code compilation, this variable remains unused since both choices evaluate to `false`.

```

/* Model step function */
void source_is_always_available_step(void)
{
    real_T Add;

    /* Sum: '<Root>/Add' incorporates:
     * Inport: '<Root>/In1'
     * Sin: '<Root>/Sine Wave'
     */
    Add = source_is_always_available_U.In1 + sin
        (source_is_always_available_M->Timing.t{0});

    /* Output: '<Root>/Out1' incorporates:
     * DiscreteFilter: '<S1>/Discrete Filter'
     * Output: '<Root>/Out1'
     */
    #if V == 1

    /* Outputs for Atomic SubSystem: '<Root>/Linear' */
    /* DiscreteFilter: '<S1>/Discrete Filter' */
    source_is_always_available_DW.DiscreteFilter_states_d = Add - 0.5 *
        source_is_always_available_DW.DiscreteFilter_states_d;
    source_is_always_available_Y.Out1 =
        source_is_always_available_DW.DiscreteFilter_states_d;

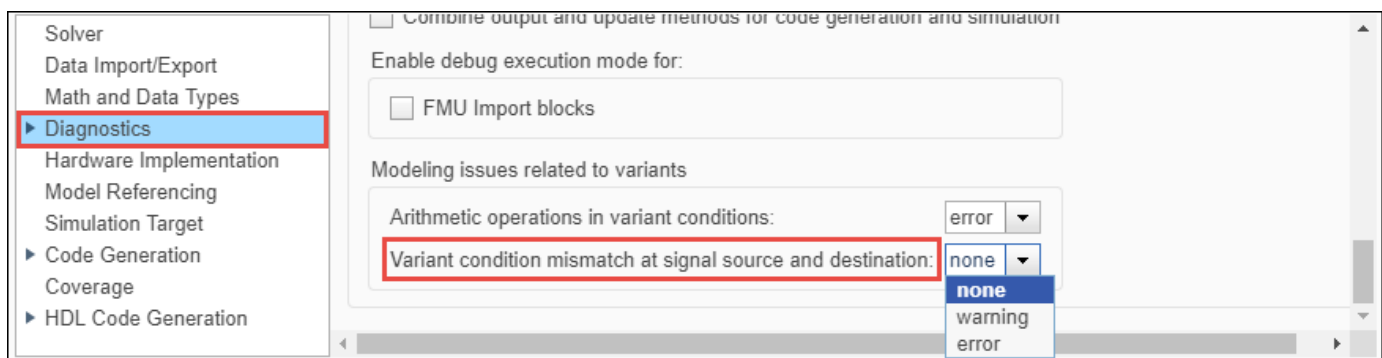
    /* End of Outputs for SubSystem: '<Root>/Linear' */
    #elif V == 2

    /* Outputs for Atomic SubSystem: '<Root>/Nonlinear' */
    /* DiscreteFilter: '<S2>/Discrete Filter' incorporates:
     * Lookup_n-D: '<S2>/Lookup Table'
     */
    source_is_always_available_DW.DiscreteFilter_tmp = look1_binlpxw(Add,
        source_is_always_available_ConstP.LookupTable_bp01Data,
        source_is_always_available_ConstP.LookupTable_tableData, 4U) - 0.5 *
        source_is_always_available_DW.DiscreteFilter_states;

```

Variable `Add` is used only if `V==1` or `V==2` evaluates to true.

To avoid modeling issues that may create unused variables in generated code, select **Model Settings** > **Diagnostics**. Expand the **Advanced** parameters section and scroll to the bottom of the dialog box. In the **Modeling issues related to variants** section, set the **Variant condition mismatch at signal source and destination** parameter to warning or error. Setting this parameter warns you about unused variables during code generation.



See Also

- “Variant condition mismatch at signal source and destination” on page 2-130

- “Prevent Creation of Unused Variables for Lenient Variant Choices” on page 2-132

Variant configuration not used by top model

Description

If your model has predefined variant configurations, select the diagnostic action to take if Simulink detects that a top-level model does not use this model for one of these configurations. This setting helps you to verify that this model is used only for its tested variant configurations.

Settings

Default: warning

none

When Simulink detects during model compilation or Variant Manager activation that a top-level model does not use this model for any of its published variant configurations, the software takes no action.

warning

When Simulink detects during model compilation or Variant Manager activation that a top-level model does not use this model for any of its published variant configurations, the software displays a warning and continues with the simulation. To suppress the warning and continue with simulation, click **Suppress**.

error

When Simulink detects during model compilation or Variant Manager activation that a top-level model does not use this model for any of its published variant configurations, the software displays an error and terminates the simulation.

Command-Line Information

Parameter: VariantConfigNotUsedByTopModel

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

More About

- “Variant Manager for Simulink” on page 21-2

Data Import/Export Parameters

Model Configuration Parameters: Data Import/Export

The **Data Import/Export** category includes parameters for configuring input data for simulation (for example, for Inport blocks) and output data (for example, from Outport blocks). The parameters allow you to import input signal and initial state data from a workspace and export output signal and state data to the MATLAB workspace during simulation. This capability allows you to use standard or custom MATLAB functions to generate a simulated system's input signals and to graph, analyze, or otherwise postprocess the system's outputs.

- 1 Specify the data to load from a workspace before simulation begins.
- 2 Specify the data to save to the MATLAB workspace after simulation completes.

Parameter	Description
"Input" on page 3-4	Loads input data from a workspace before the simulation begins.
"Initial state" on page 3-6	Loads the model initial states from a workspace before simulation begins.
"Time" on page 3-8	Saves simulation time data to the specified variable during simulation.
"States" on page 3-10	Saves state data to the specified variable during a simulation.
"Output" on page 3-12	Saves signal data to the specified variable during simulation.
"Final states" on page 3-14	Saves the model states at the end of a simulation to the specified variable.
"Format" on page 3-16	Select the data format for saving states, output, and final states data.
"Limit data points to last" on page 3-18	Log only the last n data points for outputs and states.
"Decimation" on page 3-20	Specify decimation factor, n , for output and states logging such that only every n points are logged.
"Save final operating point" on page 3-24	At the end of a simulation, Simulink saves the complete set of states of the model, including logged states.
"Signal logging" on page 3-26	Globally enable or disable signal logging for the model.
"Data stores" on page 3-28	Globally enable or disable logging of Data Store Memory block variables for the model.
"Log Dataset data to file" on page 3-30	Log data to MAT-file.
"Output options" on page 3-32	Select options for generating additional output signal data for variable-step solvers.
"Refine factor" on page 3-34	Specify how many points to generate between time steps to refine the output.

Parameter	Description
“Output times” on page 3-35	Specify times at which Simulink software should generate output in addition to, or instead of, the times of the simulation steps taken by the solver used to simulate the model.
“Single simulation output” on page 3-37	Specify whether to return simulation data as a single <code>Simulink.SimulationOutput</code> object.
“Logging intervals” on page 3-39	Set intervals for logging
“Record logged workspace data in Simulation Data Inspector” on page 3-41	Specify whether to send logged data to the Simulation Data Inspector when a simulation pauses or completes.

These configuration parameters are in the **Advanced parameters** section.

Parameter	Description
“Dataset signal format” on page 2-50	Format for <code>Dataset</code> elements.
“Stream To Workspace blocks” on page 2-54	Specify whether data logged using To Workspace blocks streams to the Simulation Data Inspector.

See Also

Related Examples

- Importing Data from a Workspace
- “Export Simulation Data”
- “Export Signal Data Using Signal Logging”
- “Load Signal Data for Simulation”
- “Save Run-Time Data from Simulation”

Input

Description

Loads input data from a workspace for a simulation.

Category: Data Import/Export

Settings

Default: Off, [t,u]

On

Loads data from a workspace.

Specify a MATLAB expression for the data to be loaded into the model from a workspace. The Simulink software resolves symbols in the expression as described in “Symbol Resolution”.

See “Load Data to Root-Level Input Ports” for information.

The **Input** parameter does not load input data from a data dictionary. When a model uses a data dictionary and you disable model access to the base workspace, the Input parameter still accesses simulation input variables in the base workspace.

Off

Does not load data from a workspace.

Tips

- If you use a `Simulink.SimulationData.Dataset` object that includes a `matlab.io.datastore.SimulationDatastore` object as an element, then the data stored in persistent storage is streamed in from a file. For more information, see “Load Big Data for Simulations”.
- You must select the **Input** check box before entering input data.
- Simulink software linearly interpolates or extrapolates input values as necessary if the **Interpolate data** option is selected for the corresponding Input.
- The use of the **Input** box is independent of the setting of the **Format** list on the **Data Import/Export** pane.
- For more information about using the **Input** parameter to load signal data to root-level inputs, see “Load Data to Root-Level Input Ports”.

Programmatic Use

Parameter: LoadExternalInput

Value: 'on' | 'off'

Default: 'off'

Parameter: ExternalInput

Type: character vector

Value: any valid value

Default: '[t,u]'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- "Load Data to Root-Level Input Ports"
- "Model Configuration Parameters: Data Import/Export" on page 3-2

Initial state

Description

Loads the model initial states from a workspace before simulation begins.

Category: Data Import/Export

Settings

Default: Off, `xInitial`

On

Simulink software loads initial states from a workspace.

Specify the name of a variable that contains the initial state values, for example, a variable containing states saved from a previous simulation.

Use the structure or structure-with-time option to specify initial states if you want to accomplish any of the following:

- Associate initial state values directly with the full path name to the states. This eliminates errors that could occur if Simulink software reorders the states, but the initial state array is not correspondingly reordered.
- Assign a different data type to each state's initial value.
- Initialize only a subset of the states.
- Initialize the states of a top model and the models that it references

See “Load State Information” for more information.

The **Initial state** parameter does not load initial state data from a data dictionary. When a model uses a data dictionary and you disable model access to the base workspace, the **Initial State** parameter still has access to resolve variables in the base workspace.

Off

Simulink software does not load initial states from a workspace.

Tips

- The initial values that the workspace variable specifies override the initial values that the model specifies (the values that the initial condition parameters of those blocks in the model that have states specify).
- Selecting the **Initial state** check box does not result in Simulink initializing discrete states in referenced models.
- Avoid using an array for an initial state. If the order of the elements in the array does not match the order in which blocks initialize, the simulation can produce unexpected results. To promote deterministic simulation results, use the **InitInArrayFormatMsg** diagnostic default setting of `warning` or set the diagnostic to `error`.

Instead of array format for the initial state, consider using a `Simulink.SimulationData.Dataset` object, structure, structure with time, or an operating point.

- If you use a format other than `Dataset`, you can convert the logged data to `Dataset` format. Converting the data to `Dataset` makes it easier to postprocess with other logged data. For more information, see “Dataset Conversion for Logged Data”.
- If you use `Dataset` format, you can specify the discrete state bus type by setting the state label to `DSTATE_NVBUS` (nonvirtual bus) or `DSTATE_VBUS` (virtual bus).

Programmatic Use

Parameter: `LoadInitialState`

Value: 'on' | 'off'

Default: 'off'

Parameter: `InitialState`

Type: variable (character vector) or vector

Value: any valid value

Default: 'xInitial'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- Importing Data from a Workspace
- “Save Block States and Simulation Operating Points”
- “Dataset Conversion for Logged Data”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Time

Description

Saves simulation time data to the specified variable during simulation.

Category: Data Import/Export

Settings

Default: On, tout

On

Simulink software exports time data to the MATLAB workspace during simulation.

Specify the name of the MATLAB variable used to store time data. See “Export Simulation Data” for more information.

Off

Simulink software does not export time data to the MATLAB workspace during simulation.

Tips

- You must select the **Time** check box before entering the time variable.
- Simulink software saves the output to the MATLAB workspace at the base sample rate of the model. Use a To Workspace block if you want to save output at a different sample rate.
- **Additional parameters** includes parameters for specifying a limit on the number of data points to export and the decimation factor.
- To specify an interval for logging, use the **Logging intervals** parameter.
- If you use a format other than **Dataset**, you can convert the logged data to **Dataset** format. Converting the data to **Dataset** makes it easier to postprocess with other logged data. For more information, see “Dataset Conversion for Logged Data”.
- Do not use a variable name that is the same as a `Simulink.SimulationOutput` object function name or property name.

Programmatic Use

Parameter: SaveTime

Value: 'on' | 'off'

Default: 'on'

Parameter: TimeSaveName

Type: character vector

Value: any valid value

Default: 'tout'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Export Simulation Data”
- “Data Format for Logged Simulation Data”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

States

Description

Saves states data to the specified variable in the MATLAB workspace.

Category: Data Import/Export

Settings

Default: Off, xout

On

Log states data to the MATLAB workspace.

Specify the name of the variable used to store the logged states data. See “Save Block States and Simulation Operating Points” for more information.

Off

Do not log states data.

Tips

- Specify the format for the logged states data using the **Format** parameter.
- To log fixed-point states data, log states data using the **Dataset** format.
- When you log states data using the **Dataset** format, states data streams to the Simulation Data Inspector during simulation.
- **Dataset** format does not support:
 - Logging states during rapid accelerator simulation.
 - Logging states inside a function-call subsystem.
 - Code generation.
- The states logging variable is empty when you enable states logging for a model that has no states.
- If you log states data in a format other than **Dataset**, you can convert the logged data to **Dataset** format. Converting the data to **Dataset** makes it easier to post process with other logged data. For more information, see “Dataset Conversion for Logged Data”.
- If you log states data in **Structure** with **time** format or in **Array** format while also logging time, select the **Record logged workspace data in Simulation Data Inspector** parameter to view the data in the Simulation Data Inspector after simulation.

Programmatic Use

Parameter: SaveState

Value: 'on' | 'off'

Default: 'off'

Parameter: StateSaveName

Type: character vector

Default: 'xout'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Save Block States and Simulation Operating Points”
- “Comparison of Signal Loading Techniques”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Output

Description

Saves data for signals connected to root-level Outport blocks to the specified MATLAB variable.

Category: Data Import/Export

Settings

Default: On, yout

On

Simulink exports root outport signal data to the MATLAB workspace during simulation.

Specify the name of the MATLAB variable used to store the data. See “Export Simulation Data” for more information.

Off

Simulink does not export root outport signal data during simulation.

Tips

- You must select the **Output** check box before entering a name for the output variable.
- Simulink saves the output to the MATLAB workspace at the base sample rate of the model if you set the **Format** parameter to a value other than **Dataset**. For **Dataset** format, logging uses the rate set for each Outport block.
- The **Additional parameters** area includes parameters for specifying other characteristics of the saved data, including the format and the decimation factor.
- To specify an interval for logging, use the **Logging intervals** parameter.
- To log **Output** data to the Simulation Data Inspector, select **Dataset** format.
- To log fixed-point data, set the **Format** parameter to **Dataset**. If you set the **Format** parameter to a value other than **Dataset**, Simulink logs fixed-point data as double.
- To log bus data, set the **Format** parameter to a value other than **Array**.
- If you use a format other than **Dataset**, you can convert the logged data to **Dataset** format. Converting the data to **Dataset** makes it easier to post-process with other logged data. For more information, see “Dataset Conversion for Logged Data”.
- For the active variant condition, Simulink creates a **Dataset** object with the logged data. For inactive variant conditions, Simulink creates MATLAB **timeseries** with zero samples.
- When you call the `sim` function inside a function, the output logged by the function is in the function workspace. To access that output in the base workspace, add a command such as this after the `sim` command:

```
assignin('base','yout',yout);
```

- Do not use a variable name that is the same as a `Simulink.SimulationOutput` object function name or property name.

- To log data for a variable-size signal connected to a root-level Outport block, use the `Dataset` format. Data for a variable-size signal is always saved as a `timetable` object that contains a cell array of data for each time step.

Programmatic Use

Parameter: SaveOutput

Value: 'on' | 'off'

Default: 'on'

Parameter: OutputSaveName

Type: character vector

Value: any valid value

Default: 'yout'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Data Format for Logged Simulation Data”
- “Export Simulation Data”
- “Model Configuration Parameters: Data Import/Export” on page 3-2
- “Dataset Conversion for Logged Data”

Final states

Description

Saves the logged states of the model at the end of a simulation to the specified MATLAB variable.

Category: Data Import/Export

Settings

Default: Off, `xFinal`

On

Simulink software exports final logged state data to the MATLAB workspace during simulation.

Specify the name of the MATLAB variable in which to store the values of these final states. See [Importing and Exporting States](#) for more information.

Off

Simulink software does not export the final state data during simulation.

Tips

- You must select the **Final states** check box before entering the final states variable.
- Simulink software saves the final states in a MATLAB workspace variable having the specified name.
- The saved data has the format that you specify with the **Format** parameter.
- Simulink creates empty variables for final state logging (`xfinal`) if both of these conditions apply:
 - You enable **Final states**.
 - A model has no states.
- Using the **Final states** is not always sufficient for complete and accurate restoration of a simulation state. The `ModelOperatingPoint` object contains the set of all variables that are related to the simulation of a model. For details, see “Save complete SimState in final state” on page 3-22 and “Use Model Operating Point for Faster Simulation Workflow”.
- See “Save Block States and Simulation Operating Points” for more information.
- If you use a format other than `Dataset`, you can convert the logged data to `Dataset` format. Converting the data to `Dataset` makes it easier to postprocess with other logged data. For more information, see “Dataset Conversion for Logged Data”.

Programmatic Use

Parameter: `SaveFinalState`

Value: 'on' | 'off'

Default: 'off'

Parameter: `FinalStateName`

Type: character vector

Default: 'xFinal'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- Importing and Exporting States
- “Model Configuration Parameters: Data Import/Export” on page 3-2
- “Dataset Conversion for Logged Data”

Format

Description

Specify data format for logging states, output, and final states data.

Category: Data Import/Export

Settings

Default: Dataset

Dataset

Logged states and outputs are each stored in a `Simulink.SimulationData.Dataset` object. Each `Dataset` object contains an element for each individual state or output. The data for each state or output is stored in a `timeseries` object by default except variable-size signal data. Variable-size signal data is always stored in a `timetable` object that contains a cell array of signal values for each time step.

Array

Logged data is stored in a matrix. Each row in the matrix corresponds to a simulation time step, and each column corresponds to a state or output. The order of the states and outputs in the matrix depends on the block sorted order, which can change from one simulation to the next. Do not use Array format to log bus data.

Structure

State and output data each log to a structure. The states structure contains a structure for each block in the model that has a state. The outputs structure contains a structure for each root-level Output block in the model.

Structure with time

The data logs to a structure with a `time` field and a `signals` field. The `time` field contains a vector of simulation times. The `signals` field contains the same data as the `Structure` format.

Tips

- When you log states and output data using `Dataset`:
 - You can work with logged data in MATLAB without a Simulink license.
 - Logging supports saving multiple data values for a given time step, which can be required for logging data in a For Iterator Subsystem, a While Iterator Subsystem, and Stateflow.
 - Logged data automatically streams to the **Simulation Data Inspector** during simulation.
 - You can log variable-size signals using root-level Output blocks.
- `Dataset` format does not support:
 - Code generation.
 - Logging states inside a function-call subsystem.
 - Logging states during rapid accelerator simulations.

- To use Array format, all logged states and outputs must be:

- All scalars or all vectors (or all matrices for states)
- All real or all complex
- The same data type

Use another format if the outputs and states in your model do not meet these conditions.

- The format specified for the **Format** parameter does not apply to final states data.

Programmatic Use

Parameter: SaveFormat

Value: 'Array' | 'Structure' | 'StructureWithTime' | 'Dataset'

Default: 'Dataset'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Export Simulation Data”
- “Data Format for Logged Simulation Data”
- “Time, State, and Output Data Format”
- “Dataset Conversion for Logged Data”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Limit data points to last

Description

Log only the last n data points for outputs and states.

Category: Data Import/Export

Settings

Default: Off, 1000

On

Limits the number of data points logged to the workspace to the specified number. This setting only applies to output and states logging.

Specify the maximum number of data points to log to the workspace. At the end of the simulation, the workspace contains the last n points generated by the simulation.

Off

Does not limit the number of data points logged to the workspace for outputs and states.

Tips

- For some models and simulation conditions, logging can produce large amounts of data. Use this parameter to limit the number of samples saved when you only need to analyze data from the end of the simulation.
- You can also apply a **Decimation** factor to reduce the number of samples saved for outputs and states.
- For more information about configuring which data points are logged for different logging techniques, see “Specify Signal Values to Log”.

Programmatic Use

Parameter: LimitDataPoints

Value: 'on' | 'off'

Default: 'off'

Parameter: MaxDataPoints

Type: character vector

Value: positive integer greater than zero

Default: '1000'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact

Application	Setting
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Export Simulation Data”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Decimation

Description

Specify the decimation factor, n , such that every n th data point is logged for outputs and states.

Category: Data Import/Export

Settings

Default: 1

- With the default value, 1, all data points are saved for logged outputs and states.
- The specified value must be a positive integer greater than zero.
- The value of this parameter is not tunable.
- Simulink outputs data at the specified number of data points. For example, specifying 2 saves every other data point, while specifying 10 saves one in ten data points.

Tips

- For some models and simulation conditions, logging can produce large amounts of data. Use this parameter to limit the number of samples saved when a reduced effective sample rate is sufficient.
- You can also use the **Limit data points to last** parameter to reduce the number of sample values saved for output and states logging.
- For more information about configuring which data points are logged for different logging techniques, see “Specify Signal Values to Log”.

Programmatic Use

Parameter: Decimation

Type: character vector

Value: positive integer greater than zero

Default: '1'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Export Simulation Data”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Save complete SimState in final state

Note **Save complete SimState in final state** is not recommended. Use **Save final operating point** instead.

Description

At the end of a simulation, Simulink saves the complete set of states of the model, including logged states, to the specified MATLAB variable.

Category: Data Import/Export

Settings

Default: Off, xFinal

On

Simulink software exports the complete set of final state data (i.e., the SimState) to the MATLAB workspace during simulation.

Specify the name of the MATLAB variable in which to store the values of the final states. See [Importing and Exporting States](#) for more information.

Off

Simulink software exports the final logged states during simulation.

Tips

- You must select the **Final states** check box to enable the **Save complete SimState in final state** option.
- Simulink saves the final states in a MATLAB workspace variable having the specified name.

Dependencies

This parameter is enabled by **Final states**.

Programmatic Use

Parameter: SaveCompleteFinalSimState

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- Importing and Exporting States
- “Limitations of Saving and Restoring Operating Point”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Save final operating point

Description

At the end of a simulation, Simulink saves the complete set of states of the model, including logged states, to the specified MATLAB variable.

Category: Data Import/Export

Settings

Default: Off, xFinal

On

Simulink software exports the complete set of final state data (i.e., the operating point) to the MATLAB workspace during simulation.

Specify the name of the MATLAB variable in which to store the values of the final states. See Importing and Exporting States for more information.

Off

Simulink software exports the final logged states during simulation.

Tips

- You must select the **Final states** check box to enable the **Save final operating point** option.
- Simulink saves the final states in a MATLAB workspace variable having the specified name.

Dependencies

This parameter is enabled by **Final states**.

Programmatic Use

Parameter: SaveOperatingPoint

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- Importing and Exporting States
- “Limitations of Saving and Restoring Operating Point”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Signal logging

Description

Globally enable or disable signal logging to the workspace for a model.

Category: Data Import/Export

Settings

Default: On, logout

On

Enables logging data for specified signals to the MATLAB workspace and the Simulation Data Inspector.

Specify the name of variable used to store logged signal data in the MATLAB workspace. For more information, see “Specify a Name for Signal Logging Data”.

Off

Disables logging signal data to the MATLAB workspace and the Simulation Data Inspector.

Tips

- You must select the **Signal logging** check box before entering the signal logging variable.
- Simulink saves the signal data in a MATLAB workspace variable with the specified name.
- The saved data is a `Simulink.SimulationData.Dataset` object.
- Data for variable-size signals is saved as a `timetable` object with a cell array of values for each time step.
- Simulink does not support signal logging for the following types of signals:
 - Output of a Function-Call Generator block
 - Signal connected to the input of a Merge block
 - Outputs of Trigger and Enable blocks
- If you select **Signal logging**, you can use the **Configure Signals to Log** button to open the Signal Logging Selector. You can use the Signal Logging Selector to:
 - Review all signals in a model hierarchy that are configured for logging
 - Override signal logging settings for specific signals
 - Control signal logging throughout a model reference hierarchy in a streamlined way

You can use the Signal Logging Selector with Simulink and Stateflow signals.

For details about the Signal Logging Selector, see “View Logging Configuration Using the Signal Logging Selector” and “Override Signal Logging Settings”.

- Do not use a variable name that is the same as a `Simulink.SimulationOutput` object function name or property name.

- For information about logging Simscape™ data, see “About Simulation Data Logging” (Simscape).

Dependencies

This parameter enables the **Configure Signals to Log** button.

Programmatic Use

Parameter: SignalLogging

Value: 'on' | 'off'

Default: 'on'

Parameter: SignalLoggingName

Type: character vector

Value: any valid value

Default: 'logout'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Export Signal Data Using Signal Logging”
- “Dataset Conversion for Logged Data”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Data stores

Description

Globally enable or disable logging of Data Store Memory block variables for this model.

Category: Data Import/Export

Settings

Default: On, dsmsout

On

Enables data store logging to the MATLAB workspace and the Simulation Data Inspector during simulation.

Specify the name of the `Simulink.SimulationData.Dataset` object for the logged data store data.

Off

Disables data store logging to the MATLAB workspace and the Simulation Data Inspector during simulation.

Tips

- Simulink saves the data in a MATLAB workspace variable having the specified name.
- The saved data has the `Simulink.SimulationData.Dataset` format.
- See “Supported Data Types, Dimensions, and Complexity for Logging Data Stores”, “Data Store Logging Limitations”, and “Data Store Logging Limitations” for more information.

Dependencies

Select the **Data stores** check box before entering the data store logging variable.

Programmatic Use

Parameter: DSMLogging

Value: 'on' | 'off'

Default: 'on'

Parameter: DSMLoggingName

Type: character vector

Value: any valid value

Default: 'dsmOut'

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

[Simulink.SimulationData.DataStoreMemory](#) | Data Store Memory

Related Examples

- “Log Data Stores”
- “Export Signal Data Using Signal Logging”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Log Dataset data to file

Description

Log simulation data saved using the `Dataset` format to a MAT-file.

Category: Data Import/Export

Settings

Default: 'off'

On

Enables logging data saved using the `Dataset` format to a MAT-file.

Log simulation data to a MAT-file when you know prior to simulation that you want to save the results in a file.

Specify the path and file name for the MAT-file.

Off

Disables logging simulation data to a MAT-file.

Tips

- To use the **Log Dataset data to file** option, select one or more of these types of data to log:
 - **States**
 - **Final states**
 - **Signal logging**
 - **Output**
 - **Data stores**
 - Stateflow states and data

If you are logging states or output data, set the **Format** parameter to `Dataset`.

If you select the **Final states** parameter, clear the **Save final operating point** parameter.

- When the data in the MAT-file fits into memory, use the `load` function to access the data.
- When the data in the MAT-file is too large to fit into memory, access the data in the MAT-file using `Simulink.SimulationData.DatasetRef` and `matlab.io.datastore.SimulationDatastore` objects.
- Except for parallel simulations, Simulink overwrites the contents of the MAT-file during each simulation unless you change the name of the file between simulations. For details, see “Save Logged Data from Successive Simulations”.

Dependencies

Select the **Log Dataset data to file** check box before entering the path to the MAT-file for logging.

Programmatic Use

Parameter: LoggingToFile

Value: 'on' | 'off'

Default: 'off'

Parameter: LoggingFileName

Value: valid path and file name

Default: 'out.mat'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

[Simulink.SimulationData.DatasetRef](#) | [Simulink.SimulationData.Dataset](#) | [load](#)

Related Examples

- “Log Data to Persistent Storage”
- “Load Big Data for Simulations”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Output options

Description

Select options for generating additional output signal data for variable-step solvers.

Category: Data Import/Export

Settings

Default: Refine output

Refine output

Generates data output between, as well as at, simulation times steps. Use **Refine factor** to specify the number of points to generate between simulation time steps. For more information, see “Refine Output”.

Produce additional output

Generates additional output at specified times. Use **Output times** to specify the simulation times at which Simulink software generates additional output.

Produce specified output only

Use **Output times** to specify the simulation times at which Simulink generates output, in addition to the simulation start and stop times.

Tips

- These settings can force the solver to calculate output values for times that it would otherwise have omitted because the calculations were not needed to achieve accurate simulation results. These extra calculations can cause the solver to locate zero crossings that it would otherwise have missed.
- For additional information on how Simulink software calculates outputs for these three options, see “Samples to Export for Variable-Step Solvers”.
- Do not use a variable name that is the same as a `Simulink.SimulationOutput` object function name or property name.

Dependencies

This parameter is enabled only if the model specifies a variable-step solver (see Solver Type on page 14-8).

Selecting Refine output enables the **Refine factor** parameter.

Selecting Produce additional output or Produce specified output only enables the **Output times** parameter.

Programmatic Use

Parameter: OutputOption

Value: 'RefineOutputTimes' | 'AdditionalOutputTimes' | 'SpecifiedOutputTimes'

Default: 'RefineOutputTimes'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Output Options”
- “Refine factor” on page 3-34
- “Refine Output”
- “Export Simulation Data”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Refine factor

Description

Specify how many points to generate between time steps to refine the output.

Category: Data Import/Export

Settings

Default: 1

- The default refine factor is 1, meaning that no extra data points are generated.
- A refine factor of 2 provides output midway between the time steps, as well as at the steps.

Tip

Simulink software ignores this option for discrete models. This is because the value of data between time steps is undefined for discrete models.

Dependency

This parameter is enabled only if you select `Refine` output as the value of **Output options**.

Programmatic Use

Parameter: `Refine`

Type: character vector

Value: any valid value

Default: `'1'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Refine Output”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Output times

Description

Specify times at which Simulink software should generate output in addition to, or instead of, the times of the simulation steps taken by the solver used to simulate the model.

Category: Data Import/Export

Settings

Default: []

- Enter a matrix containing the times at which Simulink software should generate output in addition to, or instead of, the simulation steps taken by the solver.
- If the value of **Output options** is `Produce additional output`, for the default value [], Simulink generates no additional data points.
- If the value of **Output options** is `Produce specified output only`, for the default value [], Simulink generates no data points.

Tips

- The `Produce additional output` option generates output at the specified times, as well as at the regular simulation steps.
- The `Produce specified output only` option generates output at the specified times.
- Discrete models define outputs only at major time steps. Therefore, Simulink software logs output for discrete models only at major time steps. If the **Output times** field specifies other times, Simulink displays a warning in the MATLAB Command Window.
- For additional information on how Simulink software calculates outputs for the Output options `Produce specified output only` and `Produce additional output` options, see “Samples to Export for Variable-Step Solvers”.

Dependency

This parameter is enabled only if the value of **Output options** is `Produce additional output` or `Produce specified output only`.

Programmatic Use

Parameter: OutputTimes

Type: character vector

Value: any valid value

Default: ' [] '

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Refine Output”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Single simulation output

Description

Specify whether to return simulation data as a single `Simulink.SimulationOutput` object. Simulation data includes simulation metadata and all data logged to the workspace, including outputs, states, data store memory, signals, and data logged to the workspace using blocks.

Category: Data Import/Export

Settings

Default: On, out

On

All simulation data logged to the workspace is returned in the workspace as a single `Simulink.SimulationOutput` object.

Specify the name of the variable used to store the `Simulink.SimulationOutput` object.

Off

Simulation data is returned in one or more variables, depending on model and logging configuration.

Tips

- When you log data using the To File block, the data logs to the specified file and does not appear in the single `Simulink.SimulationOutput` object.
- When you select **Log Dataset data to file**, the data that logs to the MAT file does not appear in the single `Simulink.SimulationOutput` object.
- Use the `who` function for the `Simulink.SimulationOutput` object to view a list of the variables in the object.
- To use the **Logging intervals** parameter, you must select **Single simulation output**.

Programmatic Use

Parameter: `ReturnWorkspaceOutputs`

Value: 'on' | 'off'

Default: 'on'

Parameter: `ReturnWorkspaceOutputsName`

Type: string | character vector

Value: valid MATLAB variable name

Default: 'out'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Objects

`Simulink.SimulationOutput`

Functions

`sim`

Related Examples

- “Model Configuration Parameters: Data Import/Export” on page 3-2
- “Run Simulations Programmatically”

Logging intervals

Description

Specify time intervals during which to log data

Category: Data Import/Export

Settings

Default: `[-inf, inf]`

- Specify a two-column matrix that contains real, **double** values that indicate the start and stop time for an interval in which you want to log data. The matrix can contain any number of rows to specify any number of logging intervals.
- The matrix elements cannot be **NaN**.
- Intervals must be disjoint and ordered. For example, you can specify these three intervals: `[1, 5; 6, 10; 11, 15]`.

Tips

- The logging intervals apply to:
 - Time
 - States
 - Output
 - Signal logging
 - To Workspace blocks
 - To File blocks
 - Data logged to the workspace using a Record block
- The logging intervals do not apply to:
 - Final states data
 - Data logged using scopes
 - Data logged to a file using a Record block
 - Data in the Simulation Data Inspector
- Logging intervals do not affect data displayed in the Record block or using dashboard blocks.
- Logging intervals are only supported when you select **Single simulation output**. The single simulation output is a `Simulink.SimulationOutput` object that contains all logged data that logs to the workspace.
- When you specify an interval that includes a time before the simulation start time or after the simulation stop time, no data is logged for that interval.
- The **Decimation** and **Limit data points to last** parameters also apply to logged data when you specify logging intervals.

- When you change the logging intervals while using the Simulation Stepper to step through a simulation, the new logging intervals do not apply until you step forward.
- SIL simulation mode supports logging intervals for data that logs to a `Simulink.SimulationOutput` object. In SIL mode, specified logging intervals are ignored without warning for:
 - Data logged using a To File block
 - MAT file logging (enabled with the **MAT-file logging** configuration parameter)
- PIL simulation mode does not support logging intervals. Specified logging intervals are ignored without a warning message.

Dependencies

To enable the **Logging intervals** parameter, select **Single simulation output**.

Programmatic Use

Parameter: `LoggingIntervals`

Type: two-column matrix with real, double values

Default: `[-inf, inf]`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Model Settings

Single simulation output | **Decimation** | **Limit data points**

Blocks

To File | Record, XY Graph | To Workspace

Related Examples

- “Specify Signal Values to Log”
- “Run Simulations Programmatically”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Record logged workspace data in Simulation Data Inspector

Description

Specify whether to send data logged in a format other than `Dataset` and data logged using blocks to the Simulation Data Inspector after simulation pauses or completes.

Category: Data Import/Export

Settings

Default: Off

On

Record these kinds of data to display in the Simulation Data Inspector after a simulation pauses or completes:

- States and output data logged in `Array` or `Structure` with time formats
- Data logged using `To Workspace` blocks, `To File` blocks, or `Scope` blocks

This setting adds a recording icon to the **Simulation Data Inspector** button.

Off

Do not record data logged using blocks or in a format other than `Dataset` to view in the Simulation Data Inspector.

Tips

- When you log data using the `Array` format, you must also log time for the states and output data to log to the Simulation Data Inspector.
- To open the Simulation Data Inspector, click the **Simulation Data Inspector** button.

Programmatic Use

Parameter: `InspectSignalLogs`

Value: `'on'` | `'off'`

Default: `'off'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Simulation Data Inspector

Related Examples

- “Load Signal Data for Simulation”
- “View Data in the Simulation Data Inspector”
- “Inspect Simulation Data”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Diagnostics Parameters: Compatibility

Model Configuration Parameters: Compatibility Diagnostics

The **Diagnostics > Compatibility** category includes parameters for detecting issues when you use a model that you created in an earlier release.

Parameter	Description
“S-function upgrades needed” on page 4-4	Select the diagnostic action to take if Simulink software encounters a block that has not been upgraded to use features of the current release.
“Block behavior depends on frame status of signal” on page 4-5	Select the diagnostic action to take when Simulink software encounters a block whose behavior depends on the frame status of a signal.
“Operating point object from a different release” on page 4-7	Use this check to report that a <code>Simulink.op.ModelOperatingPoint</code> object was generated by an earlier version of Simulink.

See Also

Related Examples

- Diagnosing Simulation Errors
- Solver Diagnostics on page 9-2
- Sample Time Diagnostics on page 8-2
- Data Validity Diagnostics on page 6-2
- Type Conversion Diagnostics on page 11-2
- Connectivity Diagnostics on page 5-2
- Compatibility Diagnostics on page 4-2
- “Model Configuration Parameters: Model Referencing Diagnostics” on page 7-2

Compatibility Diagnostics Overview

Configuration

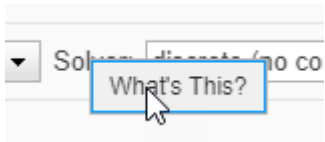
Set the parameters displayed.

Tips

- To open the **Compatibility** pane, in the Simulink Editor, in the **Modeling** tab, click **Model Settings**, then select **Diagnostics > Compatibility**.
- The options are typically to do nothing or to display a warning or an error message.
- A warning does not terminate a simulation, but an error does.

To get help on an option

- 1 Right-click the option text label.
- 2 From the context menu, select **What's This**.



See Also

Related Examples

- "Model Configuration Parameters: Compatibility Diagnostics" on page 4-2

S-function upgrades needed

Description

Select the diagnostic action to take if Simulink software encounters a block that has not been upgraded to use features of the current release.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Command-Line Information

Parameter: SFcnCompatibilityMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Compatibility Diagnostics” on page 4-2

Block behavior depends on frame status of signal

Description

Select the diagnostic action to take when Simulink software encounters a block whose behavior depends on the frame status of a signal.

In future releases, frame status will no longer be a signal attribute. To prepare for this change, many blocks received a new parameter. This parameter allows you to specify whether the block treats its input as frames of data or as samples of data. Setting this parameter prepares your model for future releases by moving control of sample- and frame-based processing from the frame status of the signal to the block.

This diagnostic helps you identify whether any of the blocks in your model relies on the frame status of a signal. By knowing this status, you can determine whether the block performs sample- or frame-based processing. For more information, see the R2012a DSP System Toolbox™ Release Notes section about frame-based processing.

Note Frame-based processing requires a DSP System Toolbox license.

Category: Diagnostics

Settings

Default: error

none

Simulink software takes no action.

warning

If your model contains any blocks whose behavior depends on the frame status of a signal, Simulink software displays a warning.

error

If your model contains any blocks whose behavior depends on the frame status of a signal, Simulink software terminates the simulation and displays an error message.

Tips

- Use the Upgrade Advisor to automatically update the blocks in your model. See “Model Upgrades”.

Command-Line Information

Parameter: FrameProcessingCompatibilityMsg

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Sample- and Frame-Based Concepts” (DSP System Toolbox)
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Compatibility Diagnostics” on page 4-2

Operating point object from a different release

Description

Use this check to report that the `Simulink.op.ModelOperatingPoint` object specified using the **Initial state** parameter was generated by a different version of Simulink.

Category: Diagnostics

Settings

Default: error

warning

Simulink restores as much of the model operating point as possible.

error

When Simulink detects that the `ModelOperatingPoint` object was generated by an earlier version of Simulink, it does not load the object to restore the model operating point.

Command-Line Information

Parameter: `NonCurrentReleaseOperatingPointMsg`

Value: 'warning' | 'error'

Default: 'error'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Use Model Operating Point for Faster Simulation Workflow”
- “Model Configuration Parameters: Compatibility Diagnostics” on page 4-2

Diagnostics Parameters: Connectivity

Model Configuration Parameters: Connectivity Diagnostics

The **Diagnostics > Connectivity** category includes parameters for detecting issues related to signal line connectivity, for example, unconnected ports and lines.

On the Configuration Parameters dialog box, the following configuration parameters are on the **Connectivity** pane.

Parameter	Description
“Signal label mismatch” on page 5-4	Select the diagnostic action to take when different names are used for the same signal as that signal propagates through blocks in a model. This diagnostic does not check for signal label mismatches on a virtual bus signal.
“Unconnected block input ports” on page 5-5	Select the diagnostic action to take when the model contains a block with an unconnected input.
“Unconnected block output ports” on page 5-6	Select the diagnostic action to take when the model contains a block with an unconnected output.
“Unconnected line” on page 5-7	Select the diagnostic action to take when the Model contains an unconnected line or an unmatched Goto or From block.
“Unspecified bus object at root Output block” on page 5-8	Select the diagnostic action to take while generating a simulation target for a referenced model if any of the model's root Output blocks is connected to a bus but does not specify a bus object (see <code>Simulink.Bus</code>).
“Element name mismatch” on page 5-10	Select the diagnostic action to take if the name of a bus element does not match the name specified by the corresponding bus object.
“Bus signal treated as vector” on page 5-12	Select the diagnostic action to take when Simulink software detects a virtual bus signal that is used as a mux signal.
“Non-bus signals treated as bus signals” on page 5-14	Detect 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.
“Repair bus selections” on page 5-16	Repair broken selections in the Bus Selector and Bus Assignment block parameter dialogs due to upstream bus hierarchy changes.
“Context-dependent inputs” on page 5-17	Select the diagnostic action to take when Simulink software has to compute any of a function-call subsystem's inputs directly or indirectly during execution of a call to a function-call subsystem.

See Also

Related Examples

- Diagnosing Simulation Errors
- Solver Diagnostics on page 9-2
- Sample Time Diagnostics on page 8-2
- Data Validity Diagnostics on page 6-2
- Type Conversion Diagnostics on page 11-2
- Compatibility Diagnostics on page 4-2
- Model Referencing Diagnostics on page 7-2

Signal label mismatch

Description

Select the diagnostic action to take when different names are used for the same signal as that signal propagates through blocks in a model. This diagnostic does not check for signal label mismatches on a virtual bus signal.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Command-Line Information

Parameter: SignalLabelMismatchMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Signal Names and Labels”
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Connectivity Diagnostics” on page 5-2

Unconnected block input ports

Description

Select the diagnostic action to take when the model contains a block with an unconnected input.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Command-Line Information

Parameter: UnconnectedInputMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Connectivity Diagnostics” on page 5-2

Unconnected block output ports

Description

Select the diagnostic action to take when the model contains a block with an unconnected output.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Command-Line Information

Parameter: UnconnectedOutputMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Connectivity Diagnostics” on page 5-2

Disconnected line

Description

Select the diagnostic action to take when the Model contains an unconnected line or an unmatched Goto or From block.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Command-Line Information

Parameter: UnconnectedLineMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- Goto
- From
- “Model Configuration Parameters: Connectivity Diagnostics” on page 5-2

Unspecified bus object at root Output block

Description

Select the diagnostic action to take when generating a simulation target for a referenced model if any root Output block of the referenced model receives a bus and does not specify a `Simulink.Bus` object.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- This diagnostic applies only when a model is used as a referenced model. Simulating or updating the model on its own does not invoke the diagnostic.
- A root Out Bus Element block does not require a Bus object for the corresponding Model block to output a bus.
- When a root Output block receives a virtual bus and does not specify a Bus object, the corresponding Model block outputs a vector.

Command-Line Information

Parameter: RootOutputRequireBusObject

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Functions

Simulink.Bus

Related Examples

- Diagnosing Simulation Errors
- Output
- “Model Configuration Parameters: Connectivity Diagnostics” on page 5-2

Element name mismatch

Description

Select the diagnostic action to take if the name of a bus element does not match the name specified by the corresponding bus object.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- You can use this diagnostic along with bus objects to ensure that your model meets bus element naming requirements imposed by some blocks, such as the Switch block.
- With a Bus Creator block, you can enforce strong data typing by using the **Use names from inputs instead of from bus object** block parameter.

Command-Line Information

Parameter: BusObjectLabelMismatch

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Connectivity Diagnostics” on page 5-2

Bus signal treated as vector

Description

Select the diagnostic action to take when Simulink software detects a virtual bus signal treated as a vector signal.

Category: Diagnostics

Settings

Default: none

none

Disables checking for virtual bus signals treated as vector signals.

warning

Simulink displays a warning if it detects a virtual bus signal treated as a vector signal.

error

Simulink terminates the simulation and displays an error message when it builds a model that uses a virtual bus signal treated as a vector signal.

Tips

- The diagnostic considers a virtual bus signal to be treated as a vector signal if the signal is input to a block that does not accept virtual bus signals. See “Bus-Capable Blocks” for details.
- Virtual buses can be treated as vector signals only when all constituent signals have the same attributes.
- You can identify bus signals that are treated as a vectors using the Model Advisor “**Check bus signals treated as vectors**” check.

Command-Line Information

Parameter: StrictBusMsg

Value: 'ErrorLevel1' | 'WarnOnBusTreatedAsVector' | 'ErrorOnBusTreatedAsVector'

Default: 'ErrorLevel1'

Here is how the StrictBusMsg parameter values map to the values of the **Bus signal treated as vector** parameter in the **Configuration Parameters > Diagnostics > Connectivity** dialog box.

Value of StrictBusMsg	Value of “Bus signal treated as vector” diagnostic
ErrorLevel1	none
WarnOnBusTreatedAsVector	warning
ErrorOnBusTreatedAsVector	error

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Functions

`Simulink.BlockDiagram.addBusToVector`

Related Examples

- Diagnosing Simulation Errors
- “Bus-Capable Blocks”
- Demux
- Bus to Vector
- “Underspecified initialization detection” on page 2-65
- “Check virtual bus inputs to blocks”
- “Model Configuration Parameters: Connectivity Diagnostics” on page 5-2

Non-bus signals treated as bus signals

Description

Detect 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.

Category: Diagnostics

Settings

Default: none

none

Implicitly converts non-bus signals to bus signals to support connecting the signal to a Bus Assignment or Bus Selector block.

warning

Simulink displays a warning, indicating that it has converted a non-bus signal to a bus signal. The warning lists the non-bus signals that Simulink converts.

error

Simulink terminates the simulation without converting non-bus signals to bus signals. The error message lists the non-bus signal that is being treated as a bus signal.

Command-Line Information

Parameter: NonBusSignalsTreatedAsBus

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Functions

`Simulink.BlockDiagram.addBusToVector`

Related Examples

- Diagnosing Simulation Errors
- “Bus-Capable Blocks”

- Demux
- Bus to Vector
- “Model Configuration Parameters: Connectivity Diagnostics” on page 5-2

Repair bus selections

Description

Repair broken selections in the Bus Selector and Bus Assignment block parameter dialogs due to upstream bus hierarchy changes.

Category: Diagnostics

Settings

Default: Warn and repair

Warn and repair

Simulink displays a warning, indicating the block parameters for Bus Selector and Bus Assignment blocks that Simulink repaired to reflect upstream bus hierarchy changes.

Error without repair

Simulink terminates the simulation and displays an error message indicating the block parameters that you need to repair for Bus Selector and Bus Assignment blocks to reflect upstream bus hierarchy changes.

Command-Line Information

Parameter: BusNameAdapt

Values: 'WarnAndRepair' | 'ErrorWithoutRepair'

Default: 'WarnAndRepair'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	Warn and repair

See Also

Related Examples

- “Resolve Circular Dependencies in Buses”
- Diagnosing Simulation Errors
- “Bus-Capable Blocks”
- “Model Configuration Parameters: Connectivity Diagnostics” on page 5-2
- “Model Configuration Parameters: Connectivity Diagnostics” on page 5-2

Context-dependent inputs

Description

Select the diagnostic action to take when Simulink has to compute any function-call subsystem inputs directly or indirectly during execution of a call to a function-call subsystem.

Category: Diagnostics

Settings

Default: error

error

Issue an error for context-dependent inputs.

warning

Issue a warning for context-dependent inputs.

Tips

- This situation occurs when executing a function-call subsystem that can change its inputs.
- For examples of function-call subsystems, see “Simulink Subsystem Semantics”.
- To fix an error or warning generated by this diagnostic, use *one* of these approaches:
 - For the Inport block inside of the function-call subsystem, enable the **Latch input for feedback signals of function-call subsystem outputs** parameter.
 - Place a Function-Call Feedback Latch block on the feedback signal.

For examples of using these approaches, open the `sl_subsys_fcncallerr12` model and press the **more info** button.

Command-Line Information

Parameter: `FcnCallInpInsideContextMsg`

Value: 'Error'|'Warning'

Default: 'Error'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	Error

See Also

Related Examples

- “Using Function-Call Subsystems”
- “Pass fixed-size scalar root inputs by value for code generation” on page 12-18
- Subsystem Semantics
- Subsystem
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Connectivity Diagnostics” on page 5-2

Diagnostics Parameters: Data Validity

Model Configuration Parameters: Data Validity Diagnostics

The **Diagnostics > Data Validity** category includes parameters for detecting issues related to data (signals, parameters, and states). These issues include:

- Loss of information due to data type quantization and overflow.
- Loss of parameter tunability in the generated code.
- Loss of information due to Data Store Write and Data Store Read block ordering.

On the Configuration Parameters dialog box, the following configuration parameters are on the **Data Validity** pane.

Parameter	Description
"Signal resolution" on page 6-6	Select how Simulink software resolves signals and states to <code>Simulink.Signal</code> objects.
"Division by singular matrix" on page 6-8	Select the diagnostic action to take if the Product, Matrix Multiply block detects a singular matrix while inverting one of its inputs in matrix multiplication mode.
"Underspecified data types" on page 6-10	Select the diagnostic action to take if Simulink software could not infer the data type of a signal during data type propagation.
"Simulation range checking" on page 6-12	Select the diagnostic action to take when signals exceed specified minimum or maximum values.
"String truncation checking" on page 6-14	Select the diagnostic action to take if the string signal is truncated.
"Wrap on overflow" on page 6-15	Select the diagnostic action to take if the value of a signal overflows the signal data type and wraps around.
"Underspecified dimensions" on page 6-19	Select the diagnostic action to take if Simulink software could not infer the signal dimension at compile time.
"Saturate on overflow" on page 6-17	Select the diagnostic action to take if the value of a signal is too large to be represented by the signal data type, resulting in a saturation.
"Inf or NaN block output" on page 6-20	Select the diagnostic action to take if the value of a block output is <code>Inf</code> or <code>NaN</code> at the current time step.
"rt" prefix for identifiers" on page 6-22	Select the diagnostic action to take during code generation if a Simulink object name (the name of a parameter, block, or signal) begins with <code>rt</code> .
"Detect downcast" on page 6-24	Select the diagnostic action to take when a parameter downcast occurs during code generation.

Parameter	Description
"Detect overflow" on page 6-26	Select the diagnostic action to take if a parameter overflow occurs during simulation.
"Detect underflow" on page 6-28	Select the diagnostic action to take when a parameter underflow occurs during simulation.
"Detect precision loss" on page 6-30	Select the diagnostic action to take when parameter precision loss occurs during simulation.
"Detect loss of tunability" on page 6-32	Select the diagnostic action to take when an expression with tunable variables is reduced to its numerical equivalent in the generated code.
"Detect read before write" on page 6-34	Select the diagnostic action to take if the model attempts to read data from a data store to which it has not written data in this time step.
"Detect write after read" on page 6-36	Select the diagnostic action to take if the model attempts to write data to a data store after previously reading data from it in the current time step.
"Detect write after write" on page 6-38	Select the diagnostic action to take if the model attempts to write data to a data store twice in succession in the current time step.
"Multitask data store" on page 6-40	Select the diagnostic action to take when one task reads data from a Data Store Memory block to which another task writes data.
"Duplicate data store names" on page 6-42	Select the diagnostic action to take when the model contains multiple data stores that have the same name. The data stores can be defined with Data Store Memory blocks or Simulink.Signal objects.

These configuration parameters are in the **Advanced parameters** section.

Parameter	Description
"Array bounds exceeded" on page 2-56	Ensure that Simulink-allocated memory used in S-functions does not write beyond its assigned array bounds when writing to its outputs, states, or work vectors.
"Model Verification block enabling" on page 2-58	Enable model verification blocks in the current model either globally or locally.
"Detect multiple driving blocks executing at the same time step" on page 2-63	Select the diagnostic action to take when the software detects a Merge block with more than one driving block executing at the same time step.

Parameter	Description
"Underspecified initialization detection" on page 2-65	Select how Simulink software handles initialization of initial conditions for conditionally executed subsystems, Merge blocks, subsystem elapsed time, and Discrete-Time Integrator blocks.
"Detect ambiguous custom storage class final values" on page 2-115	Detect if a signal using a Reusable custom storage class does not have a unique endpoint. The run-time environment should not read the variable because its value is ambiguous.
"Detect non-reused custom storage classes" on page 2-117	Detect if a signal uses a Reusable custom storage class that the code generator cannot reuse with other uses of the same Reusable custom storage class. If the code generator cannot implement reuse, the generated code will likely contain additional global variables.

See Also

Related Examples

- Diagnosing Simulation Errors
- "Data Types Supported by Simulink"
- Solver Diagnostics on page 9-2
- Sample Time Diagnostics on page 8-2
- Type Conversion Diagnostics on page 11-2
- Connectivity Diagnostics on page 5-2
- Compatibility Diagnostics on page 4-2
- Model Referencing Diagnostics on page 7-2

Data Validity Diagnostics Overview

Configuration

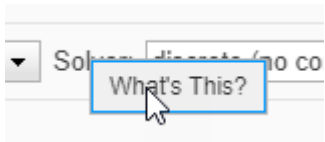
Set the parameters displayed.

Tips

- To open the **Data Validity** pane, in the Simulink Editor, in the **Modeling** tab, click **Model Settings**, then select **Diagnostics > Data Validity**.
- The options are typically to do nothing or to display a warning or an error message.
- A warning does not terminate a simulation, but an error does.

To get help on an option

- 1 Right-click the option text label.
- 2 From the context menu, select **What's This**.



See Also

Related Examples

- "Model Configuration Parameters: Data Validity Diagnostics" on page 6-2

Signal resolution

Description

Select how a model resolves signals and states to `Simulink.Signal` objects. See “Explicit and Implicit Symbol Resolution” for more information.

Category: Diagnostics

Settings

Default: Explicit only

None

Do not perform signal resolution. None of the signals, states, Stateflow data, and MATLAB Function block data in the model can resolve to `Simulink.Signal` objects.

This setting does not affect data stores that you define by creating `Simulink.Signal` objects (instead of using Data Store Memory blocks).

Explicit only

Do not perform implicit signal resolution. Only explicitly specified signal resolution occurs. This is the recommended setting.

Explicit and implicit

Perform implicit signal resolution wherever possible, without posting any warnings about the implicit resolutions.

Explicit and warn implicit

Perform implicit signal resolution wherever possible, posting a warning of each implicit resolution that occurs.

Tips

- To reduce the dependency of the model on variables and objects in workspaces and data dictionaries, which can improve model portability, readability, and ease of maintenance, use **None**.

When you use this setting, migrate design attributes from existing `Simulink.Signal` objects into the model by using block parameters and signal properties (for example, in the Model Data Editor or in Signal Properties dialog boxes).

- Use the Signal Properties dialog box to specify explicit resolution for signals. For more information, see **Signal Properties**.
- Use the **State Attributes** pane on dialog boxes of blocks that have discrete states, e.g., the Discrete-Time Integrator block, to specify explicit resolution for discrete states.
- Multiple signals can resolve to the same signal object and have the properties that the object specifies. However, the signal object cannot use a storage class other than `Auto` or `Reusable`.
- 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.

Command-Line Information

Parameter: `SignalResolutionControl`

Value: `'None' | 'UseLocalSettings' | 'TryResolveAll' | 'TryResolveAllWithWarning'`

Default: `'UseLocalSettings'`

SignalResolutionControl Value	Equivalent Signal Resolution Value
<code>'None'</code>	None
<code>'UseLocalSettings'</code>	Explicit only
<code>'TryResolveAll'</code>	Explicit and implicit
<code>'TryResolveAllWithWarning'</code>	Explicit and warn implicit

Recommended Settings

Application	Setting
Debugging	Explicit only or None
Traceability	Explicit only or None
Efficiency	Explicit only or None
Safety precaution	Explicit only

See Also

Objects

`Simulink.Signal`

Tools

Signal Properties

Related Examples

- Diagnosing Simulation Errors
- Discrete-Time Integrator
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Division by singular matrix

Description

Select the diagnostic action to take if the Product, Matrix Multiply block detects a singular matrix while inverting one of its inputs in matrix multiplication mode.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

For models referenced in Accelerator mode, Simulink ignores the **Division by singular matrix** parameter setting if you set it to a value other than None.

You can use the Model Advisor to identify referenced models for which Simulink changes configuration parameter settings during accelerated simulation.

- 1 In the Simulink Editor, in the **Modeling** tab, click **Model Advisor**, then click **OK**.
- 2 Select **By Task**.
- 3 Run the **Check diagnostic settings ignored during accelerated model reference simulation** check.

Command-Line Information

Parameter: CheckMatrixSingularityMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- Product, Matrix Multiply
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Underspecified data types

Description

Select the diagnostic action to take if Simulink software could not infer the data type of a signal during data type propagation.

Category: Diagnostics

Identify and Resolve Underspecified Data Types

This example shows how to use the configuration parameter **Underspecified data types** to identify and resolve an underspecified data type.

- 1 Open the example model `ex_underspecified_data_types`.
- 2 Set the **Underspecified data types** configuration parameter to `warning`.
- 3 Update the diagram.

The signals in the model use the data type `uint8`, and the model generates a warning.

- 4 Open the Diagnostic Viewer. The warning indicates that the output signal of the Constant block has an underspecified data type.
- 5 Open the Constant block dialog box.

On the **Signal Attributes** tab, **Output data type** is set to `Inherit: Inherit via back propagation`. The Constant block output inherits a data type from the destination block. In this case, the destination is the Sum block.

- 6 Open the Sum block dialog box.

On the **Signal Attributes** tab, **Accumulator data type** is set to `Inherit: Inherit via internal rule`. Sum blocks cast all of their input signals to the selected accumulator data type. In this case, the accumulator data type is specified as an inherited type.

- 7 Open the Inport block dialog box. On the **Signal Attributes** tab, **Data type** is set to `uint8`.

The data type of the Constant block output signal is underspecified because the source and destination blocks each apply an inherited data type. The signal cannot identify an explicit data type to inherit. In cases like this, Simulink applies heuristic rules to select a data type to use.

To resolve the underspecified data type, you can use one of these techniques:

- On the **Signal Attributes** tab of the Constant block dialog box, specify **Output data type** as a particular numeric type, such as `uint8`.
- On the **Signal Attributes** tab of the Sum block dialog box, select the check box **Require all inputs to have the same data type**.

With this setting, the Sum block applies the data type of the first input, `uint8`, to the underspecified data type of the second input.

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Command-Line Information

Parameter: UnderSpecifiedDataTypeMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Default for underspecified data type” on page 22-4
- Diagnosing Simulation Errors
- “Use single Data Type as Default for Underspecified Types” (Embedded Coder)
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Simulation range checking

Description

Select the diagnostic action to take when signals exceed specified minimum or maximum values.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- For information about specifying minimum and maximum values for signals and about how Simulink checks nondouble signals, see “Specify Signal Ranges”.
- For referenced models, Simulink performs signal range checking for only root-level I/O signals. It does not check internal signals.
- If you have an Embedded Coder license, you can perform signal range checking in top-model or Model block software-in-the-loop (SIL) and processor-in-the-loop (PIL) simulations (Embedded Coder).

Command-Line Information

Parameter: SignalRangeChecking

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	warning or error
Traceability	warning or error
Efficiency	none
Safety precaution	error

See Also

Related Examples

- “Specify Signal Ranges”
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

String truncation checking

Description

Select the diagnostic action to take if the string signal is truncated.

Category: Diagnostics

Settings

Default: error

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Command-Line Information

Parameter:StringTruncationChecking

Value: 'none' | 'warning' | 'error'

Default: 'error'

Recommended Settings

Application	Setting
Debugging	error
Traceability	No impact
Efficiency	No impact
Safety precaution	error

Wrap on overflow

Description

Select the diagnostic action to take if the value of a signal overflows the signal data type and wraps around.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation or code generation and displays an error message.

Tips

- This diagnostic applies only to overflows which wrap for integer and fixed-point data types.
- This diagnostic also reports division by zero for all data types, including floating-point data types.
- To check for floating-point overflows (for example, `Inf` or `NaN`) for `double` or `single` data types, select the **Inf or NaN block output** diagnostic. (See “Inf or NaN block output” on page 6-20 for more information.)
- If a floating-point to integer or floating-point to fixed-point overflow is signaled, set the model parameter `EfficientFloat2IntCast` to `'off'` to ensure that simulation and the generated code agree. See Remove code from floating-point to integer conversions that wraps out-of-range values (Simulink Coder) for more detail.
- For models referenced in accelerator mode, Simulink ignores the **Wrap on overflow** parameter setting if you set it to a value other than `None`.

You can use the Model Advisor to identify referenced models for which Simulink changes configuration parameter settings during accelerated simulation.

- 1 In the Simulink Editor, in the **Modeling** tab, click **Model Advisor**, then click **OK**.
 - 2 Select **By Task**.
 - 3 Run the **Check diagnostic settings ignored during accelerated model reference simulation** check.
- During code generation, Simulink may simulate a few blocks in the model for optimization purposes. If simulation of these blocks triggers this diagnostic to report an error, the software terminates code generation.

Command-Line Information

Parameter: IntegerOverflowMsg

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	warning
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Handle Overflows in Simulink Models” (Fixed-Point Designer)
- Diagnosing Simulation Errors
- “Local and Global Data Stores”
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Saturate on overflow

Description

Select the diagnostic action to take if the value of a signal is too large to be represented by the signal data type, resulting in a saturation.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation or code generation and displays an error message.

Tips

- This diagnostic applies only to overflows which saturate for integer and fixed-point data types.
- To check for floating-point overflows (for example, Inf or NaN) for double or single data types, select the **Inf or NaN block output** diagnostic. (See “Inf or NaN block output” on page 6-20 for more information.)
- During code generation, Simulink may simulate a few blocks in the model for optimization purposes. If simulation of these blocks triggers this diagnostic to report an error, the software terminates code generation.

Command-Line Information

Parameter: IntegerSaturationMsg

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	warning
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Handle Overflows in Simulink Models” (Fixed-Point Designer)
- Diagnosing Simulation Errors
- “Local and Global Data Stores”
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Underspecified dimensions

Description

Select the diagnostic action to take if Simulink software could not infer the signal dimension at compile time.

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Command-Line Information

Parameter: UnderSpecifiedDimensionMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

Inf or NaN block output

Description

Select the diagnostic action to take if the value of a block output is `Inf` or `NaN` at the current time step.

Note Accelerator mode does not support any runtime diagnostics.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- This diagnostic applies only to floating-point overflows for `double` or `single` data types.
- To check for integer and fixed-point overflows, select the **Wrap on overflow** diagnostic. (See “Wrap on overflow” on page 6-15 for more information.)
- For models referenced in accelerator mode, Simulink ignores the **Info or NaN block output** parameter setting if you set it to a value other than `None`.

You can use the Model Advisor to identify referenced models for which Simulink changes configuration parameter settings during accelerated simulation.

- 1 In the Simulink Editor, in the **Modeling** tab, click **Model Advisor**, then click **OK**.
- 2 Select **By Task**.
- 3 Run the **Check diagnostic settings ignored during accelerated model reference simulation** check.

Command-Line Information

Parameter: `SignalInfNanChecking`

Value: `'none' | 'warning' | 'error'`

Default: `'none'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Validate a Floating-Point Embedded Model”
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

"rt" prefix for identifiers

Description

Select the diagnostic action to take during code generation if a Simulink object name (the name of a parameter, block, or signal) begins with `rt`.

Category: Diagnostics

Settings

Default: error

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- The default setting (`error`) causes code generation to terminate with an error if it encounters a Simulink object name (parameter, block, or signal), that begins with `rt`.
- This is intended to prevent inadvertent clashes with generated identifiers whose names begins with `rt`.

Command-Line Information

Parameter: RTPrefix

Value: 'none' | 'warning' | 'error'

Default: 'error'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Detect downcast

Description

Select the diagnostic action to take when a parameter downcast occurs during code generation.

Category: Diagnostics

Settings

Default: error

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates code generation and displays an error message.

Tips

- A parameter downcast occurs if the computation of block output required converting the parameter's specified type to a type having a smaller range of values (for example, from `uint32` to `uint8`).
- This diagnostic applies only to named tunable parameters (parameters with a non-Auto storage class).

Command-Line Information

Parameter: ParameterDowncastMsg

Value: 'none' | 'warning' | 'error'

Default: 'error'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Detect overflow

Description

Select the diagnostic action to take if a parameter overflow occurs during simulation.

Category: Diagnostics

Settings

Default: error

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- A parameter overflow occurs if Simulink software encounters a parameter whose data type's range is not large enough to accommodate the parameter's ideal value (the ideal value is either too large or too small to be represented by the data type). For example, suppose that the parameter's ideal value is 200 and its data type is `int8`. Overflow occurs in this case because the maximum value that `int8` can represent is 127.
- Parameter overflow differs from parameter precision loss, which occurs when the ideal parameter value is within the range of the data type and scaling being used, but cannot be represented exactly.
- Both parameter overflow and precision loss are quantization errors, and the distinction between them can be a fine one. The **Detect overflow** diagnostic reports all quantization errors greater than one bit. For very small parameter quantization errors, precision loss will be reported rather than an overflow when

$$(Max + Slope) \geq V_{ideal} > (Min - Slope)$$

where

- *Max* is the maximum value representable by the parameter data type
- *Min* is the minimum value representable by the parameter data type
- *Slope* is the slope of the parameter data type (slope = 1 for integers)
- V_{ideal} is the ideal value of the parameter

Command-Line Information

Parameter: ParameterOverflowMsg

Value: 'none' | 'warning' | 'error'

Default: 'error'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Detect underflow

Description

Select the diagnostic action to take when parameter quantization causes a non-zero value to underflow to zero during simulation.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- Parameter underflow occurs when Simulink software encounters a parameter whose data type does not have enough precision to represent the parameter's ideal value because the ideal value is too small.
- When parameter underflow occurs, casting the ideal non-zero value to the parameter's data type causes the modeled value to become zero.
- Parameter underflow can occur for any data type, including floating-point, fixed-point, and integer data types. For example, the ideal value $1e-46$ will quantize to zero for single-precision, half-precision, all integer types, and most commonly used fixed-point types.
- The absolute quantization error will be small relative to the precision of the data type, but the relative quantization error will be 100%. Depending on how the parameter is used in your algorithm, the effects of underflow will be significant. For example, if the parameter is directly used in multiplication or division, then the impact of a 100% relative quantization error can be significant.

Command-Line Information

Parameter: ParameterUnderflowMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Detect precision loss

Description

Select the diagnostic action to take when parameter precision loss occurs during simulation.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- Precision loss occurs when Simulink software encounters a parameter whose data type does not have enough precision to represent the parameter's value exactly. As a result, the modeled value differs from the ideal value.
- Parameter precision loss differs from parameter overflow, which occurs when the range of the parameter's data type, i.e., that maximum value that it can represent, is smaller than the ideal value of the parameter.
- Both parameter overflow and precision loss are quantization errors, and the distinction between them can be a fine one. The **Detect Parameter overflow** diagnostic reports all parameter quantization errors greater than one bit. For very small parameter quantization errors, precision loss will be reported rather than an overflow when

$$(Max + Slope) \geq V_{ideal} > (Min - Slope)$$

where

- *Max* is the maximum value representable by the parameter data type.
- *Min* is the minimum value representable by the parameter data type.
- *Slope* is the slope of the parameter data type (slope = 1 for integers).
- *V_{ideal}* is the full-precision, ideal value of the parameter.

Command-Line Information

Parameter: ParameterPrecisionLossMsg

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Detect loss of tunability

Description

Select the diagnostic action to take when an expression with tunable variables is reduced to its numerical equivalent in the generated code.

Category: Diagnostics

Settings

Default: warning for GRT targets | error for ERT targets

none

Take no action.

warning

Generate a warning.

error

Terminate simulation or code generation and generate an error.

Tips

- This diagnostic applies only to named tunable parameters (parameters with a non-Auto storage class).
- The default value for **Detect loss of tunability** for ERT-based targets is **error**. When you switch from a system target file that is not ERT-based to one that is ERT-based, **Detect loss of tunability** is set to **error**. However, you can change the setting of **Detect loss of tunability** later.
- If a tunable workspace variable is modified by Mask Initialization code, or is used in an arithmetic expression with unsupported operators or functions, the expression is reduced to its numeric value and therefore cannot be tuned.

Command-Line Information

Parameter: ParameterTunabilityLossMsg

Type: character vector

Value: 'none' | 'warning' | 'error'

Default: 'warning' for GRT targets | 'error' for ERT targets

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- “Tunable Expression Limitations” (Simulink Coder)
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Detect read before write

Description

Select the diagnostic action to take if the model attempts to read data from a data store to which it has not written data in this time step.

Category: Diagnostics

Settings

Default: Use local settings

Use local settings

For each local data store (defined by a Data Store Memory block or Simulink.Signal object in a model workspace) use the setting specified by the block. For each global data store (defined by a Simulink.Signal object in the base workspace) disable the diagnostic.

Disable all

Disables this diagnostic for all data stores accessed by the model.

Enable all as warnings

Displays diagnostic as a warning at the MATLAB command line.

Enable all as errors

Halts the simulation and displays the diagnostic in an error dialog box.

Note During model referencing simulation in accelerator and rapid accelerator mode, if the **Detect read before write** parameter is set to **Enable all as warnings**, **Enable all as errors**, or **Use local settings**, Simulink temporarily changes the setting to **Disable all**.

You can use the Model Advisor to identify referenced models for which Simulink changes configuration this parameter setting during accelerated simulation.

- 1 In the Simulink Editor, in the **Modeling** tab, click **Model Advisor**, then click **OK**.
- 2 Select **By Task**.
- 3 Run the **Check diagnostic settings ignored during accelerated model reference simulation** check.

Command-Line Information

Parameter: ReadBeforeWriteMsg

Value: 'UseLocalSettings' | 'DisableAll' | 'EnableAllAsWarning' | 'EnableAllAsError'

Default: 'UseLocalSettings'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	Enable all as errors

See Also

Simulink.Signal

Related Examples

- Diagnosing Simulation Errors
- “Local and Global Data Stores”
- Data Store Memory
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Detect write after read

Description

Select the diagnostic action to take if the model attempts to write data to a data store after previously reading data from it in the current time step.

Category: Diagnostics

Settings

Default: Use local settings

Use local settings

For each local data store (defined by a Data Store Memory block or Simulink.Signal object in a model workspace) use the setting specified by the block. For each global data store (defined by a Simulink.Signal object in the base workspace) disable the diagnostic.

Disable all

Disables this diagnostic for all data stores accessed by the model.

Enable all as warnings

Displays diagnostic as a warning at the MATLAB command line.

Enable all as errors

Halts the simulation and displays the diagnostic in an error dialog box.

Note During model referencing simulation in accelerator and rapid accelerator mode, if the **Detect write after read** parameter is set to **Enable all as warnings**, **Enable all as errors**, or **Use local settings**, Simulink temporarily changes the setting to **Disable all**.

You can use the Model Advisor to identify referenced models for which Simulink changes configuration this parameter setting during accelerated simulation.

- 1 In the Simulink Editor, in the **Modeling** tab, click **Model Advisor**, then click **OK**.
- 2 Select **By Task**.
- 3 Run the **Check diagnostic settings ignored during accelerated model reference simulation** check.

Command-Line Information

Parameter: WriteAfterReadMsg

Value: 'UseLocalSettings' | 'DisableAll' | 'EnableAllAsWarning' | 'EnableAllAsError'

Default: 'UseLocalSettings'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	Enable all as errors

See Also

Simulink.Signal

Related Examples

- Diagnosing Simulation Errors
- “Local and Global Data Stores”
- Data Store Memory
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Detect write after write

Description

Select the diagnostic action to take if the model attempts to write data to a data store twice in succession in the current time step.

Category: Diagnostics

Settings

Default: Use local settings

Use local settings

For each local data store (defined by a Data Store Memory block or Simulink.Signal object in a model workspace) use the setting specified by the block. For each global data store (defined by a Simulink.Signal object in the base workspace) disable the diagnostic.

Disable all

Disables this diagnostic for all data stores accessed by the model.

Enable all as warnings

Displays diagnostic as a warning at the MATLAB command line.

Enable all as errors

Halts the simulation and displays the diagnostic in an error dialog box.

Note During model referencing simulation in accelerator and rapid accelerator mode, if the **Detect write after write** parameter is set to **Enable all as warnings**, **Enable all as errors**, or **Use local settings**, Simulink temporarily changes the setting to **Disable all**.

You can use the Model Advisor to identify referenced models for which Simulink changes configuration this parameter setting during accelerated simulation.

- 1 In the Simulink Editor, in the **Modeling** tab, click **Model Advisor**, then click **OK**.
- 2 Select **By Task**.
- 3 Run the **Check diagnostic settings ignored during accelerated model reference simulation** check.

Command-Line Information

Parameter: WriteAfterWriteMsg

Value: 'UseLocalSettings' | 'DisableAll' | 'EnableAllAsWarning' | 'EnableAllAsError'

Default: 'UseLocalSettings'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	Enable all as errors

See Also

Simulink.Signal

Related Examples

- Diagnosing Simulation Errors
- “Local and Global Data Stores”
- Data Store Memory
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Multitask data store

Description

Select the diagnostic action to take when one task reads data from a Data Store Memory block to which another task writes data.

Category: Diagnostics

Settings

Default: error

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- Such a situation is safe only if one of the tasks cannot interrupt the other, such as when the data store is a scalar and the writing task uses an atomic copy operation to update the store or the target does not allow the tasks to preempt each other.
- You should disable this diagnostic (set it to `none`) only if the application warrants it, such as if the application uses a cyclic scheduler that prevents tasks from preempting each other.

Command-Line Information

Parameter: MultiTaskDSMMsg

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Simulink.Signal

Related Examples

- Diagnosing Simulation Errors
- “Local and Global Data Stores”
- Data Store Memory
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Duplicate data store names

Description

Select the diagnostic action to take when the model contains multiple data stores that have the same name. The data stores can be defined with Data Store Memory blocks or `Simulink.Signal` objects.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tip

This diagnostic is useful for detecting errors that can occur when a lower-level data store unexpectedly shadows a higher-level data store that has the same name.

Command-Line Information

Parameter: `UniqueDataStoreMsg`

Value: `'none' | 'warning' | 'error'`

Default: `'none'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

`Simulink.Signal`

Related Examples

- Diagnosing Simulation Errors

- “Local and Global Data Stores”
- Data Store Memory
- “Model Configuration Parameters: Data Validity Diagnostics” on page 6-2

Diagnostics Parameters: Model Referencing

Model Configuration Parameters: Model Referencing Diagnostics

The **Diagnostics > Model Referencing** category includes parameters for detecting issues related to referenced models (Model blocks).

Parameter	Description
"Model block version mismatch" on page 7-3	Select the diagnostic action to take when loading or updating this model if Simulink software detects a mismatch between the version of the model used to create or refresh a Model block in this model and the referenced model's current version.
"Port and parameter mismatch" on page 7-5	Select the diagnostic action to take if Simulink software detects a port or parameter mismatch during model loading or updating.
"Invalid root Inport/Outport block connection" on page 7-7	Select the diagnostic action to take if Simulink software detects invalid internal connections to this model's root-level Output port blocks.
"Unsupported data logging" on page 7-11	Select the diagnostic action to take if this model contains To Workspace blocks or Scope blocks with data logging enabled.
"No explicit final value for model arguments" on page 7-13	Select the diagnostic action to take when the final value of a model argument is the default value instead of an explicit value.

See Also

Related Examples

- "Model References"
- Diagnosing Simulation Errors
- Solver Diagnostics on page 9-2
- Sample Time Diagnostics on page 8-2
- Data Validity Diagnostics on page 6-2
- Type Conversion Diagnostics on page 11-2
- Connectivity Diagnostics on page 5-2
- Compatibility Diagnostics on page 4-2

Model block version mismatch

Description

Select the diagnostic action to take when loading or updating this model if Simulink software detects a mismatch between the version of the model used to create or refresh a Model block in this model and the current version of the referenced model.

Category: Diagnostics

Settings

Default: none

none

Simulink software refreshes the Model block.

warning

Simulink software displays a warning and refreshes the Model block.

error

Simulink software displays an error message and does not refresh the Model block.

When you receive an error related to a Model block version mismatch, you can manually refresh the Model block. Select the Model block. Then, on the **Model Block** tab, select **Refresh**. Alternatively, use the `Simulink.ModelReference.refresh` function.

Tips

- Version mismatches can occur when you modify, save, and close a referenced model while the model that references it is not loaded. For more information, see “Manage Model Versions and Specify Model Properties”.
- Model block icons can display a message indicating version mismatches. To enable this feature, from the parent model, on the **Debug** tab, select **Information Overlays > Ref. Model Version**. The Model block displays a version mismatch, for example: `Rev:1.0 != 1.2`.

Command-Line Information

Parameter: `ModelReferenceVersionMismatchMessage`

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact

Application	Setting
Safety precaution	No recommendation

See Also

Related Examples

- “Model References”
- Diagnosing Simulation Errors
- “Model Version Numbers”
- “Model Configuration Parameters: Model Referencing Diagnostics” on page 7-2

Port and parameter mismatch

Description

Select the diagnostic action to take when loading or updating this model if Simulink software detects a port or parameter mismatch between a Model block and its referenced model.

Category: Diagnostics

Settings

Default: none

none

Simulink software refreshes the Model block.

warning

Simulink software displays a warning and refreshes the Model block.

error

Simulink software displays an error message and does not refresh the Model block.

When you receive an error related to a Model block port or parameter mismatch, you can manually refresh the Model block. Select the Model block. Then, on the **Model Block** tab, select **Refresh**. Alternatively, use the `Simulink.ModelReference.refresh` function.

Tips

- Port mismatches occur when the input and output ports of a Model block do not match the root-level input and output ports of the model it references.
- Parameter mismatches occur when the parameter arguments recognized by the Model block do not match the parameter arguments declared by the referenced model.
- Model block icons can display a message indicating port or parameter mismatches. To enable this feature, from the parent model, on the **Debug** tab, select **Information Overlays > Ref Model I/O Mismatch**.

Command-Line Information

Parameter: ModelReferenceIOMismatchMessage

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact

Application	Setting
Safety precaution	error

See Also

Related Examples

- “Model References”
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Model Referencing Diagnostics” on page 7-2

Invalid root Inport/Outport block connection

Description

Select the diagnostic action to take if Simulink software detects invalid internal connections to this model's root-level Output port blocks.

Category: Diagnostics

Settings

Default: none

none

Simulink software silently inserts hidden blocks to satisfy the constraints wherever possible.

warning

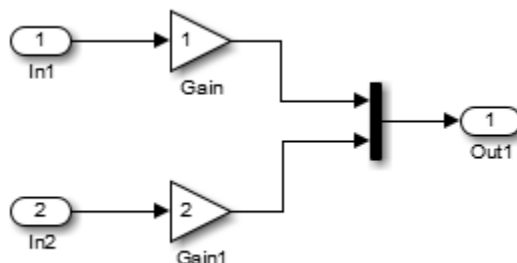
Simulink software warns you that a connection constraint has been violated and attempts to satisfy the constraint by inserting hidden blocks.

error

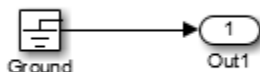
Simulink software terminates the simulation or code generation and displays an error message.

Tips

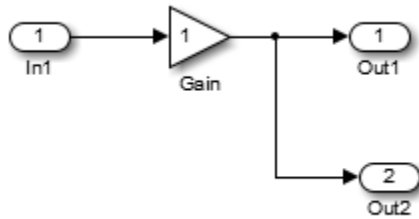
- In some cases (such as function-call feedback loops), automatically inserted hidden blocks may introduce delays and thus may change simulation results.
- Auto-inserting hidden blocks to eliminate root I/O problems stops at subsystem boundaries. Therefore, you may need to manually modify models with subsystems that violate any of the constraints below.
- The types of invalid internal connections are:
 - A root output port is connected directly or indirectly to more than one nonvirtual block port:



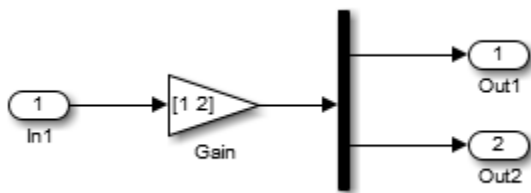
- A root output port is connected to a Ground block:



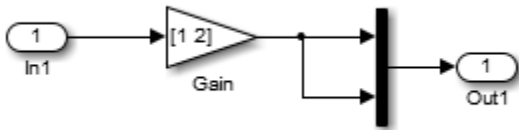
- Two root Output blocks are connected to the same block port:



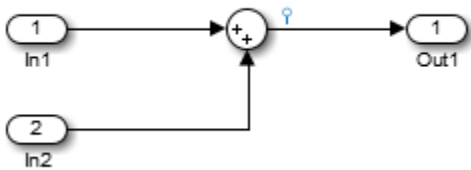
- An Output block is connected to some elements of a block output and not others:



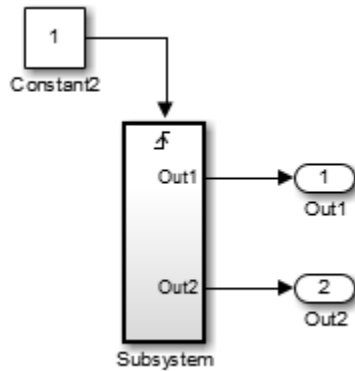
- An Output block is connected more than once to the same element:



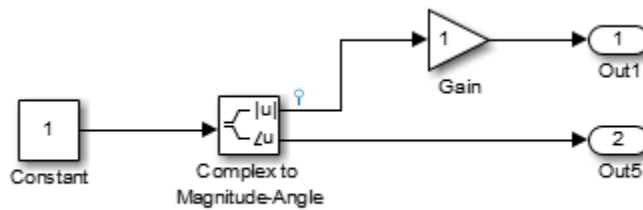
- The signal driving the root output port is a test point:



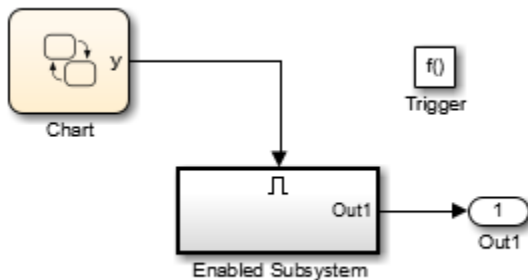
- The output port has a constant sample time, but the driving block has a non-constant sample time:



- The driving block has a constant sample time and multiple output ports, and one of the other output ports of the block is a test point.



- The root output port is conditionally computed, you are using Function Prototype Control or a Encapsulated C++ target, and the Function Prototype specification or C++ target specification states that the output variable corresponding to that root output port is returned by value.



Command-Line Information

Parameter: ModelReferenceIOMsg
Value: 'none' | 'warning' | 'error'
Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Model Reference Requirements and Limitations”
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Model Referencing Diagnostics” on page 7-2

Unsupported data logging

Description

Select the diagnostic action to take if this model contains To Workspace blocks or Scope blocks with data logging enabled.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- The default action warns you that Simulink software does not support use of these blocks to log data from referenced models.
- See “Models with Model Referencing: Overriding Signal Logging Settings” for information on how to log signals from a reference to this model.

Command-Line Information

Parameter: ModelReferenceDataLoggingMessage

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Model References”
- Diagnosing Simulation Errors
- “Models with Model Referencing: Overriding Signal Logging Settings”
- To Workspace
- Scope
- “Model Configuration Parameters: Model Referencing Diagnostics” on page 7-2

No explicit final value for model arguments

Description

Select the diagnostic action to take when the topmost Model block that can set the value for a model argument uses a default value instead of providing an explicit value.

In the Model Data Editor, Property Inspector, or Model Explorer, the default value for a model argument displays as either `<inherited>` or `<from below>`.

- If the **Argument** check box is selected, Simulink displays `<inherited>` to indicate that, in the case that a model references the Model block, the model argument value is provided by the parent.
- If the **Argument** check box is cleared, Simulink displays `<from below>` to indicate that its value is provided by the last model to specify a value in the model hierarchy below.

At the command-line, a default value for a model argument is represented by an empty string.

The value of this configuration parameter in the top model applies to each model argument in the model hierarchy.

Category: Diagnostics

Settings

Default: none

none

If the topmost Model block uses the default value for a model argument, Simulink uses the last value specified in the model hierarchy below.

warning

If the topmost Model block uses the default value for a model argument, Simulink uses the last value specified in the model hierarchy below, but displays a warning that the model argument does not have an explicit final value.

error

If the topmost Model block uses a default value for a model argument, Simulink displays an error message at compile time.

Command-Line Information

Parameter: ModelReferenceNoExplicitFinalValueMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Model References”
- Diagnosing Simulation Errors
- “Parameterize Instances of a Reusable Referenced Model”
- “Parameterize a Referenced Model Programmatically”
- “Model Configuration Parameters: Model Referencing Diagnostics” on page 7-2

Diagnostics Parameters: Sample Time

Model Configuration Parameters: Sample Time Diagnostics

The **Diagnostics > Sample Time** category includes parameters for detecting issues related to sample time and sample time specifications.

Parameter	Description
"Source block specifies -1 sample time" on page 8-3	Select the diagnostic action to take if a source block (such as a Sine Wave block) specifies a sample time of -1.
"Multitask data transfer" on page 8-5	Select the diagnostic action to take if an invalid rate transition occurred between two blocks operating in multitasking mode.
"Single task data transfer" on page 8-7	Select the diagnostic action to take if a rate transition occurred between two blocks operating in single-tasking mode.
"Multitask conditionally executed subsystem" on page 8-9	Select the diagnostic action to take if Simulink software detects a subsystem that may cause data corruption or non-deterministic behavior.
"Tasks with equal priority" on page 8-11	Select the diagnostic action to take if Simulink software detects two tasks with equal priority that can preempt each other in the target system.
"Exported tasks rate transition" on page 8-15	Select the diagnostic action to take if Simulink software detects unspecified data transfers between exported tasks.
"Enforce sample times specified by Signal Specification blocks" on page 8-13	Select the diagnostic action to take if the sample time of the source port of a signal specified by a Signal Specification block differs from the signal's destination port.
"Unspecified inheritability of sample time" on page 8-16	Select the diagnostic action to take if this model contains S-functions that do not specify whether they preclude this model from inheriting their sample times from a parent model.

See Also

Related Examples

- Diagnosing Simulation Errors
- Solver Diagnostics on page 9-2
- Data Validity Diagnostics on page 6-2
- Type Conversion Diagnostics on page 11-2
- Connectivity Diagnostics on page 5-2
- Compatibility Diagnostics on page 4-2
- Model Referencing Diagnostics on page 7-2

Source block specifies -1 sample time

Description

Select the diagnostic action to take if a source block (such as a Sine Wave block) specifies a sample time of -1.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- The Random Source block does not obey this parameter. If its **Sample time** parameter is set to -1, the Random Source block inherits its sample time from its output port and never produces warnings or errors.
- Some Communications Toolbox™ blocks internally inherit sample times, which can be a useful and valid modeling technique. Set this parameter to none for these types of models.

Command-Line Information

Parameter: InheritedTsInSrcMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Sample Time Diagnostics” on page 8-2

Multitask data transfer

Description

Select the diagnostic action to take if an invalid rate transition occurred between two blocks operating in multitasking mode.

Category: Diagnostics

Settings

Default: error

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- This parameter allows you to adjust error checking for sample rate transitions between blocks that operate at different sample rates.
- Use this option for models of real-time multitasking systems to ensure detection of illegal rate transitions between tasks that can result in a task's output being unavailable when needed by another task. You can then use Rate Transition blocks to eliminate such illegal rate transitions from the model.

Command-Line Information

Parameter: MultiTaskRateTransMsg

Value: 'warning' | 'error'

Default: 'error'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Rate Transition

- “Model Execution and Rate Transitions” (Simulink Coder)
- Single-Tasking and Multitasking Execution Modes (Simulink Coder)
- “Handle Rate Transitions” (Simulink Coder)
- “Treat each discrete rate as a separate task” on page 14-42
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Sample Time Diagnostics” on page 8-2

Single task data transfer

Description

Select the diagnostic action to take if a rate transition occurred between two blocks operating in single-tasking mode.

Category: Diagnostics

Settings

Default: none

none

Simulink takes no action.

warning

Simulink displays a warning.

error

Simulink terminates the simulation and displays an error message.

Tips

- This parameter allows you to adjust error checking for sample rate transitions between blocks that operate at different sample rates.
- Use this parameter when you are modeling a single-tasking system. In such systems, task synchronization is not an issue.
- Since variable step solvers are always single tasking, this parameter applies to them.
- The **Single task data transfer** parameter affects the blocks that are inserted if the **Automatically handle data transfers** parameter is also selected. Those inserted blocks may change the simulation results and block sorted order in some cases.

Command-Line Information

Parameter: SingleTaskRateTransMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	none or error

See Also

Related Examples

- Rate Transition
- “Model Execution and Rate Transitions” (Simulink Coder)
- Single-Tasking and Multitasking Execution Modes (Simulink Coder)
- “Handle Rate Transitions” (Simulink Coder)
- “Treat each discrete rate as a separate task” on page 14-42
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Sample Time Diagnostics” on page 8-2

Multitask conditionally executed subsystem

Description

Select the diagnostic action to take if Simulink software detects a subsystem that may cause data corruption or non-deterministic behavior.

Category: Diagnostics

Settings

Default: error

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- These types of subsystems can be caused by either of the following conditions:
 - Your model uses multitasking solver mode and it contains an enabled subsystem that operates at multiple rates.
 - Your model contains a conditionally executed subsystem that can reset its states and that contains an asynchronous subsystem.

These types of subsystems can cause corrupted data or nondeterministic behavior in a real-time system that uses code generated from the model.

- For models that use multitasking solver mode and contain an enabled subsystem that operates at multiple rates, consider using single-tasking solver mode or using a single-rate enabled subsystem instead.
- For models that contain a conditionally executed subsystem that can reset its states and that contains an asynchronous subsystem, consider moving the asynchronous subsystem outside the conditionally executed subsystem.

Command-Line Information

Parameter: MultiTaskCondExecSysMsg

Value: 'none' | 'warning' | 'error'

Default: 'error'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Treat each discrete rate as a separate task” on page 14-42
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Sample Time Diagnostics” on page 8-2

Tasks with equal priority

Description

Select the diagnostic action to take if Simulink software detects two tasks with equal priority that can preempt each other in the target system.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- This condition can occur when one asynchronous task of the target represented by this model has the same priority as one of the target's asynchronous tasks.
- This option must be set to Error if the target allows tasks having the same priority to preempt each other.

Command-Line Information

Parameter: TasksWithSamePriorityMsg

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	none or error

See Also

Related Examples

- Diagnosing Simulation Errors
- “Rate Transitions and Asynchronous Blocks” (Simulink Coder)
- “Model Configuration Parameters: Sample Time Diagnostics” on page 8-2

Enforce sample times specified by Signal Specification blocks

Description

Select the diagnostic action to take if the sample time of the source port of a signal specified by a Signal Specification block differs from the signal's destination port.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Note The setting of this diagnostic is ignored when the **Default parameter behavior** parameter in the **Code Generation > Optimization** pane of the model Configuration Parameters is set to Tunable.

Tips

- The Signal Specification block allows you to specify the attributes of the signal connected to its input and output ports. If the specified attributes conflict with the attributes specified by the blocks connected to its ports, Simulink software displays an error when it compiles the model, for example, at the beginning of a simulation. If no conflict exists, Simulink software eliminates the Signal Specification block from the compiled model.
- You can use the Signal Specification block to ensure that the actual attributes of a signal meet desired attributes, or to ensure correct propagation of signal attributes throughout a model.

Command-Line Information

Parameter: SigSpecEnsureSampleTimeMsg

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact

Application	Setting
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- Signal Specification
- “Model Configuration Parameters: Sample Time Diagnostics” on page 8-2

Exported tasks rate transition

Description

Select the diagnostic action to take if Simulink software detects unspecified data transfers between exported tasks.

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Command-Line Information

Parameter: ExportedTasksRateTransMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Unspecified inheritability of sample time

Description

Select the diagnostic action to take if this model contains S-functions that do not specify whether they preclude this model from inheriting their sample times from a parent model.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- Not specifying an inheritance rule may lead to incorrect simulation results.
- Simulink software checks for this condition only if the solver used to simulate this model is a fixed-step discrete solver and the periodic sample time constraint for the solver is set to ensure sample time independence
- For more information, see “Periodic sample time constraint” on page 14-59.

Command-Line Information

Parameter: UnknownTsInhSupMsg

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- “Periodic sample time constraint” on page 14-59
- Solver Diagnostics on page 9-2
- “Model Configuration Parameters: Sample Time Diagnostics” on page 8-2

Diagnostics Parameters

Model Configuration Parameters: Diagnostics

The **Diagnostics** category includes parameters for detecting issues related to solvers and solver settings, for example, algebraic loops.

Parameter	Description
"Algebraic loop" on page 9-5	Select the diagnostic action to take if Simulink software detects an algebraic loop while compiling the model.
"Minimize algebraic loop" on page 9-7	Select the diagnostic action to take if artificial algebraic loop minimization cannot be performed for an atomic subsystem or Model block because an input port has direct feedthrough.
"Block priority violation" on page 9-9	Select the diagnostic action to take if Simulink software detects a block priority specification error.
"Min step size violation" on page 9-11	Select the diagnostic action to take if Simulink software detects that the next simulation step is smaller than the minimum step size specified for the model.
"Consecutive zero-crossings violation" on page 9-13	Select the diagnostic action to take when Simulink software detects that the number of consecutive zero crossings exceeds the specified maximum.
"Automatic solver parameter selection" on page 9-15	Select the diagnostic action to take if Simulink software changes a solver parameter setting.
"Extraneous discrete derivative signals" on page 9-17	Select the diagnostic action to take when a discrete signal appears to pass through a Model block to the input of a block with continuous states.
"State name clash" on page 9-19	Select the diagnostic action to take when a name is used for more than one state in the model.
"Operating point restore interface checksum mismatch" on page 9-23	Use this check to ensure that the interface checksum is identical to the model checksum before loading the OperatingPoint.

These configuration parameters are in the **Advanced parameters** section.

Parameter	Description
"Allow symbolic dimension specification" on page 2-111	Specify whether Simulink propagates dimension symbols throughout the model and preserves these symbols in the propagated signal dimensions.
"Allowed unit systems" on page 2-46	Specify unit systems allowed in the model.

Parameter	Description
"Units inconsistency messages" on page 2-48	Specify if unit inconsistencies should be reported as warnings. Select the diagnostic action to take when the Simulink software detects unit inconsistencies.
"Allow automatic unit conversions" on page 2-49	Allow automatic unit conversions in the model.
"Check undefined subsystem initial output" on page 2-60	Specify whether to display a warning if the model contains a conditionally executed subsystem in which a block with a specified initial condition drives an Outport block with an undefined initial condition.
"Solver data inconsistency" on page 2-67	Select the diagnostic action to take if Simulink software detects S-functions that have continuous sample times, but do not produce consistent results when executed multiple times.
"Ignored zero crossings" on page 2-69	Select the diagnostic action to take if Simulink detects zero-crossings that are being ignored
"Masked zero crossings" on page 2-71	Select the diagnostic action to take if Simulink detects zero-crossings that are being masked.
"Block diagram contains disabled library links" on page 2-72	Select the diagnostic action to take when saving a model containing disabled library links.
"Block diagram contains parameterized library links" on page 2-74	Select the diagnostic action to take when saving a model containing parameterized library links.
"Initial state is array" on page 2-75	Message behavior when the initial state is an array
"Insufficient maximum identifier length" on page 2-77	For referenced models, specify diagnostic action when the configuration parameter Maximum identifier length does not provide enough character length to make global identifiers unique across models.
"Combine output and update methods for code generation and simulation" on page 2-119	When output and update code is in one function, force simulation execution order to be the same as code generation order. For certain modeling patterns, setting this parameter prevents a potential simulation and code generation mismatch. Setting this parameter might cause artificial algebraic loops.
"Behavior when pregenerated library subsystem code is missing" on page 2-123	When generating code for a model that contains an instance of a reusable library subsystem with a function interface, specify whether or not to display a warning or an error when the model cannot use pregenerated library code or pregenerated library code is missing.
"FMU Import blocks" on page 2-129	When the debug execution mode is enabled, FMU binaries are executed in a separate process.

Parameter	Description
“Arithmetic operations in variant conditions” on page 2-125	Specify the diagnostic action to take when arithmetic operations are found in variant conditions.
“Variant condition mismatch at signal source and destination” on page 2-130	Specify the diagnostic action to take when there are variant-related modeling issues that may result in unused Simulink variables in the generated code.
“Variant activation time inherited from Simulink.VariantControl” on page 2-127	Specify the diagnostic action to take when a variant block with its activation time set to <code>inherit from Simulink.VariantControl</code> has no variant control variable of type <code>Simulink.VariantControl</code> .
“Variant configuration not used by top model” on page 2-138	Specify the diagnostic action to take when Simulink detects during simulation or Variant Manager activation that a top-level model does not use a referenced model for any of the published variant configurations of the referenced model.

See Also

Related Examples

- “Algebraic Loop Concepts”
- Diagnosing Simulation Errors
- Sample Time Diagnostics on page 8-2
- Data Validity Diagnostics on page 6-2
- Type Conversion Diagnostics on page 11-2
- Connectivity Diagnostics on page 5-2
- Compatibility Diagnostics on page 4-2
- Model Referencing Diagnostics on page 7-2

Algebraic loop

Description

Select the diagnostic action to take if Simulink software detects an algebraic loop while compiling the model.

Category: Diagnostics

Settings

Default: warning

none

When the Simulink software detects an algebraic loop, the software tries to solve the algebraic loop. If the software cannot solve the algebraic loop, it reports an error and the simulation terminates.

warning

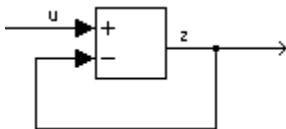
When Simulink software detects an algebraic loop, it displays a warning and tries to solve the algebraic loop. If the software cannot solve the algebraic loop, it reports an error and the simulation terminates.

error

When Simulink software detects an algebraic loop, it terminates the simulation, displays an error message, and highlights the portion of the block diagram that comprises the loop.

Tips

- An *algebraic loop* generally occurs when an input port with direct feedthrough is driven by the output of the same block, either directly, or by a feedback path through other blocks with direct feedthrough. An example of an algebraic loop is this simple scalar loop.



- When a model contains an algebraic loop, Simulink software calls a loop-solving routine at each time step. The loop solver performs iterations to determine the solution to the problem (if it can). As a result, models with algebraic loops run slower than models without them.
- Use the **error** option to highlight algebraic loops when you simulate a model. This causes Simulink software to display an error dialog (the Diagnostic Viewer) and recolor portions of the diagram that represent the first algebraic loop that it detects. Simulink software uses red to color the blocks and lines that constitute the loop. Closing the error dialog restores the diagram to its original colors.
- See “Algebraic Loop Concepts” for more information.

Command-Line Information

Parameter: AlgebraicLoopMsg
Value: 'none' | 'warning' | 'error'
Default: 'warning'

Recommended Settings

Application	Setting
Debugging	error
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Algebraic Loop Concepts”
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Diagnostics” on page 9-2

Minimize algebraic loop

Description

Select the diagnostic action to take if artificial algebraic loop minimization cannot be performed for an atomic subsystem or Model block because an input port has direct feedthrough.

When you set the **Minimize algebraic loop occurrences** parameter for an atomic subsystem or a Model block, if Simulink detects an artificial algebraic loop, it attempts to eliminate the loop by checking for non-direct-feedthrough blocks before simulating the model. If Simulink cannot minimize the artificial algebraic loop, the simulation performs the diagnostic action specified by the **Minimize algebraic loop** parameter.

Category: Diagnostics

Settings

Default: warning

none

Simulink takes no action.

warning

Simulink displays a warning that it cannot minimize the artificial algebraic loop.

error

Simulink terminates the simulation and displays an error that it cannot minimize the artificial algebraic loop.

Tips

- If the port is involved in an artificial algebraic loop, Simulink software can remove the loop only if at least one other input port in the loop lacks direct feedthrough.
- Simulink software cannot minimize artificial algebraic loops containing signals designated as test points (see Working with Test Points).

Command-Line Information

Parameter: ArtificialAlgebraicLoopMsg

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Efficiency	No impact
Traceability	No impact

Application	Setting
Safety precaution	error

See Also

Related Examples

- “How Simulink Eliminates Artificial Algebraic Loops”
- Diagnosing Simulation Errors
- Working with Test Points
- “Model Configuration Parameters: Diagnostics” on page 9-2

Block priority violation

Description

Select the diagnostic action to take if Simulink software detects a block priority specification error.

Category: Diagnostics

Settings

Default: warning

warning

When Simulink software detects a block priority specification error, it displays a warning.

error

When Simulink software detects a block priority specification error, it terminates the simulation and displays an error message.

Tips

- Simulink software allows you to assign update priorities to blocks. Simulink software executes the output methods of higher priority blocks before those of lower priority blocks.
- Simulink software honors the block priorities that you specify only if they are consistent with the Simulink block sorting algorithm. If Simulink software is unable to honor a user specified block priority, it generates a block priority specification error.

Command-Line Information

Parameter: BlockPriorityViolationMsg

Value: 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Controlling and Displaying the Sorted Order

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Diagnostics” on page 9-2

Min step size violation

Description

Select the diagnostic action to take if Simulink software detects that the next simulation step is smaller than the minimum step size specified for the model.

Category: Diagnostics

Settings

Default: warning

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- A minimum step size violation can occur if the specified error tolerance for the model requires a step size smaller than the specified minimum step size. See “Min step size” on page 14-20 and “Maximum order” on page 14-28 for more information.
- Simulink software allows you to specify the maximum number of consecutive minimum step size violations permitted (see “Number of consecutive min steps” on page 14-32).

Command-Line Information

Parameter: MinStepSizeMsg

Value: 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Min step size” on page 14-20

- “Maximum order” on page 14-28
- “Number of consecutive min steps” on page 14-32
- “Purely Discrete Systems”
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Diagnostics” on page 9-2

Consecutive zero-crossings violation

Description

Select the diagnostic action to take when Simulink software detects that the number of consecutive zero crossings exceeds the specified maximum.

Category: Diagnostics

Settings

Default: error

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- If you select `warning` or `error`, Simulink software reports the current simulation time, the number of consecutive zero crossings counted, and the type and name of the block in which Simulink software detected the zero crossings.
- For more information, see “Preventing Excessive Zero Crossings”.

Dependency

This diagnostic applies only when you are using a variable-step solver and the zero-crossing control is set to either `Enable all` or `Use local settings`.

Command-Line Information

Parameter: `MaxConsecutiveZCsMsg`

Value: `'none' | 'warning'`

Default: `'error'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	warning or error

See Also

Related Examples

- “Zero-Crossing Detection”
- “Zero-crossing control” on page 14-49
- “Number of consecutive zero crossings” on page 14-53
- “Time tolerance” on page 14-51
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Diagnostics” on page 9-2

Automatic solver parameter selection

Description

Select the diagnostic action to take if Simulink software changes a solver parameter setting.

Category: Diagnostics

Settings

Default: none

none

Simulink takes no action.

warning

Simulink displays a warning.

error

Simulink terminates the simulation and displays an error message.

Tips

When enabled, this option notifies you if:

- Simulink changes a user-modified parameter to make it consistent with other model settings.
- Simulink automatically selects solver parameters for the model, such as `FixedStepSize`.

For example, if you simulate a discrete model that specifies a continuous solver, Simulink changes the solver type to discrete and displays a warning about this change.

Command-Line Information

Parameter: `SolverPrmCheckMsg`

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- Choosing a Solver
- “Model Configuration Parameters: Diagnostics” on page 9-2

Extraneous discrete derivative signals

Description

Select the diagnostic action to take when a discrete signal appears to pass through a Model block to the input of a block with continuous states.

Category: Diagnostics

Settings

Default: error

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- This error can occur if a discrete signal passes through a Model block to the input of a block with continuous states, such as an Integrator block. In this case, Simulink software cannot determine with certainty the minimum rate at which it needs to reset the solver to solve this model accurately.
- If this diagnostic is set to `none` or `warning`, Simulink software resets the solver whenever the value of the discrete signal changes. This ensures accurate simulation of the model if the discrete signal is the source of the signal entering the block with continuous states. However, if the discrete signal is not the source of the signal entering the block with continuous states, resetting the solver at the rate the discrete signal changes can lead to the solver being reset more frequently than necessary, slowing down the simulation.
- If this diagnostic is set to `error`, Simulink software halts when compiling this model and displays an error.

Dependency

This diagnostic applies only when you are using a variable-step ode solver and the block diagram contains Model blocks.

Command-Line Information

Parameter: ModelReferenceExtraNoncontSigs

Value: 'none' | 'warning' | 'error'

Default: 'error'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- Diagnosing Simulation Errors
- Choosing a Solver
- “Model Configuration Parameters: Diagnostics” on page 9-2

State name clash

Description

Select the diagnostic action to take when a name is used for more than one state in the model.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

Tips

- This diagnostic applies for continuous and discrete states during simulation.
- This diagnostic applies only if you save states to the MATLAB workspace using the format **Structure** or **Structure with time**. If you do not save states in structure format, the state names are not used, and therefore the diagnostic will not warn you about a naming conflict.

Command-Line Information

Parameter: StateNameClashWarn

Value: 'none' | 'warning'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Data Import/Export” on page 3-2
- “Save Run-Time Data from Simulation”

- “Model Configuration Parameters: Diagnostics” on page 9-2

SimState interface checksum mismatch

Note **SimState interface checksum mismatch** is not recommended. Use **Operating point interface checksum mismatch** instead.

Description

Use this check to ensure that the interface checksum is saved in a `SimState` object, identical to the model checksum before loading the operating point.

Category: Diagnostics

Settings

Default: warning

none

Simulink software does not compare the interface checksum to the model checksum.

warning

The interface checksum in the `SimState` is different than the model checksum.

error

When Simulink detects that a change in the configuration settings occurred after saving the `SimState`, it does not load the `SimState` and reports an error.

Command-Line Information

Parameter: `SimStateInterfaceChecksumMismatchMsg`

Value: 'warning' | 'error' | 'none'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

`Simulink.BlockDiagram.getChecksum`

Related Examples

- “Use Model Operating Point for Faster Simulation Workflow”

- “Model Configuration Parameters: Diagnostics” on page 9-2

Operating point restore interface checksum mismatch

Description

Use this check to ensure that the interface checksum is identical to the model checksum before loading the `Simulink.op.ModelOperatingPoint` object specified using the **Initial state** parameter.

Category: Diagnostics

Settings

Default: warning

none

Simulink software does not compare the interface checksum to the model checksum.

warning

The interface checksum in the operating point is different than the model checksum.

error

When Simulink detects that a change in the configuration settings occurred after saving the operating point, it does not load the `ModelOperatingPoint` object and reports an error.

Command-Line Information

Parameter: `OperatingPointInterfaceChecksumMismatchMsg`

Value: 'warning' | 'error' | 'none'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

`Simulink.BlockDiagram.getChecksum`

Related Examples

- “Use Model Operating Point for Faster Simulation Workflow”
- “Model Configuration Parameters: Diagnostics” on page 9-2

Diagnostics Parameters: Stateflow

Model Configuration Parameters: Stateflow Diagnostics

The **Diagnostics > Stateflow** category includes parameters for detecting issues related to Stateflow charts.

Parameter	Description
“Unused data, events, messages, and functions” on page 10-4	Select the diagnostic action to take for detection of unused data, events, and messages in a chart. Removing unused data, events, and messages can minimize the size of your model.
“Unexpected backtracking” on page 10-6	Select the diagnostic action to take when a chart junction has both of the following conditions. The junction: <ul style="list-style-type: none"> • Does not have an unconditional transition path to a state or a terminal junction • Has multiple transition paths leading to it
“Invalid input data access in chart initialization” on page 10-8	Select the diagnostic action to take when a chart: <ul style="list-style-type: none"> • Has the <code>ExecuteAtInitialization</code> property set to <code>true</code> • Accesses input data on a default transition or associated state entry actions, which execute at chart initialization
“No unconditional default transitions” on page 10-10	Select the diagnostic action to take when a chart does not have an unconditional default transition to a state.
“Transition outside natural parent” on page 10-12	Select the diagnostic action to take when a chart contains a transition that loops outside of the parent state or junction.
“Undirected event broadcasts” on page 10-13	Select the diagnostic action to take when a chart contains undirected local event broadcasts.
“Transition action specified before condition action” on page 10-14	Select the diagnostic action to take when a transition action executes before a condition action in a transition path with multiple transition segments.
“Read-before-write to output in Moore chart” on page 10-16	Select the diagnostic action to take when a Moore chart uses a previous output value to determine the current state.
“Absolute time temporal value shorter than sampling period” on page 10-17	Select the diagnostic action to take when a state or transition absolute time operator uses a time value that is shorter than the sample time for the Stateflow block.
“Self transition on leaf state” on page 10-18	Select the diagnostic action to take when you can remove a self-transition on a leaf state.

Parameter	Description
“Execute-at-Initialization disabled in presence of input events” on page 10-19	Select the diagnostic action to take when Stateflow detects triggered or enabled charts that are not running at initialization.
“Unreachable execution path” on page 10-23	Select the diagnostic action to take when there are chart constructs not on a valid execution path.

This configuration parameter is in the **Advanced parameters** section.

Parameter	Description
“Use of machine-parented data instead of Data Store Memory” on page 10-21	Select the diagnostic action to take when Stateflow detects machine-parented data that can replace with chart-parented data of scope Data Store Memory.

See Also

Related Examples

- Diagnosing Simulation Errors
- Solver Diagnostics on page 9-2
- Sample Time Diagnostics on page 8-2
- Data Validity Diagnostics on page 6-2
- Type Conversion Diagnostics on page 11-2
- Connectivity Diagnostics on page 5-2
- Compatibility Diagnostics on page 4-2
- Model Referencing Diagnostics on page 7-2

Unused data, events, messages, and functions

Description

Select the diagnostic action to take for detection of unused data, events, messages, and functions in a chart. Removing unused data, events, messages, and functions can minimize the size of your model.

Category: Diagnostics

Settings

Default: warning

none

No warning or error appears.

warning

A warning appears, with a link to delete the unused data, event, or message in your chart.

error

An error appears and stops the simulation.

Tip

This diagnostic does not detect these types of data and events:

- Machine-parented data
- Inputs and outputs of MATLAB functions
- Input events

Command-Line Information

Parameter: SFUnusedDataAndEventsDiag

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	warning
Traceability	No impact
Efficiency	No impact (for simulation) none (for production code generation)
Safety precaution	warning

See Also

Related Examples

- “Synchronize Model Components by Broadcasting Events” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 10-2

Unexpected backtracking

Description

Select the diagnostic action to take when a chart junction has both of the following conditions. The junction:

- Does not have an unconditional transition path to a state or a terminal junction
- Has multiple transition paths leading to it

This chart configuration can lead to unwanted backtracking during simulation.

Category: Diagnostics

Settings

Default: error

none

No warning or error appears.

warning

A warning appears, with a link to examples of unwanted backtracking.

error

An error appears and stops the simulation.

Tip

To avoid unwanted backtracking, consider adding an unconditional transition from the chart junction to a terminal junction.

Command-Line Information

Parameter: SFUnexpectedBacktrackingDiag

Value: 'none' | 'warning' | 'error'

Default: 'error'

Recommended Settings

Application	Setting
Debugging	warning
Traceability	No impact
Efficiency	No impact (for simulation) No impact (for production code generation)
Safety precaution	error

See Also

Related Examples

- “Model Configuration Parameters: Stateflow Diagnostics” on page 10-2
- “Best Practices for Creating Flow Charts” (Stateflow)
- “Backtrack in Flow Charts” (Stateflow)
- “Detect Modeling Errors During Edit Time” (Stateflow)
- “Unexpected backtracking” (Stateflow)

Invalid input data access in chart initialization

Description

Select the diagnostic action to take when a chart:

- Has the `ExecuteAtInitialization` property set to `true`
- Accesses input data on a default transition or associated state entry actions, which execute at chart initialization

In this chart configuration, blocks that connect to chart input ports might not initialize their outputs during initialization. To locate this configuration in your model and correct it, use this diagnostic.

When using Embedded Coder for a component model configured with a service interface, this parameter is not relevant and, therefore, is not supported.

Category: Diagnostics

Settings

Default: warning

none

No warning or error appears.

warning

A warning appears.

error

An error appears and stops the simulation.

Tip

In charts that do not contain states, the `ExecuteAtInitialization` property has no effect.

Command-Line Information

Parameter: `SFInvalidInputDataAccessInChartInitDiag`

Value: `'none' | 'warning' | 'error'`

Default: `'warning'`

Recommended Settings

Application	Setting
Debugging	warning
Traceability	No impact
Efficiency	No impact (for simulation) No impact (for production code generation)

Application	Setting
Safety precaution	error

See Also

Related Examples

- “Execution of a Chart at Initialization” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 10-2

No unconditional default transitions

Description

Select the diagnostic action to take when a chart does not have an unconditional default transition to a state.

This chart construct can cause inconsistency errors. To locate this construct in your model and correct it, use this diagnostic. If a chart contains local event broadcasts or implicit events, detection of a state inconsistency might not be possible until run time.

Category: Diagnostics

Settings

Default: error

none

No warning or error appears.

warning

A warning appears.

error

An error appears and stops the simulation.

Command-Line Information

Parameter: SFNoUnconditionalDefaultTransitionDiag

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	error
Traceability	No impact
Efficiency	No impact (for simulation) none (for production code generation)
Safety precaution	error

See Also

Related Examples

- “Model Configuration Parameters: Stateflow Diagnostics” on page 10-2
- “Transition Between Operating Modes” (Stateflow)

- “Detect State Inconsistencies” (Stateflow)
- “Detect Modeling Errors During Edit Time” (Stateflow)
- “Default transition path does not terminate in a state” (Stateflow)

Transition outside natural parent

Description

Select the diagnostic action to take when a chart contains a transition that loops outside of the parent state or junction.

Category: Diagnostics

Settings

Default: warning

none

No warning or error appears.

warning

A warning appears.

error

An error appears and stops the simulation.

Command-Line Information

Parameter: SFTTransitionOutsideNaturalParentDiag

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	warning
Traceability	No impact
Efficiency	No impact (for simulation) none (for production code generation)
Safety precaution	error

See Also

Related Examples

- “Model Configuration Parameters: Stateflow Diagnostics” on page 10-2
- “Transition Between Operating Modes” (Stateflow)
- “Detect Modeling Errors During Edit Time” (Stateflow)
- “Transition loops outside natural parent” (Stateflow)
- “Unconditional path out of state with during actions or child states” (Stateflow)

Undirected event broadcasts

Description

Select the diagnostic action to take when a chart contains undirected local event broadcasts.

Undirected local event broadcasts can cause unwanted recursive behavior in a chart and inefficient code generation. To flag these types of event broadcasts and fix them, use this diagnostic.

Category: Diagnostics

Settings

Default: warning

none

No warning or error appears.

warning

A warning appears.

error

An error appears and stops the simulation.

Command-Line Information

Parameter: SFUndirectedBroadcastEventsDiag

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	warning
Traceability	No impact
Efficiency	warning
Safety precaution	error

See Also

Related Examples

- “Avoid Unwanted Recursion in a Chart” (Stateflow)
- “Broadcast Local Events to Synchronize Parallel States” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 10-2

Transition action specified before condition action

Description

Select the diagnostic action to take when a transition action executes before a condition action in a transition path with multiple transition segments.

When a transition with a specified transition action precedes a transition with a specified condition action in the same transition path, out-of-order execution can occur. To flag such behavior in your chart and fix it, use this diagnostic.

Category: Diagnostics

Settings

Default: warning

none

No warning or error appears.

warning

A warning appears.

error

An error appears and stops the simulation.

Command-Line Information

Parameter: SFTransitionActionBeforeConditionDiag

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	warning
Traceability	warning
Efficiency	warning
Safety precaution	error

See Also

Related Examples

- “Model Configuration Parameters: Stateflow Diagnostics” on page 10-2
- “Transition Between Operating Modes” (Stateflow)
- “Detect Modeling Errors During Edit Time” (Stateflow)

- “Transition action precedes a condition action along this path” (Stateflow)

Read-before-write to output in Moore chart

Description

Select the diagnostic action to take when a Moore chart uses a previous output value to determine the current state. This behavior violates Moore machine semantics. In a Moore machine, output is a function of current state only. To allow output values from the previous time step in calculating current state, set this diagnostic to `warning` or `none`.

Category: Diagnostics

Settings

Default: `error`

`none`

No warning or error appears.

`warning`

A warning appears.

`error`

An error appears and stops the simulation.

Command-Line Information

Parameter: `SFOutputUsedAsStateInMooreChartDiag`

Value: `'none' | 'warning' | 'error'`

Default: `'error'`

Recommended Settings

Application	Setting
Debugging	<code>error</code>
Traceability	<code>error</code>
Efficiency	<code>error</code>
Safety precaution	<code>error</code>

See Also

Related Examples

- “Design Considerations for Moore Charts” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 10-2

Absolute time temporal value shorter than sampling period

Description

Select the diagnostic action to take when a state or transition absolute time operator uses a time value that is shorter than the sample time for the Stateflow block. Stateflow cannot update states in smaller increments than the sample time for the block. For example, a model with a sample rate of 0.1 sec and an operator `after(5,usec)` triggers this diagnostic. If this parameter is set to warning or none, then the operator is evaluated as true at every time step.

Category: Diagnostics

Settings

Default: warning

none

No warning or error appears.

warning

A warning appears.

error

An error appears and stops the simulation.

Command-Line Information

Parameter: SFTemporalDelaySmallerThanSampleTimeDiag

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	error
Traceability	error
Efficiency	error
Safety precaution	error

See Also

Related Examples

- “Update Method” (Stateflow)
- “Control Chart Execution by Using Temporal Logic” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 10-2

Self transition on leaf state

Description

Select the diagnostic action to take when you can remove a self-transition on a leaf state. Some self-transitions with no actions in the leaf state or on the self-transition have no effect on chart execution. Removing these transitions simplifies the state diagram.

Category: Diagnostics

Settings

Default: warning

none

No warning or error appears.

warning

A warning appears.

error

An error appears and stops the simulation.

Command-Line Information

Parameter: SFSelfTransitionDiag

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	error
Traceability	error
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Model Configuration Parameters: Stateflow Diagnostics” on page 10-2

Execute-at-Initialization disabled in presence of input events

Description

Select the diagnostic action to take when Stateflow detects triggered or enabled charts that are not running at initialization. When the chart does not execute at initialization, then the chart default transitions are processed at the first input event. Until then, any data that you initialize in the chart or active state data is not valid at time 0.

To initialize the chart configuration at time 0 rather than at the first input event, select the chart property **Execute (enter) Chart At Initialization**.

Category: Diagnostics

Settings

Default: warning

none

No warning or error appears.

warning

A warning appears.

error

An error appears and stops the simulation.

Command-Line Information

Parameter: SFExecutionAtInitializationDiag

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	error
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Model Configuration Parameters: Stateflow Diagnostics” on page 10-2
- “Execution of a Chart at Initialization” (Stateflow)

- “Specify Properties for Stateflow Charts” (Stateflow)

Use of machine-parented data instead of Data Store Memory

Description

Select the diagnostic action to take when Stateflow detects machine-parented data that you can replace with chart-parented data of scope Data Store Memory.

Note Machine-parented data will no longer be supported in a future release. Use the Upgrade Advisor to convert machine-parented data to chart-parented data store memory. For more information, see “Consult the Upgrade Advisor” and “Check for machine-parented data”.

Category: Diagnostics

Settings

Default: error

none

No warning or error appears.

warning

A warning appears.

error

An error appears and stops the simulation.

Command-Line Information

Parameter: SFMachineParentedDataDiag

Value: 'none' | 'warning' | 'error'

Default: 'error'

Recommended Settings

Application	Setting
Debugging	error
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- “Best Practices for Using Data in Charts” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 10-2

- “Consult the Upgrade Advisor”
- “Check for machine-parented data”

Unreachable execution path

Description

Select the diagnostic action to take when there are chart constructs not on a valid execution path. These constructs can cause unreachable execution paths:

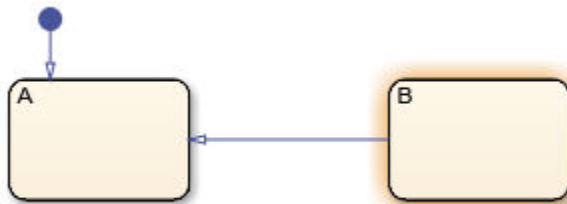
- Dangling transitions not connected to a destination state, junction, or port



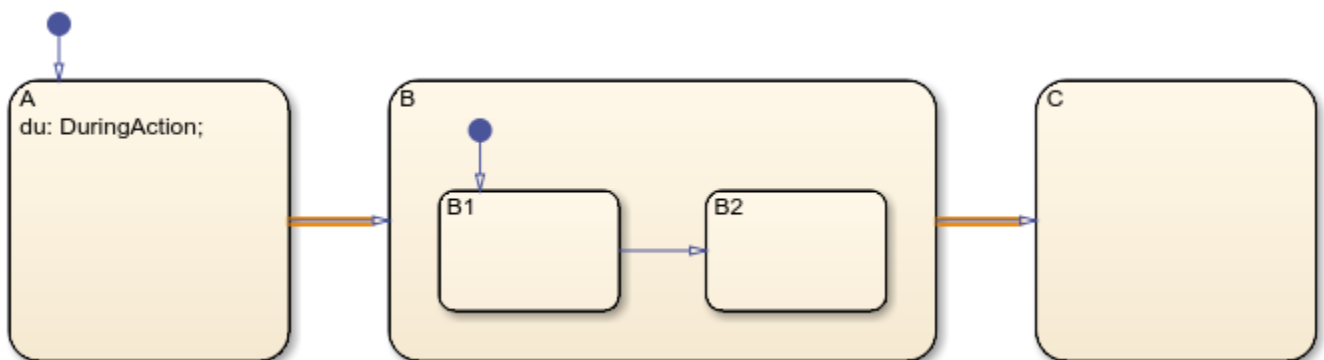
- Transition shadowing caused by an unconditional transition that prevents other transitions from the same source from executing



- States, junctions, or ports not connected with a transition from a reachable source



- Unconditional transitions leading out of a state that prevent the execution of the during actions in the state and the transitions between child states



Category: Diagnostics

Settings

Default: warning

none

No warning or error appears.

warning

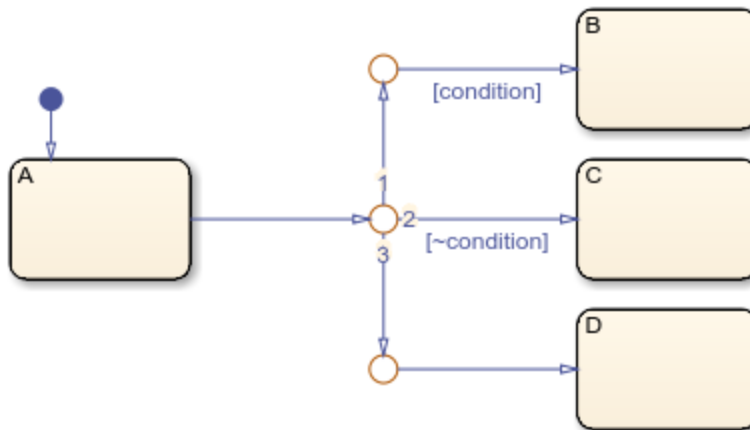
A warning appears.

error

An error appears and stops the simulation.

Tip

This diagnostic does not detect unreachable execution paths caused by transition conditions that are always true or false. For example, in this chart, the diagnostic does not detect that the unconditional transition to state D is never valid.



If you have Simulink Design Verifier, you can use dead logic detection to analyze your chart for this type of unreachable execution path. For more information, see “Dead Logic Detection” (Simulink Design Verifier).

Command-Line Information

Parameter: SFUnreachableExecutionPathDiag

Value: 'none' | 'warning' | 'error'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	warning
Traceability	No impact

Application	Setting
Efficiency	No impact (for simulation) none (for production code generation)
Safety precaution	error

See Also

Related Examples

- “Model Configuration Parameters: Stateflow Diagnostics” on page 10-2
- “Detect Modeling Errors During Edit Time” (Stateflow)
- “Dead Logic Detection” (Simulink Design Verifier)

Diagnostics Parameters: Type Conversion

Model Configuration Parameters: Type Conversion Diagnostics

The **Diagnostics > Type Conversion** category includes parameters for detecting issues related to data type conversions (for example, from `int32` to `single`).

Parameter	Description
“Unnecessary type conversions” on page 11-3	Select the diagnostic action to take when Simulink software detects a Data Type Conversion block used where no type conversion is necessary.
“Vector/matrix block input conversion” on page 11-4	Select the diagnostic action to take when Simulink software detects a vector-to-matrix or matrix-to-vector conversion at a block input.
“32-bit integer to single precision float conversion” on page 11-6	Select the diagnostic action to take if Simulink software detects a 32-bit integer value was converted to a floating-point value.
“Detect underflow” on page 11-7	Select the diagnostic action to take if Simulink software detects a 32-bit integer value was converted to a floating-point value.
“Detect precision loss” on page 11-9	Specifies diagnostic action to take when a fixed-point constant precision loss occurs during simulation.
“Detect overflow” on page 11-11	Specifies diagnostic action to take when a fixed-point constant overflow occurs during simulation.

See Also

Related Examples

- Diagnosing Simulation Errors
- “Data Types Supported by Simulink”
- Solver Diagnostics on page 9-2
- Sample Time Diagnostics on page 8-2
- Data Validity Diagnostics on page 6-2
- Connectivity Diagnostics on page 5-2
- Compatibility Diagnostics on page 4-2
- Model Referencing Diagnostics on page 7-2

Unnecessary type conversions

Description

Select the diagnostic action to take when Simulink software detects a Data Type Conversion block used where no type conversion is necessary.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

Command-Line Information

Parameter: UnnecessaryDatatypeConvMsg

Value: 'none' | 'warning'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	warning

See Also

Related Examples

- Diagnosing Simulation Errors
- Data Type Conversion
- “Model Configuration Parameters: Type Conversion Diagnostics” on page 11-2

Vector/matrix block input conversion

Description

Select the diagnostic action to take when Simulink software detects a vector-to-matrix or matrix-to-vector conversion at a block input.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

Simulink software converts vectors to row or column matrices and row or column matrices to vectors under the following circumstances:

- If a vector signal is connected to an input that requires a matrix, Simulink software converts the vector to a one-row or one-column matrix.
- If a one-column or one-row matrix is connected to an input that requires a vector, Simulink software converts the matrix to a vector.
- If the inputs to a block consist of a mixture of vectors and matrices and the matrix inputs all have one column or one row, Simulink software converts the vectors to matrices having one column or one row, respectively.

Command-Line Information

Parameter: VectorMatrixConversionMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	error

See Also

Related Examples

- Diagnosing Simulation Errors
- Determining Output Signal Dimensions
- “Model Configuration Parameters: Type Conversion Diagnostics” on page 11-2

32-bit integer to single precision float conversion

Description

Select the diagnostic action to take if Simulink software detects a 32-bit integer value was converted to a floating-point value.

Category: Diagnostics

Settings

Default: warning

none

Simulink software takes no action.

warning

Simulink software displays a warning.

Tip

Converting a 32-bit integer value to a floating-point value can result in a loss of precision. See Working with Data Types for more information.

Command-Line Information

Parameter: Int32ToFloatConvMsg

Value: 'none' | 'warning'

Default: 'warning'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	warning

See Also

Related Examples

- Diagnosing Simulation Errors
- Working with Data Types
- “Model Configuration Parameters: Type Conversion Diagnostics” on page 11-2

Detect underflow

Specifies diagnostic action to take when a fixed-point constant underflow occurs during simulation.

Description

Select the diagnostic action to take if Simulink software detects a 32-bit integer value was converted to a floating-point value.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- This diagnostic applies only to fixed-point constants (net slope and net bias).
- Fixed-point constant underflow occurs when Simulink software encounters a fixed-point constant whose data type does not have enough precision to represent the ideal value of the constant because the ideal value is too small.
- When fixed-point constant underflow occurs, casting the ideal value to the data type causes the value of the fixed-point constant to become zero, and therefore to differ from its ideal value.

Dependency

This parameter requires a Fixed-Point Designer license.

Command-Line Information

Parameter: FixptConstUnderflowMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact

Application	Setting
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- Net Slope and Net Bias Precision Issues (Fixed-Point Designer)
- “Model Configuration Parameters: Type Conversion Diagnostics” on page 11-2

Detect precision loss

Description

Specifies diagnostic action to take when a fixed-point constant precision loss occurs during simulation.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- This diagnostic applies only to fixed-point constants (net slope and net bias).
- Precision loss occurs when Simulink software converts a fixed-point constant to a data type which does not have enough precision to represent the exact value of the constant. As a result, the quantized value differs from the ideal value.
- Fixed-point constant precision loss differs from fixed-point constant overflow. Overflow occurs when the range of the parameter's data type, that is, the maximum value that it can represent, is smaller than the ideal value of the parameter.

Dependency

This parameter requires a Fixed-Point Designer license.

Command-Line Information

Parameter: FixptConstPrecisionLossMsg

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact

Application	Setting
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- Net Slope and Net Bias Precision Issues (Fixed-Point Designer)
- “Model Configuration Parameters: Type Conversion Diagnostics” on page 11-2

Detect overflow

Description

Specifies diagnostic action to take when a fixed-point constant overflow occurs during simulation.

Category: Diagnostics

Settings

Default: none

none

Simulink software takes no action.

warning

Simulink software displays a warning.

error

Simulink software terminates the simulation and displays an error message.

Tips

- This diagnostic applies only to fixed-point constants (net slope and net bias).
- Overflow occurs when the Simulink software converts a fixed-point constant to a data type whose range is not large enough to accommodate the ideal value of the constant. The ideal value is either too large or too small to be represented by the data type. For example, suppose that the ideal value is 200 and the converted data type is `int8`. Overflow occurs in this case because the maximum value that `int8` can represent is 127.
- Fixed-point constant overflow differs from fixed-point constant precision loss. Precision loss occurs when the ideal fixed-point constant value is within the range of the data type and scaling being used, but cannot be represented exactly.

Dependency

This parameter requires a Fixed-Point Designer license.

Command-Line Information

Parameter: `FixptConstOverflowMsg`

Value: 'none' | 'warning' | 'error'

Default: 'none'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact

Application	Setting
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- Net Slope and Net Bias Precision Issues (Fixed-Point Designer)
- “Model Configuration Parameters: Type Conversion Diagnostics” on page 11-2

Model Referencing Parameters

Model Configuration Parameters: Model Referencing

The **Model Referencing** pane of the Configuration Parameters dialog box allows you to specify options for:

- Including other models in this model.
- Including the current model in other models.

The option descriptions use the term *this model* to refer to the model that you are configuring and the term *referenced model* to designate models referenced by *this model*.

To open the Configuration Parameters dialog box for the top model in a model hierarchy, in the Simulink Toolstrip, on the **Modeling** tab, click **Model Settings**.

To open the Configuration Parameters dialog box for the current referenced model, on the **Modeling** tab, click the **Model Settings** button arrow, then select **Model Settings** in the **Referenced Model** section.

Parameter	Description
"Rebuild" on page 12-4	Select the method used to determine when to rebuild simulation and code generation targets for referenced models before updating, simulating, or generating code from this model.
"Never rebuild diagnostic" on page 12-9	Select the diagnostic action that Simulink software should take if it detects a model reference target that needs to be rebuilt.
"Enable parallel model reference builds" on page 12-11	Specify whether to use automatic parallel building of the model reference hierarchy whenever possible.
"MATLAB worker initialization for builds" on page 12-13	Specify how to initialize MATLAB workers for parallel builds.
"Enable strict scheduling checks for referenced models" on page 12-15	This parameter enables these checks for referenced models: <ul style="list-style-type: none"> • Scheduling order consistency of function-call subsystems in a referenced export function model • Sample time consistency across the boundary of a referenced export function model or referenced rate-based model
"Total number of instances allowed per top model" on page 12-16	Specify how many references to this model can occur in another model.
"Propagate sizes of variable-size signals" on page 12-27	Select how variable-size signals propagate through referenced models.
"Minimize algebraic loop occurrences" on page 12-20	Try to eliminate artificial algebraic loops from a model that involve the current referenced model

Parameter	Description
“Propagate all signal labels out of the model” on page 12-22	Pass propagated signal names to output signals of Model block.
“Pass fixed-size scalar root inputs by value for code generation” on page 12-18	Specify whether a model that references this model passes its scalar inputs to this model by value for code generation.
“Model dependencies” on page 12-29	Add user-created dependencies to the set of known target dependencies by using the Model dependencies parameter.
“Perform consistency check on parallel pool” on page 12-31	Specify whether to perform a consistency check on the parallel pool before starting a parallel build.
“Include custom code for referenced models” on page 2-121	Use custom code with Stateflow or with MATLAB Function blocks during model reference accelerator simulation.
“Use local solver when referencing model” on page 12-25	Speed up simulation in model references using local solvers

See Also

Model

Related Examples

- “Model Reference Basics”
- Model Dependencies

Rebuild

Description

Select the method to determine when to rebuild simulation and Simulink Coder targets for referenced models before updating, simulating, or generating code from the model.

Category: Model Referencing

Settings

Default: If any changes detected

Always

Always rebuild targets for referenced models. This setting requires the most processing time because it can trigger unnecessary builds. To make all model reference targets up to date, use this setting before you deploy a model.

If any changes detected

Conditionally rebuild targets for referenced models when Simulink detects a change that could affect simulation results. To perform extensive change detection on dependencies of referenced models, use this setting.

If Simulink finds no changes in known dependencies, it computes the structural checksum of the model. The structural checksum detects changes that occur in user-created dependencies that are not specified using the **Model dependencies** configuration parameter. If the structural checksum has changed, Simulink rebuilds the model reference target.

If any changes in known dependencies detected

Conditionally rebuild targets for referenced models when Simulink detects a change that could affect simulation results. To reduce the time required for change detection, use this setting.

If Simulink finds no changes in known or potential dependencies, it does *not* compute the structural checksum of the model and does *not* rebuild the model reference target. To avoid invalid simulation results, you must list all user-created dependencies in the **Model dependencies** parameter.

Never

Do not rebuild targets for referenced models. This setting requires the least processing time and, when available, uses Simulink cache files for faster simulations. To avoid rebuilds when developing a model, use this setting.

If model reference targets are out of date, the simulation may present invalid results. To have Simulink check for changes in known target dependencies and report if the model reference targets may be out of date, use the **Never rebuild diagnostic** parameter. To manually rebuild model reference targets, use the `slbuild` function.

For information on using and sharing Simulink cache files, see “Share Simulink Cache Files for Faster Simulation”.

Definitions

Known target dependencies

Known target dependencies are files and data external to model files that Simulink examines for changes when checking if a model reference target is up to date. Simulink automatically computes a set of known target dependencies. Examples of known target dependencies are:

- Changes to the model workspace, if its data source is a MAT-file or MATLAB file
- Enumerated type definitions
- User-written S-functions and their TLC files
- Files specified in the **Model dependencies** parameter
- External files used by Stateflow, a MATLAB Function block, or a MATLAB System block
- Dataflow subsystems - Analysis of dataflow subsystems requires that the simulation target rebuilds to profile and rebuilds again to partition the subsystem. In addition, the simulation target must rebuild if the machine running the simulation has fewer cores than the subsystem is partitioned to use, for example, if the simulation target was last built on a machine with a greater number of cores. For more information, see “Simulation of Dataflow Domains” (DSP System Toolbox).

Potential target dependencies

Potential target dependencies are files and data external to model files and model configuration settings that Simulink examines for changes when checking if a model reference target is up to date. Simulink automatically computes a set of potential target dependencies. Examples of potential target dependencies are:

- Changes to global variables
- Changes to targets of models referenced by this model
- The **Configuration Parameters > Diagnostics > Data Validity > Signal resolution** parameter when set to either `Explicit` and `implicit` or `Explicit` and `warn implicit`

Simulink examines each potential target dependency to determine whether its state triggers a structural checksum check.

User-created dependencies

User-created dependencies are files that Simulink does not automatically identify, in spite of their potential impact on simulation results. Examples of user-created dependencies are:

- MATLAB files that contain code executed by callbacks
- MAT-files that contain definitions for variables used by the model that are loaded as part of a customized initialization script

You can add user-created dependencies to the set of known target dependencies by using the **Model dependencies** parameter.

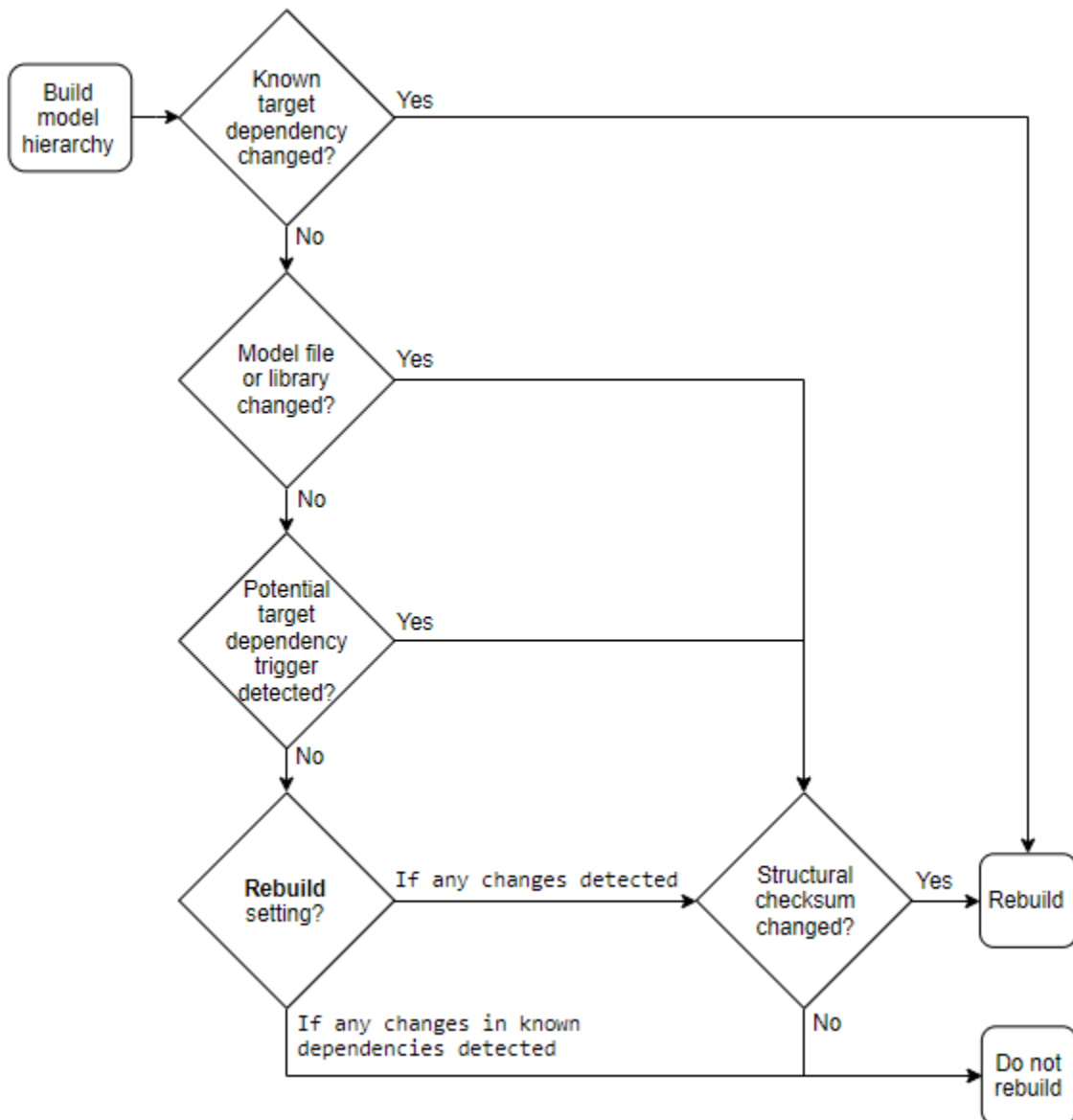
Structural checksum

A structural checksum is a computation used to detect changes in the model that can affect simulation results. When Simulink computes the structural checksum, it loads and compiles the model. To compile the model, Simulink must execute callbacks and access all variables that the model uses. The structural checksum detects changes in user-created dependencies, regardless of whether you have specified those user-created dependencies in the **Model dependencies** parameter.

For more information about the kinds of changes that affect the structural checksum, see `Simulink.BlockDiagram.getChecksum`.

Tips

- Models in a model hierarchy can have different rebuild settings. When you update, simulate, or generate code for a model, the rebuild setting for that model applies to all its referenced models.
- Models that execute in normal mode do not generate simulation targets and are unaffected by **Rebuild** settings.
- To improve rebuild detection speed and accuracy, use the **Model dependencies on page 12-29** configuration parameter to specify user-created dependencies.
- This flow chart describes the processing Simulink performs when you set **Rebuild** to either **If any changes detected** or **If any changes in known dependencies detected**.



- This example explains the difference between the `If any changes detected` and `If any changes in known dependencies detected` settings.

If you change a MATLAB file that is executed as part of a callback script that you have not listed in the **Model dependencies** parameter:

- `If any changes detected` causes a rebuild because the change affects the structural checksum of the model.
- `If any changes in known dependencies detected` does not cause a rebuild because no known target dependency has changed.

Dependency

Selecting `Never` enables the **Never rebuild diagnostic** on page 12-9 parameter.

Command-Line Information

Parameter: `UpdateModelReferenceTargets`

Value: `'Force' | 'IfOutOfDateOrStructuralChange' | 'IfOutOfDate' | 'AssumeUpToDate'`

Default: `'IfOutOfDateOrStructuralChange'`

UpdateModelReferenceTargets Value	Equivalent Rebuild Value
<code>'Force'</code>	Always
<code>'IfOutOfDateOrStructuralChange'</code>	If any changes detected
<code>'IfOutOfDate'</code>	If any changes in known dependencies detected
<code>'AssumeUpToDate'</code>	Never

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	If any changes detected or <code>Never</code> If you use the <code>Never</code> setting, then set the Never rebuild diagnostic parameter to <code>Error if rebuild required</code> .

Compatibility Considerations

Starting in R2019b, `If any changes detected` ignores cosmetic changes, such as repositioning a block.

See Also

Blocks

Model

Model Settings

Never rebuild diagnostic | Model dependencies

Functions

`Simulink.BlockDiagram.getChecksum`

Related Examples

- “Manage Simulation Targets for Referenced Models”
- “Share Simulink Cache Files for Faster Simulation”
- “Model Configuration Parameters: Model Referencing” on page 12-2

Never rebuild diagnostic

Description

Select the diagnostic action that Simulink software should take if it detects a model reference target that needs to be rebuilt.

Category: Model Referencing

Settings

Default: Error if rebuild required

none

Simulink takes no action.

Warn if rebuild required

Simulink displays a warning.

Error if rebuild required

Simulink terminates the simulation and displays an error message.

Tip

If you set the **Rebuild** parameter to **Never** and set the **Never rebuild diagnostic** parameter to **Error if rebuild required** or **Warn if rebuild required**, then Simulink:

- Performs the same change detection processing as for the **If any changes in known dependencies detected rebuild** option setting, except it does not compare structural checksums
- Issues an error or warning (depending on the **Never rebuild diagnostic** setting), if it detects a change
- Never rebuilds the model reference target

Selecting **None** bypasses dependency checking, and thus enables faster updating, simulation, and code generation. However, the **None** setting can cause models that are not up to date to malfunction or generate incorrect results. For more information on the dependency checking, see “Rebuild” on page 12-4.

Dependency

This parameter is enabled only if you select **Never** in the **Rebuild** field.

Command-Line Information

Parameter: CheckModelReferenceTargetMessage

Value: 'none' | 'warning' | 'error'

Default: 'error'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	Error if rebuild required

See Also

Blocks

Model

Model Settings

Rebuild

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Model Referencing” on page 12-2

Enable parallel model reference builds

Description

Specify whether to use automatic parallel building of the model reference hierarchy whenever possible.

Category: Model Referencing

Settings

Default: Off

On

Simulink software builds the model reference hierarchy in parallel whenever possible (based on computing resources and the structure of the model reference hierarchy).

Off

Simulink never builds the model reference hierarchy in parallel.

Dependency

Selecting this option enables the **MATLAB worker initialization for builds** parameter. Parallel building requires Parallel Computing Toolbox™.

Tip

You only need to set **Enable parallel model reference builds** for the top model of the model reference hierarchy to which it applies.

Command-Line Information

Parameter: EnableParallelModelReferenceBuilds

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Model

Related Examples

- “Reduce Update Time for Referenced Models by Using Parallel Builds”
- “Reduce Build Time for Referenced Models by Using Parallel Builds” (Simulink Coder)
- “Model Configuration Parameters: Model Referencing” on page 12-2

MATLAB worker initialization for builds

Description

Specify how to initialize MATLAB workers for parallel builds.

Category: Model Referencing

Settings

Default: None

None

Simulink software takes no action. Specify this value if the child models in the model reference hierarchy do not rely on anything in the base workspace beyond what they explicitly set up (for example, with a model load function).

Copy base workspace

Simulink attempts to copy the base workspace to each MATLAB worker. Specify this value if you use a setup script to prepare the base workspace for all models to use.

Load top model

Simulink loads the top model on each MATLAB worker. Specify this value if the top model in the model reference hierarchy handles all of the base workspace setup (for example, with a model load function).

Limitation

For values other than None, limitations apply to global variables in the base workspace. Global variables are not propagated across parallel workers and do not reflect changes made by top and child model scripts.

Dependency

Selecting the option **Enable parallel model reference builds** enables this parameter. Parallel building requires Parallel Computing Toolbox.

Command-Line Information

Parameter: ParallelModelReferenceMATLABWorkerInit

Value: 'None' | 'Copy Base Workspace' | 'Load Top Model'

Default: 'None'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact

Application	Setting
Efficiency	No impact
Safety precaution	No impact

See Also

Model

Related Examples

- “Reduce Update Time for Referenced Models by Using Parallel Builds”
- “Reduce Build Time for Referenced Models by Using Parallel Builds” (Simulink Coder)
- “Model Configuration Parameters: Model Referencing” on page 12-2

Enable strict scheduling checks for referenced models

Description

This parameter enables these checks for referenced models:

- Scheduling order consistency of function-call subsystems in referenced export function models
- Sample time consistency across the boundary of referenced export function models
- Sample time consistency across the boundary of referenced rate-based models that are function-call adapted.

Category: Model Referencing

Settings

Default: On

On

Simulink enforces strict checks on scheduling order and sample time consistency in referenced models.

Off

Simulink does not enforce strict checks on scheduling order and sample time consistency in referenced models.

Command-Line Information

Parameter: EnableRefExpFcnMdlSchedulingChecks

Value: 'on' | 'off'

Default: 'on'

See Also

Model

Related Examples

- “Export-Function Models Overview”
- “Sorting Rules for Explicitly Scheduled Model Components”
- “Model Configuration Parameters: Model Referencing” on page 12-2

Total number of instances allowed per top model

Description

Specify how many references to this model can occur in another model.

Category: Model Referencing

Settings

Default: Multiple

Zero

The model cannot be referenced. An error occurs if a reference to the model occurs in another model.

One

The model can be referenced at most once in a model reference hierarchy. An error occurs if more than one reference exists.

Multiple

The model can be referenced more than once in a hierarchy, provided that it contains no constructs that preclude multiple reference. An error occurs if the model cannot be referenced multiple times, even if only one reference exists.

To use multiple instances of a referenced model in normal mode, use the **Multiple** setting. For details, see “Simulate Multiple Referenced Model Instances in Normal Mode”.

Command-Line Information

Parameter: ModelReferenceNumInstancesAllowed

Value: 'Zero' | 'Single' | 'Multi'

Default: 'Multi'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Model

Related Examples

- Diagnosing Simulation Errors

- “Model Configuration Parameters: Model Referencing” on page 12-2

Pass fixed-size scalar root inputs by value for code generation

Description

Specify whether a model that references this model passes its scalar inputs to this model by value.

Category: Model Referencing

Settings

Default: Off (GUI), 'on' (command-line)

On

A model that references this model passes scalar inputs to this model by value.

Off

The parent model passes the inputs by reference (it passes the addresses of the inputs rather than the input values).

Tips

- This option is ignored in either of these two cases:
 - The C function prototype control is not the default.
 - The C++ encapsulation interface is not the default.
- Passing root inputs by value allows this model to read its scalar inputs from register or local memory, which is faster than reading the inputs from their original locations.
- Enabling this parameter can result in the simulation behavior differing from the generated code behavior under certain modeling semantics. Simulink reports cases where the modeling semantics may result in inconsistent behaviors for simulation and for generated code. If the diagnostic identifies an issue, latch the function-call subsystem inputs. For more information about latching function-call subsystems, see “Context-dependent inputs” on page 5-17.
- If the Context-dependent inputs diagnostic reports no issues for a model, consider enabling the **Pass fixed-size scalar root inputs by value for code generation** parameter, which usually generates more efficient code for such a model.
- If you have a Simulink Coder license, selecting this option can affect reuse of code generated for subsystems. See “Generate Reentrant Code from Subsystems” (Simulink Coder) for more information.
- For SIM targets, a model that references this model passes inputs by reference, regardless of how you set the **Pass fixed-size scalar root inputs by value for code generation** parameter.

Command-Line Information

Parameter: ModelReferencePassRootInputsByReference

Value: 'on' | 'off'

Default: 'on'

Note The command-line values are reverse of the settings values. Therefore, 'on' in the command line corresponds to the description of "Off" in the settings section, and 'off' in the command line corresponds to the description of "On" in the settings section.

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

For the diagnostic action to take when the software has to compute the input to a function-call subsystem, see "Context-dependent inputs" on page 5-17.

See Also

Model

Related Examples

- "Using Function-Call Subsystems"
- "Generate Reentrant Code from Subsystems" (Simulink Coder)
- "Model Configuration Parameters: Model Referencing" on page 12-2

Minimize algebraic loop occurrences

Description

Try to eliminate artificial algebraic loops from a model that involve the current referenced model.

Category: Model Referencing

Settings

Default: Off

On

Simulink software tries to eliminate artificial algebraic loops from a model that involve the current referenced model.

Off

Simulink software does not try to eliminate artificial algebraic loops from a model that involve the current referenced model.

Tips

Enabling this parameter together with the Simulink Coder **Single output/update function** parameter results in an error.

Command-Line Information

Parameter: ModelReferenceMinAlgLoopOccurrences

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Model

Related Examples

- “Algebraic Loop Concepts”
- “Model Blocks and Direct Feedthrough”

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Model Referencing” on page 12-2

Propagate all signal labels out of the model

Description

Pass propagated signal names to output signals of Model block.

Category: Model Referencing

Settings

Default: On

On

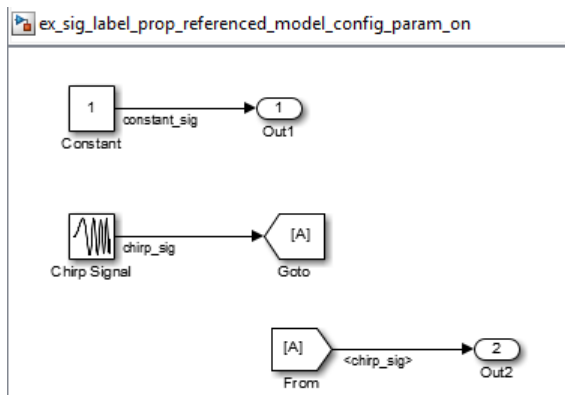
Simulink propagates signal names to output signals of the Model block.

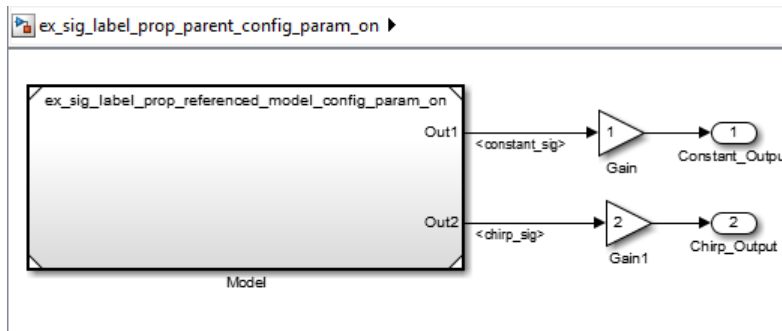
Off

Simulink does not propagate signal names to output signals of the Model block.

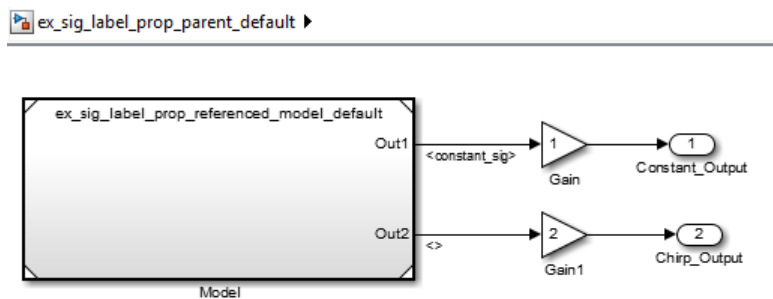
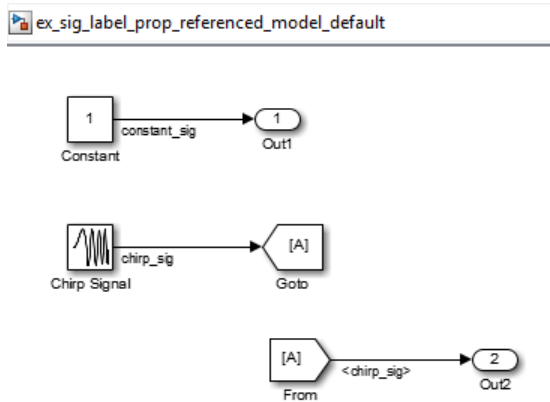
Tips

- By default, each instance of a referenced model propagates signal labels. Clear the setting for any instance that you do not want to propagate signal labels.
- The following models illustrate the behavior when you use the default setting of the **Propagate all signal labels out of the model** parameter of enabled for the referenced model. The output signal from the Model block Out2 port displays the propagated signal name (`chirp_sig`), whose source is inside the referenced model.





- The following models illustrate the behavior when you clear this parameter, if you enable signal label propagation for every eligible signal. Inside the referenced model, signal label propagation occurs as in any model. However, the output signal from the Model block Out2 port displays empty brackets for the propagated signal label.



Command-Line Information

Parameter: PropagateSignalLabelsOutOfModel

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Model

Related Examples

- “Signal Label Propagation”
- “Model Configuration Parameters: Model Referencing” on page 12-2

Use local solver when referencing model

Description

Improve simulation performance in model references.

Category: Model Referencing

Settings

Default: Off

On

Continuous states in the model are solved independently from the top by the solver specified in the referenced **Configuration Parameters**.

Off

Continuous states in the model are solved by the top solver.

Tips

- Normally, Simulink uses one solver for the entire model and the solver specified by a referenced model's configuration set is ignored. When this parameter is enabled, the model's continuous states will be solved separately from the top model's solver. The local solver used follows the solver settings in the configuration set of the referenced model.
- When this parameter is on, the software tries to speed up simulation by using inexpensive solvers on slow states of the current referenced model.

Command-Line Information

Parameter: UseModelRefSolver

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	Off
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No impact

See Also

Model | "Local Solver Basics"

Related Examples

- “Model Configuration Parameters: Model Referencing” on page 12-2

Propagate sizes of variable-size signals

Description

Select how variable-size signals propagate through referenced models.

Category: Model Referencing

Settings

Default: Infer from blocks in model

Infer from blocks in model

Searches a referenced model and groups blocks into the following categories.

Category	Description	Example Blocks in This Category
1	Output signal size depends on input signal values.	<ul style="list-style-type: none"> Switch block Enabled Subsystem block with an Enable block that sets Propagate sizes of variable-size signals to During execution
2	States require resetting when the input signal size changes.	<ul style="list-style-type: none"> Unit Delay block Enabled Subsystem block with an Enable block that sets Propagate sizes of variable-size signals to Only when enabling
3	Output signal size depends on only the input signal size.	Gain block

The search stops at the boundary of enable, function-call, and action subsystems because these subsystems can specify when to propagate the size of a variable-size signal.

Simulink sets the propagation of variable-size signals for a referenced model as follows:

Referenced Model Contents	Referenced Model Propagation of Variable-Size Signals
One or more blocks in category 1, and all other blocks in category 3	Supports During execution .
One or more blocks in category 2, and all other blocks in category 3	Supports Only when enabling.
Blocks in category 1 and category 2	Errors.
All blocks in category 3 with at least one conditionally executed subsystem that is not an enable, function-call, or action subsystem	Errors. In this case, Simulink cannot determine when to propagate sizes of variable-size signals.

Referenced Model Contents	Referenced Model Propagation of Variable-Size Signals
All blocks in category 3 with only conditionally executed subsystems that are enable, function-call, or action subsystems	Supports both Only with enabling and During execution.

Only when enabling

Propagates sizes of variable-size signals for the referenced model only when enabling (at Enable method).

During execution

Propagates sizes of variable-size signals for the referenced model during execution (at Outputs method).

Command-Line Information

Parameter: PropagateVarSize

Value: 'Infer from blocks in model' | 'Only when enabling' | 'During execution'

Default: 'Infer from blocks in model'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Model

Related Examples

- “Model Configuration Parameters: Model Referencing” on page 12-2

Model dependencies

Description

Model dependencies are files and data that potentially impact simulation results. Simulink does not automatically identify user-created dependencies. Examples of user-created dependencies are:

- MATLAB files that contain code executed by callbacks
- MAT-files that contain definitions for variables used by the model that are loaded as part of a customized initialization script

To avoid invalid simulation results, list all user-created dependencies in the **Model dependencies** parameter. When determining whether a model reference target is up to date, Simulink examines dependencies that it automatically identifies and files specified by the **Model dependencies** parameter.

Category: Model Referencing

Settings

Default: ''

Specify dependencies as a cell array of character vectors, where each cell array entry is one of the following:

- File name — Simulink looks on the MATLAB path for a file with the given name. If the file is not on the MATLAB path, specify the path to the dependent file. The file name must include a file extension, such as `.m` or `.mat`.
- Path to the dependent file — The path can be relative or absolute, and must include the file name.
- Folder — Simulink treats every file in that folder as a dependent file. Simulink does not include files of subfolders of the folder you specify.

Cell array entries can include:

- Spaces
- The token `$MDL` as a prefix to a dependency to indicate that the path to the dependency is relative to the location of this model file
- An asterisk (`*`) as a wild card
- A percent sign (`%`) to comment out a line
- An ellipsis (`...`) to continue a line

For example:

```
{'D:\Work\parameters.mat', '$MDL\mdlvars.mat', ...
'D:\Work\masks\*.m'}
```

Tips

- To improve rebuild detection speed and accuracy, use the **Model dependencies** parameter to specify user-created dependencies when the **Rebuild** on page 12-4 parameter is set to either **If any changes detected** or **If any changes in known dependencies detected**.
- To prevent invalid simulation results, if the **Rebuild** setting is **If any changes in known dependencies detected**, add every user-created dependency.
- To help identify model dependencies, use the Dependency Analyzer. For more information, see “Analyze Model Dependencies”.
- If Simulink cannot find a specified dependent file when you update or simulate a model that references this model, Simulink displays a warning.
- The dependencies automatically include the model and linked library files, so you do not need to specify those files with the **Model dependencies** parameter.

Command-Line Information

Parameter: ModelDependencies

Type: character vector

Value: any valid value

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Blocks

Model

Model Settings

Rebuild

Related Examples

- “Model Configuration Parameters: Model Referencing” on page 12-2

Perform consistency check on parallel pool

Description

Specify if you want the Simulink software to perform checks on the parallel pool before starting a parallel build.

The software performs these criteria checks:

- The pool is spmd compatible.
- The platform is consistent between workers and client.
- The workers have a Simulink Coder license.
- The workers have write access to the current working folder.

Category: Model Referencing

Settings

Default: On

On

If a check fails, the parallel build stops, producing an error.

Off

If a check fails, a warning is issued and a serial build is performed.

Command-Line Information

Parameter: ParallelModelReferenceErrorOnInvalidPool

Value: 'on' | 'off'

Default: 'on'

See Also

Related Examples

- “Reduce Update Time for Referenced Models by Using Parallel Builds”
- “Reduce Build Time for Referenced Models by Using Parallel Builds” (Simulink Coder)
- “Model Configuration Parameters: Model Referencing” on page 12-2

Simulation Target Parameters

Model Configuration Parameters: Simulation Target

The **Simulation Target** category includes parameters for configuring the simulation target for a model. In the Configuration Parameters dialog box, the following parameters are in the **Simulation Target** pane.

Parameter	Description	Location
"GPU acceleration" on page 13-8	Specify whether or not to accelerate MATLAB Function blocks on NVIDIA® GPUs. This option requires a GPU Coder™ license.	
"Language" on page 13-6	Specify C or C++ code generation for simulation targets.	
"Include headers" on page 13-11	Specify interface header code containing types and function declarations to import into Simulink.	Code information tab
"Include directories" on page 13-14	Specify directories containing header and source files.	Code information tab
"Source files" on page 13-16	Specify custom code source files.	Code information tab
"Libraries" on page 13-17	Specify a list of static and/or shared libraries that contain custom object code to link into the target.	Code information tab
"Defines" on page 13-19	Specify preprocessor macro definitions to be added to the compiler command line.	Code information tab
"Compiler flags" on page 13-20	Specify additional flags to be added to the compiler command line.	Code information tab
"Linker flags" on page 13-21	Specify additional flags to be added to the linker command line.	Code information tab
"Initialize code" on page 13-12	Specify C/C++ code to execute at the start of simulation.	Additional source code tab
"Terminate code" on page 13-13	Specify C/C++ code to execute at the end of simulation.	Additional source code tab
"Additional code" on page 13-10	Specify additional custom code to import into Simulink.	Additional source code tab
"Simulate custom code in a separate process" on page 13-30	Run custom code in a separate process outside of MATLAB during model simulation.	Import settings tab

Parameter	Description	Location
“Enable custom code analysis” on page 13-9	Specify whether or not to enable Simulink Coverage and Simulink Design Verifier support for custom code.	Import settings tab
“Enable global variables as function interfaces” on page 13-32	Specify the behavior of global variables in custom code called by the C Caller block.	Import settings tab
“Undefined function and variable handling” on page 13-28	Specify undefined function behavior for all external C functions called by C Caller, MATLAB Function, MATLAB System blocks or Stateflow charts.	Import settings tab
“Deterministic functions” on page 13-22	Specify whether custom code functions are deterministic.	Import settings tab
“Specify by function” on page 13-24	Specify which custom code functions are deterministic.	Import settings tab
“Default function array layout” on page 13-26	Specify the default array layout for all external C functions used by the C Caller block.	Import settings tab
“Exception by function” on page 13-33	Specify the array layout for each external C function used by the C Caller block.	Import settings tab
“Target library” (Simulink Coder)	Specify the target deep learning library to use for simulation. MKL - DNN requires a Simulink Coder license. cuDNN or TensorRT requires a GPU Coder license.	
“Auto tuning” (Simulink Coder)	Use auto tuning for cuDNN library. Enabling auto tuning allows the cuDNN library to find the fastest convolution algorithms. This parameter requires Simulink Coder and GPU Coder licenses.	

These configuration parameters are in the **Advanced parameters** section.

Parameter	Description
“Import custom code” on page 2-79	Specify whether or not to parse available custom code variables and functions and compile custom code into its own simulation target.

Parameter	Description
"Echo expressions without semicolons" on page 2-100	Enable run-time output in the MATLAB Command Window, such as actions that do not terminate with a semicolon.
Break on Ctrl+C on page 2-92	Enables responsiveness checks in code generated for MATLAB Function blocks, Stateflow charts, and dataflow domains.
"Generate typedefs for imported bus and enumeration types" on page 2-109	Determines typedef handling and generation for imported bus and enumeration data types in Stateflow and MATLAB Function blocks.
Enable memory integrity checks on page 2-107	Detects violations of memory integrity in code generated for MATLAB Function blocks and stops execution with a diagnostic.
"Enable run-time recursion for MATLAB functions" on page 2-95	Allow recursive functions in code that is generated for MATLAB code that contains recursive functions.
"Enable implicit expansion in MATLAB functions" on page 2-94	Enable implicit expansion in code that is generated for MATLAB code that contains binary operations and functions.
"Compile-time recursion limit for MATLAB functions" on page 2-93	For compile-time recursion, control the number of copies of a function that are allowed in the generated code.
"Block reduction" on page 2-86	Reduce execution time by collapsing or removing groups of blocks.
"Compiler optimization level" on page 2-81	Sets the degree of optimization used by the compiler when generating code for acceleration.
"Hardware acceleration" on page 2-122	Select whether or not to use hardware acceleration and the level of hardware acceleration.
"Conditional input branch execution" on page 2-90	Improve model execution when the model contains Switch and Multiport Switch blocks.
"Verbose accelerator builds" on page 2-83	Select the amount of information displayed during code generation for Simulink Accelerator mode, referenced model Accelerator mode, and Rapid Accelerator mode.
"Dynamic memory allocation in MATLAB functions" on page 2-96	Use dynamic memory allocation (malloc) for variable-size arrays whose size (in bytes) is greater than or equal to the dynamic memory allocation threshold. This parameter applies to MATLAB code in a MATLAB Function block, a Stateflow chart, or a System object associated with a MATLAB System block.

Parameter	Description
"Dynamic memory allocation threshold in MATLAB functions" on page 2-98	Use dynamic memory allocation (malloc) for variable-size arrays whose size (in bytes) is greater than or equal to a threshold. This parameter applies to MATLAB code in a MATLAB Function block, a Stateflow chart, or a System object associated with a MATLAB System block.
"Enable continuous-time MATLAB functions to write to initialized persistent variables" on page 2-101	Enable continuous-time MATLAB functions to write to initialized persistent variables. If disabled, continuous-time MATLAB functions can only initialize and read persistent variables.
"Allow setting breakpoints during simulation" on page 2-103	Enable adding breakpoints in MATLAB Function blocks, Stateflow charts, State Transition blocks, and Truth Table blocks during simulation.
"Reserved names" on page 2-105	Enter the names of variables or functions in the generated code that match the names of variables or functions specified in custom code for a model that contains MATLAB Function blocks, Stateflow charts, or Truth Table blocks.

See Also

Related Examples

- "What Is Acceleration?"
- "How Acceleration Modes Work"
- "Determine Why Simulink Accelerator Is Regenerating Code"
- "Manage Simulation Targets for Referenced Models"
- "Speed Up Simulation" (Stateflow)

Language

Description

Specify C or C++ code generation for simulation targets. This configuration parameter affects:

- Model reference simulation targets.
- Rapid accelerator simulation targets.
- Custom code you implement with C Function blocks, C Caller blocks, MATLAB Function blocks, MATLAB System blocks, and Stateflow charts. If **Import custom code** is selected, available custom code variables and functions are parsed.

The **Simulation cache folder** configuration parameter determines where to save the generated C or C++ files.

Category: Simulation Target

Settings

Default: C

C

Generates C code for simulation targets.

C++

Generates C++ code for simulation targets.

Select the C++ option to:

- Generate MATLAB Function block or Stateflow chart MEX code as C++ files and compile code using C++. For MATLAB Function and MATLAB System blocks, if you add C++ code to `buildInfo` using `coder.updateBuildInfo` or `coder.ExternalDependency`, set **Language** to C++.
- Simulate a MATLAB System block through code generation and compilation using C++.
- Use a C Function block to interface with C++ classes defined in your custom code. See “Interface with C++ Classes Using C Function Block”.

A model cannot have a C++ simulation target if it contains a Simscape block with a source file that contains MATLAB functions.

Before you build a system, configure Simulink to use a compiler system using `mex -setup`.

Command-Line Information

Parameter: `SimTargetLang`

Value: 'C' | 'C++'

Default: 'C'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Model Configuration Parameters: Simulation Target” on page 13-2
- “Manage Simulation Targets for Referenced Models”

GPU acceleration

Description

Speed up the execution of MATLAB Function block on NVIDIA GPUs by generating CUDA[®] code. This option requires a GPU Coder license.

Category: Simulation Target

Settings

Default: Off

On

Enables simulation acceleration by using GPU Coder. When this option is on, the software generates CUDA MATLAB executable (MEX) code from the block and dynamically links the generated code to Simulink during simulation.

Off

Disables simulation acceleration by using GPU Coder. When this option is off, Simulink uses the interpreted code mode normally used in simulations.

Command-Line Information

Parameter: GPUAcceleration

Type: character vector

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	On
Traceability	No impact
Efficiency	No impact
Safety precaution	On

See Also

Related Examples

- “Model Configuration Parameters: Simulation Target” on page 13-2

Enable custom code analysis

Description

Specify whether or not to enable Simulink Coverage and Simulink Design Verifier support for custom code. This option affects the C Caller block, the C Function block, the MATLAB Function block, the MATLAB System block, and Stateflow charts.

Category: Simulation Target

Settings

Default: Off

On

Enables Simulink Coverage and Simulink Design Verifier support for custom code.

Off

Disables Simulink Coverage and Simulink Design Verifier support for custom code.

Command-Line Information

Parameter: SimAnalyzeCustomCode

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No recommendation
Traceability	No recommendation
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Coverage for Custom C/C++ Code in Simulink Models” (Simulink Coverage)

Additional code

Description

Specify additional custom code to import into Simulink.

Category: Simulation Target

Settings

Default: ''

Command-Line Information

Parameter: SimCustomSourceCode

Type: character vector

Value: any C code

Default: ''

Recommended Settings

Application	Setting
Debugging	No recommendation
Traceability	No recommendation
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Reuse Custom Code in Stateflow Charts” (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Include headers

Description

Specify interface header code containing types and function declarations to import into Simulink.

Category: Simulation Target

Settings

Default: ''

Tips

- When you include a custom header file, enclose the file name in double quotes. For example, `#include "sample_header.h"` is a valid declaration for a custom header file.
- You can include extern declarations of variables or functions.

Command-Line Information

Parameter: SimCustomHeaderCode

Type: character vector

Value: any C code

Default: ''

Recommended Settings

Application	Setting
Debugging	No recommendation
Traceability	No recommendation
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Reuse Custom Code in Stateflow Charts” (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Initialize code

Description

Specify C/C++ code to execute at the start of simulation.

Category: Simulation Target

Settings

Default: ''

Tip

- Use this code to invoke functions that allocate memory or to perform other initializations of your custom code.

Command-Line Information

Parameter: SimCustomInitializer

Type: character vector

Value: any C code

Default: ''

Recommended Settings

Application	Setting
Debugging	No recommendation
Traceability	No recommendation
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Reuse Custom Code in Stateflow Charts” (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Terminate code

Description

Specify C/C++ code to execute at the end of simulation.

Category: Simulation Target

Settings

Default: ''

Tip

- Use this code to invoke functions that free memory allocated by the custom code or to perform other cleanup tasks.

Command-Line Information

Parameter: SimCustomTerminator

Type: character vector

Value: any C code

Default: ''

Recommended Settings

Application	Setting
Debugging	No recommendation
Traceability	No recommendation
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Reuse Custom Code in Stateflow Charts” (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Include directories

Description

Specify directories containing header and source files.

Category: Simulation Target

Settings

Default: ' '

Enter a space-separated list of folder paths.

- Specify absolute or relative paths to the directories.
- Relative paths must be relative to the folder containing your model files, not relative to the build folder.
- The order in which you specify the directories is the order in which they are searched for header, source, and library files.

Note If you specify a Windows® path containing one or more spaces, you must enclose the character vector in double quotes. For example, the second and third paths in the **Include directories** entry below must be double-quoted:

```
C:\Project "C:\Custom Files" "C:\Library Files"
```

If you set the equivalent command-line parameter `SimUserIncludeDirs`, each path containing spaces must be separately double-quoted within the single-quoted third argument, for example,

```
>> set_param('mymodel', 'SimUserIncludeDirs', ...  
            'C:\Project "C:\Custom Files" "C:\Library Files"')
```

Command-Line Information

Parameter: `SimUserIncludeDirs`

Type: character vector

Value: any folder path

Default: ' '

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- “Reuse Custom Code in Stateflow Charts” (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Source files

Description

Specify custom code source files.

Category: Simulation Target

Settings

Default: ' '

You can separate source files with a comma, a space, or a new line.

Limitation

This parameter does not support Windows file names that contain embedded spaces.

Tip

- The file name is sufficient if the file is in the current MATLAB folder or in one of the directories specified in **Include directories**.

Command-Line Information

Parameter: SimUserSources

Type: character vector

Value: any file name

Default: ' '

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- “Reuse Custom Code in Stateflow Charts” (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Libraries

Description

Specify a list of static libraries and/or shared libraries that contain custom object code to link into the target.

Category: Simulation Target

Settings

Default: ' '

Enter a space-separated list of library files.

Limitation

This parameter does not support Windows file names that contain embedded spaces.

Tips

- The file name is sufficient if the file is in the current MATLAB folder or in one of the directories specified in **Include directories**.
- If you are using a shared library (DLL) file on a Windows system, the DLL file must be located in the same drive as the build directory, which must be a local (non-network) drive.

Command-Line Information

Parameter: SimUserLibraries

Type: character vector

Value: any library file name

Default: ' '

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- “Reuse Custom Code in Stateflow Charts” (Stateflow)

- “Model Configuration Parameters: Simulation Target” on page 13-2

Defines

Description

Specify preprocessor macro definitions to be added to the compiler command line.

Category: Simulation Target

Settings

Default: ''

Enter a list of macro definitions for the compiler command line. Specify the parameters with a space-separated list of macro definitions. If a makefile is generated, these macro definitions are added to the compiler command line in the makefile. The list can include simple definitions (for example, `-DDEF1`), definitions with a value (for example, `-DDEF2=1`), and definitions with a space in the value (for example, `-DDEF3="my value"`). Definitions can omit the `-D` (for example, `-DF00=1` and `F00=1` are equivalent). If the toolchain uses a different flag for definitions, the code generator overrides the `-D` and uses the appropriate flag for the toolchain.

Command-Line Information

Parameter: SimUserDefines

Type: character vector

Value: preprocessor macro definition

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- "Integrate External Code by Using Model Configuration Parameters" (Simulink Coder)
- "Model Configuration Parameters: Simulation Target" on page 13-2

Compiler flags

Description

Specify additional flags to be added to the compiler command line.

Category: Simulation Target

Settings

Default: ''

Enter a list of additional flags for the compiler command line. Specify the flags with a space-separated list. If a makefile is generated, these flags are added to the compiler command line in the makefile. Acceptable flag content and syntax depend on the compiler being used.

Examples: -Zi -Wall, -O3, -w

Command-Line Information

Parameter: SimCustomCompilerFlags

Type: character vector

Value: compiler flags

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- “Integrate External Code by Using Model Configuration Parameters” (Simulink Coder)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Linker flags

Description

Specify additional flags to be added to the linker command line.

Category: Simulation Target

Settings

Default: ''

Enter a list of flags for the linker command line. Specify the flags with a space-separated list. If a makefile is generated, these macro definitions are added to the linker command line in the makefile. Acceptable flag content and syntax depend on the linker being used.

Examples: -T, -MD -Gy

Command-Line Information

Parameter: SimCustomLinkerFlags

Type: character vector

Value: linker flags

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- “Integrate External Code by Using Model Configuration Parameters” (Simulink Coder)
- “Model Configuration Parameters: Simulation Target” on page 13-2

Deterministic functions

Description

Specify which custom code functions are deterministic, that is, always producing the same outputs for the same inputs. If a custom code function is specified as deterministic, then a C Caller or C Function block that calls that function can be used in a For Each subsystem or with continuous sample time, and the block is optimized for use in conditional input branch execution. When a block is optimized for use in conditional input branch execution, it is executed only if it is in the active branch of a Switch or Multiport Switch block, both in simulation and in generated code. See **Conditional input branch execution**. This parameter is enabled only if **Import custom code** is selected.

Category: Simulation Target

Settings

Default: None

None

None of the custom code functions are deterministic.

All

All of the custom code functions are deterministic.

By function

The custom code functions that are deterministic are listed in “Specify by function” on page 13-24.

Note If a C Function block references any custom code global variables in its code, then this parameter must be set to All in order for the block to be used in a For Each subsystem or with continuous sample time, or to be optimized for use in conditional input branch execution.

Command-Line Information

Parameter: DefaultCustomCodeDeterministicFunctions

Type: character vector

Value: 'None' | 'All' | 'ByFunction'

Default: 'None'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Model Configuration Parameters: Simulation Target” on page 13-2
- C Caller
- C Function
- For Each Subsystem
- “Conditional input branch execution” on page 2-90

Specify by function

Description

Specify which custom code functions are deterministic, that is, always producing the same outputs for the same inputs. This parameter is enabled only when the “Deterministic functions” on page 13-22 parameter is set to **By Function**. If a custom code function is specified as deterministic, then a C Caller or C Function block that calls that function can be used in a For Each subsystem or with continuous sample time, and the block is optimized for use in conditional input branch execution. When a block is optimized for use in conditional input branch execution, it is executed only if it is in the active branch of a Switch or Multiport Switch block, both in simulation and in generated code. See **Conditional input branch execution**.

Category: Simulation Target

Settings

Specify the names of custom code functions that are deterministic, that is, always producing the same outputs for the same inputs.



Add

Add a name to the list of custom code deterministic functions.



Remove

Remove a name from the list of custom code deterministic functions.

Tip If you do not see a list of your custom code functions in the **Specify by function** dialog, close the dialog, click **Validate custom code**, and click **Specify by function** again.

Command-Line Information

Parameter: CustomCodeDeterministicFunctions

Type: character vector

Value: names of custom code functions, separated by commas

Default: ''

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Model Configuration Parameters: Simulation Target” on page 13-2
- C Caller
- C Function
- For Each Subsystem
- “Conditional input branch execution” on page 2-90

Default function array layout

Description

Specify how input array data is handled by external C/C++ functions and class methods. This parameter affects C/C++ functions and methods called by C Caller, C Function, MATLAB Function, and MATLAB System blocks and Stateflow charts.

Category: Simulation Target

Settings

Default: Not specified

Column-major

External C/C++ functions and class methods assume input array data is in column-major layout.

Row-major

External C/C++ functions and class methods assume input array data is in row-major layout.

Any

External C/C++ functions and class methods are indifferent about input data array layout. If your external function and class method algorithms do not require the matrix data to be in a specific array layout, for example if they perform only element-wise operations on input array data, use this option.

Not specified

External C/C++ functions and class methods make no assumption about input data array layout. However, if **Array layout** (Simulink Coder) is set to Row-major, Simulink reports an error. You can turn off the error by changing the **External functions compatibility for row-major code generation** (Simulink Coder) to warning or none.

The **Default function array layout** parameter controls the default array layout of custom code functions and class methods. To specify array layout for individual functions or class methods, click **Exception by function** on page 13-33.

Command-Line Information

Parameter: DefaultCustomCodeFunctionArrayLayout

Type: character vector

Value: 'Column-major' | 'Row-major' | 'Any' | 'NotSpecified'

Default: 'NotSpecified'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation

Application	Setting
Safety precaution	No recommendation

See Also

Related Examples

- “Integrate External Code by Using Model Configuration Parameters” (Simulink Coder)
- “Model Configuration Parameters: Simulation Target” on page 13-2
- “Integrate C Code Using C Caller Blocks”

Undefined function and variable handling

Description

Specify the behavior of undefined functions and variables in the C source code file of a model that contains Stateflow charts or C Caller, C Function, MATLAB Function, or MATLAB System blocks. If you declare a function or variable in the header file but do not implement it in the source code, Simulink behaves according to this setting. Depending on the setting, Simulink takes functions and variables from the header file and creates stub functions and variables equal to zero if they are not defined in the C source code file. If your code is not compatible with desktop simulation, or you want to introduce the interface to external code with the relevant header files, set this parameter to **Use interface only**.

Category: Simulation Target

Settings

Default: Filter out

Throw error

Return an error if a function or variable in the C source code is undefined. Simulink does not generate stub functions or variables equal to zero, but shows the functions and variables in the **Port Specification** table of the C Caller block.

Filter out

Filter out undefined functions and variables in the C source code. Simulink does not automatically generate stub functions or variables equal to zero, and the **Port Specification** table of the C Caller block does not show these functions and variables.

If you have undefined functions or variables in the C source code and a model that contains a Stateflow chart, MATLAB Function, or MATLAB System block calls these functions or variables, Simulink returns an error. If the custom code in the blocks in your model has undefined functions or variables, Simulink displays a warning.

Do not detect

Do not detect undefined functions or variables in the source code. Simulink does not automatically generate stub functions or variables equal to zero, but shows the functions and variables in the **Port Specification** table of the C Caller block.

Use interface only

Detect undefined functions and variables in the C source code. Simulink generates stub functions and variables equal to zero, makes them visible in the model, and allows you to call them from Stateflow charts and C Function, MATLAB Function, and MATLAB System blocks.

Command-Line Information

Parameter: CustomCodeUndefinedFunction

Value: 'ThrowError' | 'FilterOut' | 'DoNotDetect' | 'UseInterfaceOnly'

Default: 'FilterOut'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Integrate External Code by Using Model Configuration Parameters” (Simulink Coder)
- “Model Configuration Parameters: Simulation Target” on page 13-2
- C Caller

Simulate custom code in a separate process

Description

Run custom code in a separate process outside of MATLAB during model simulation. This option applies to external code integrated into the model using C Caller, C Function, MATLAB Function, and MATLAB System blocks and Stateflow charts.

Category: Simulation Target

Settings

Default: Off

On

Custom code runs in a separate process during model simulation, thus preventing a MATLAB crash due to unexpected exceptions in the custom code or errors in the interface between Simulink and the custom code. A run-time exception in the custom code produces an error message in Simulink that provides detailed information about the exception, such as which block or line number is responsible, to help resolve any issues with the code. If a supported external debugger is installed, the error message provides a button to launch the external debugger. For more information, see “Debug Custom C/C++ Code”.

Selecting this parameter allows you to map a pointer type argument of a custom code C function to an output port of a C Caller block without explicitly specifying its size in the **Port Specification** table of the block. See “Map C Function Arguments to Simulink Ports”.

Off

Custom code runs in the same process as the rest of the model during simulation. Simulation usually runs faster, but a run-time exception in the custom code could cause MATLAB to crash.

Command-Line Information

Parameter: SimDebugExecutionForCustomCode

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

“Model Configuration Parameters: Simulation Target” on page 13-2 | C Caller | C Function | MATLAB Function | MATLAB System

More About

- “Integrate C Code Using C Caller Blocks”
- “Integrate External C/C++ Code into Simulink Using C Function Blocks”

Enable global variables as function interfaces

Description

Specify the behavior of global variables in custom code called by the C Caller block. If this option is selected, the variables declared as global in the custom code can be used as global arguments on the block interface.

Category: Simulation Target

Settings

Default: Off

On

Enables parsing of custom code global variables on the function interface. C Caller block treats the global variables in your custom code as global arguments on the block interface. These arguments appear in bold on the **Port Specification** table.

Off

Disables parsing of global variables on the block interface.

Command-Line Information

Parameter: CustomCodeGlobalsAsFunctionIO

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

“Model Configuration Parameters: Simulation Target” on page 13-2 | C Caller | FunctionPortSpecification | getGlobalArg

More About

- “Integrate C Code Using C Caller Blocks”

Exception by function

Description

Specify how input array data is handled by each external C/C++ function and class method.

Category: Simulation Target

Settings

Specify how input array data is handled by each external C/C++ function and class method in your custom code. The array layout specified for an individual function or class method takes precedence over the option specified in **Default function array layout** on page 13-26. Use these options to add or remove the array layout setting for an individual function or class method:



Add

Add custom C/C++ function or class method and specify its array layout setting. To specify a class method, use the syntax *ClassName::MethodName*.



Remove

Remove custom C/C++ function or class method from the exception list and apply default array layout to the function or class method.

Tip If you do not see a list of your custom code functions in the **Exception by function** dialog, close the dialog, click **Validate custom code**, and click **Exception by function** again.

Command-Line Information

Parameter: CustomCodeFunctionArrayLayout

Type: structure array

Value: structure with 'FunctionName' and 'ArrayLayout' fields. 'ArrayLayout' can be 'Column-major', 'Row-major' or 'Any'.

Default: ' '

Example

Consider the model `foo_model`. If you have external C/C++ functions and class methods that you interface with the model, execute these MATLAB commands to specify array layouts for the functions and class methods.

```
arrayLayout(1).FunctionName = 'MyCFunction1';
arrayLayout(1).ArrayLayout = 'Column-major';
arrayLayout(2).FunctionName = 'MyCFunction2';
arrayLayout(2).ArrayLayout = 'Row-major';
arrayLayout(3).FunctionName = 'myClass::getboolRes';
arrayLayout(3).ArrayLayout = 'Row-major';
set_param('foo_model', 'CustomCodeFunctionArrayLayout', arrayLayout)
```

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Related Examples

- “Integrate External Code by Using Model Configuration Parameters” (Simulink Coder)
- “Model Configuration Parameters: Simulation Target” on page 13-2
- C Caller

Solver Parameters

Solver Pane

The **Solver** category includes parameters for configuring a solver for a model. A solver computes a dynamic system's states at successive time steps over a specified time span. You also use these parameters to specify the simulation start and stop times.

Once the model compiles, the Solver Information tooltip displays

- Compiled solver name
- Step size (**Max step size** or **Fixed step size**)

Once the model compiles, the status bar displays the solver used for compiling and a carat (^) when:

- Simulink selects a different solver during compilation.
- You set the step size to **auto**. The Solver Information tooltip displays the step size that Simulink calculated.

When configuring the solver, note that:

- Simulation time is not the same as clock time. For example, running a simulation for 10 seconds usually does not take 10 seconds. Total simulation time depends on factors such as model complexity, solver step sizes, and computer speed.
- **Fixed-step** solver type is required for code generation, unless you use an S-function or RSim target.
- **Variable-step** solver type can significantly shorten the time required to simulate models in which states change rapidly or which contain discontinuities.

Parameter	Description
"Start time" on page 14-6	Specify the start time for the simulation or generated code as a double-precision value, scaled to seconds.
"Stop time" on page 14-7	Specify the stop time for the simulation or generated code as a double-precision value, scaled to seconds.
"Type" on page 14-8	Select the type of solver you want to use to simulate your model.
"Solver" on page 14-10	Select the solver you want to use to compute the states of the model during simulation or code generation.
"Max step size" on page 14-16	Specify the largest time step that the solver can take.
"Integration method" on page 14-73	Specify the integration order of the odeN solver
"Initial step size" on page 14-18	Specify the size of the first time step that the solver takes.
"Min step size" on page 14-20	Specify the smallest time step that the solver can take.

Parameter	Description
“Relative tolerance” on page 14-22	Specify the largest acceptable solver error, relative to the size of each state during each time step. If the relative error exceeds this tolerance, the solver reduces the time step size.
“Absolute tolerance” on page 14-24	Specify the largest acceptable solver error, as the value of the measured state approaches zero. If the absolute error exceeds this tolerance, the solver reduces the time step size.
“Shape preservation” on page 14-26	At each time step use derivative information to improve integration accuracy.
“Maximum order” on page 14-28	Select the order of the numerical differentiation formulas (NDFs) used in the <code>ode15s</code> solver.
“Solver reset method” on page 14-30	Select how the solver behaves during a reset, such as when it detects a zero crossing.
“Number of consecutive min steps” on page 14-32	Specify the maximum number of consecutive minimum step size violations allowed during simulation.
“Solver Jacobian Method” on page 14-34	Specify the method to compute the Jacobian matrix for an implicit solver.
“Daessc mode” on page 14-36	Fine-tune the <code>daessc</code> solver performance.
“Treat each discrete rate as a separate task” on page 14-42	Specify whether Simulink executes blocks with periodic sample times individually or in groups.
“Automatically handle rate transition for data transfer” on page 14-44	Specify whether Simulink software automatically inserts hidden Rate Transition blocks between blocks that have different sample rates to ensure: the integrity of data transfers between tasks; and optional determinism of data transfers for periodic tasks.
“Deterministic data transfer” on page 14-46	Control whether the Rate Transition block parameter Ensure deterministic data transfer (maximum delay) is set for auto-inserted Rate Transition blocks.
“Higher priority value indicates higher task priority” on page 14-48	Specify whether the real-time system targeted by the model assigns higher or lower priority values to higher priority tasks when implementing asynchronous data transfers.
“Zero-crossing control” on page 14-49	Enables zero-crossing detection during model simulation. For most models, this speeds up simulation by enabling the solver to take larger time steps.
“Time tolerance” on page 14-51	Specify a tolerance factor that controls how closely zero-crossing events must occur to be considered consecutive.

Parameter	Description
“Number of consecutive zero crossings” on page 14-53	Specify the number of consecutive zero crossings that can occur before Simulink software displays a warning or an error.
“Algorithm” on page 14-55	Specifies the algorithm to detect zero crossings when a variable-step solver is used.
“Signal threshold” on page 14-57	Specifies the deadband region used during the detection of zero crossings. Signals falling within this region are defined as having crossed through zero.
“Periodic sample time constraint” on page 14-59	Select constraints on the sample times defined by this model. If the model does not satisfy the specified constraints during simulation, Simulink software displays an error message.
“Fixed-step size (fundamental sample time)” on page 14-61	Specify the step size used by the selected fixed-step solver.
“Sample time properties” on page 14-63	Specify and assign priorities to the sample times that this model implements.
“Extrapolation order” on page 14-65	Select the extrapolation order used by the <code>ode14x</code> solver to compute a model's states at the next time step from the states at the current time step.
“Number of Newton's iterations” on page 14-67	Specify the number of Newton's method iterations used by the <code>ode14x</code> solver to compute a model's states at the next time step from the states at the current time step.
“Allow tasks to execute concurrently on target” on page 14-69	Enable concurrent tasking behavior for model.
“Auto scale absolute tolerance” on page 14-71	Enable automatic absolute tolerance adaptation
“Allow multiple tasks to access inputs and outputs” on page 14-41	Enable Branched Input Multiple Outputs in rate-based models
“Enable zero-crossing detection for fixed-step solver” on page 14-38	Enable zero-crossing detection with fixed step
“Maximum number of bracketing iterations” on page 14-39	Specify maximum number of bracketing iterations performed when locating a zero crossing
“Maximum number of zero-crossings per step” on page 14-40	Specify the maximum number of zero-crossings to locate in one fixed step

These configuration parameters are in the **Advanced parameters** section.

Parameter	Description
“Enable decoupled continuous integration” on page 2-112	Removes the coupling between continuous and discrete rates.

Parameter	Description
"Enable minimal zero-crossing impact integration" on page 2-114	Minimizes the impact of zero-crossings on the integration of continuous states.

See Also

Related Examples

- "Solver Selection Criteria"

Start time

Description

Specify the start time for the simulation or generated code as a double-precision value, scaled to seconds.

Category: Solver

Settings

Default: 0.0

- A start time must be less than or equal to the stop time. For example, use a nonzero start time to delay the start of a simulation while running an initialization script.
- The values of block parameters with initial conditions must match the initial condition settings at the specified start time.
- Simulation time is not the same as clock time. For example, running a simulation for 10 seconds usually does not take 10 seconds. Total simulation time depends on factors such as model complexity, solver step sizes, and computer speed.

Programmatic Use

Parameter: StartTime

Type: character vector

Default: '0.0'

See Also

Related Examples

- “Solver Pane” on page 14-2

Stop time

Description

Specify the stop time for the simulation or generated code as a double-precision value, scaled to seconds.

Category: Solver

Settings

Default: 10

- Stop time must be greater than or equal to the start time.
- Specify `inf` to run a simulation or generated program until you explicitly pause or stop it.
- If the stop time is the same as the start time, the simulation or generated program runs for one step.
- Simulation time is not the same as clock time. For example, running a simulation for 10 seconds usually does not take 10 seconds. Total simulation time depends on factors such as model complexity, solver step sizes, and computer speed.
- If your model includes blocks that depend on absolute time and you are creating a design that runs indefinitely, see “Blocks That Depend on Absolute Time”.

Programmatic Use

Parameter: `StopTime`

Type: character vector

Value: any valid value

Default: `'10.0'`

See Also

Related Examples

- “Blocks That Depend on Absolute Time”
- “Use Blocks to Stop or Pause a Simulation”
- “Solver Pane” on page 14-2

Type

Description

Select the type of solver you want to use to simulate your model.

Category: Solver

Settings

Default: Variable-step

Variable-step

Step size varies from step to step, depending on model dynamics. A variable-step solver:

- Reduces step size when model states change rapidly, to maintain accuracy.
- Increases step size when model states change slowly, to avoid unnecessary steps.

Variable-step is recommended for models in which states change rapidly or that contain discontinuities. In these cases, a variable-step solver requires fewer time steps than a fixed-step solver to achieve a comparable level of accuracy. This can significantly shorten simulation time.

Fixed-step

Step size remains constant throughout the simulation. You require a fixed-step solver for code generation, unless you use an S-function or RSim target. Typically, lower order solvers are computationally less expensive than higher order solvers. However, they also provide less accuracy.

Note The solver computes the next time as the sum of the current time and the step size.

Dependencies

Selecting Variable-step enables the following parameters:

- **Solver**
- **Max step size**
- **Min step size**
- **Initial step size**
- **Relative tolerance**
- **Absolute tolerance**
- **Shape preservation**
- **Initial step size**
- **Number of consecutive min steps**
- **Zero-crossing control**
- **Time tolerance**

- **Algorithm**

Selecting Fixed-step enables the following parameters:

- **Solver**
- **Periodic sample time constraint**
- **Fixed-step size (fundamental sample time)**
- **Treat each discrete rate as a separate task**
- **Higher priority value indicates higher task priority**
- **Automatically handle rate transitions for data transfers**

Programmatic Use

Parameter: SolverType

Value: 'Variable-step' | 'Fixed-step'

Default: 'Variable-step'

See Also

Related Examples

- “Choose a Solver”
- “Purely Discrete Systems”
- “Solver Pane” on page 14-2

Solver

Description

Select the solver you want to use to compute the states of the model during simulation or code generation.

Category: Solver

Settings

Select from these types:

- “Fixed-step Solvers” on page 14-10
- “Variable-step Solvers” on page 14-11

The default setting for new models is `VariableStepAuto`.

Fixed-step Solvers

Default: `FixedStepAuto`

In general, all fixed-step solvers except for `ode14x` calculate the next step as:

$$X(n+1) = X(n) + h \, dX(n)$$

where X is the state, h is the step size, and dX is the state derivative. $dX(n)$ is calculated by a particular algorithm using one or more derivative evaluations depending on the order of the method.

`auto`

Computes the state of the model using a fixed-step solver that `auto` solver selects. At the time the model compiles, `auto` changes to a fixed-step solver that `auto` solver selects based on the model dynamics. Click on the solver hyperlink in the lower right corner of the model to accept or change this selection.

`ode3` (Bogacki-Shampine)

Computes the state of the model at the next time step as an explicit function of the current value of the state and the state derivatives, using the Bogacki-Shampine Formula integration technique to compute the state derivatives.

Discrete (no continuous states)

Computes the time of the next time step by adding a fixed step size to the current time.

Use this solver for models with no states or discrete states only, using a fixed step size. Relies on the model's blocks to update discrete states.

The accuracy and length of time of the resulting simulation depends on the size of the steps taken by the simulation: the smaller the step size, the more accurate the results but the longer the simulation takes.

Note The fixed-step discrete solver cannot be used to simulate models that have continuous states.

ode8 (Dormand-Prince RK8(7))

Uses the eighth-order Dormand-Prince formula to compute the model state at the next time step as an explicit function of the current value of the state and the state derivatives approximated at intermediate points.

ode5 (Dormand-Prince)

Uses the fifth-order Dormand-Prince formula to compute the model state at the next time step as an explicit function of the current value of the state and the state derivatives approximated at intermediate points.

ode4 (Runge-Kutta)

Uses the fourth-order Runge-Kutta (RK4) formula to compute the model state at the next time step as an explicit function of the current value of the state and the state derivatives.

ode2 (Heun)

Uses the Heun integration method to compute the model state at the next time step as an explicit function of the current value of the state and the state derivatives.

ode1 (Euler)

Uses the Euler integration method to compute the model state at the next time step as an explicit function of the current value of the state and the state derivatives. This solver requires fewer computations than a higher order solver. However, it provides comparatively less accuracy.

ode14x (extrapolation)

Uses a combination of Newton's method and extrapolation from the current value to compute the model's state at the next time step, as an *implicit* function of the state and the state derivative at the next time step. In the following example, X is the state, dX is the state derivative, and h is the step size:

$$X(n+1) - X(n) - h dX(n+1) = 0$$

This solver requires more computation per step than an explicit solver, but is more accurate for a given step size.

ode1be (Backward Euler)

The ode1be solver is a Backward Euler type solver that uses a fixed number of Newton iterations, and incurs only a fixed cost. You can use the ode1be solver as a computationally inexpensive fixed-step alternative to the ode14x solver.

Variable-step Solvers**Default: VariableStepAuto****auto**

Computes the state of the model using a variable-step solver that auto solver selects. At the time the model compiles, auto changes to a variable-step solver that auto solver selects based on the model dynamics. Click on the solver hyperlink in the lower right corner of the model to accept or change this selection.

ode45 (Dormand-Prince)

Computes the model's state at the next time step using an explicit Runge-Kutta (4,5) formula (the Dormand-Prince pair) for numerical integration.

ode45 is a one-step solver, and therefore only needs the solution at the preceding time point.

Use `ode45` as a first try for most problems.

Discrete (no continuous states)

Computes the time of the next step by adding a step size that varies depending on the rate of change of the model's states.

Use this solver for models with no states or discrete states only, using a variable step size.

ode23 (Bogacki-Shampine)

Computes the model's state at the next time step using an explicit Runge-Kutta (2,3) formula (the Bogacki-Shampine pair) for numerical integration.

`ode23` is a one-step solver, and therefore only needs the solution at the preceding time point.

`ode23` is more efficient than `ode45` at crude tolerances and in the presence of mild stiffness.

ode113 (Adams)

Computes the model's state at the next time step using a variable-order Adams-Bashforth-Moulton PECE numerical integration technique.

`ode113` is a multistep solver, and thus generally needs the solutions at several preceding time points to compute the current solution.

`ode113` can be more efficient than `ode45` at stringent tolerances.

ode15s (stiff/NDF)

Computes the model's state at the next time step using variable-order numerical differentiation formulas (NDFs). These are related to, but more efficient than the backward differentiation formulas (BDFs), also known as Gear's method.

`ode15s` is a multistep solver, and thus generally needs the solutions at several preceding time points to compute the current solution.

`ode15s` is efficient for stiff problems. Try this solver if `ode45` fails or is inefficient.

ode23s (stiff/Mod. Rosenbrock)

Computes the model's state at the next time step using a modified Rosenbrock formula of order 2.

`ode23s` is a one-step solver, and therefore only needs the solution at the preceding time point.

`ode23s` is more efficient than `ode15s` at crude tolerances, and can solve stiff problems for which `ode15s` is ineffective.

ode23t (Mod. stiff/Trapezoidal)

Computes the model's state at the next time step using an implementation of the trapezoidal rule with a "free" interpolant.

`ode23t` is a one-step solver, and therefore only needs the solution at the preceding time point.

Use `ode23t` if the problem is only moderately stiff and you need a solution with no numerical damping.

ode23tb (stiff/TR-BDF2)

Computes the model's state at the next time step using a multistep implementation of TR-BDF2, an implicit Runge-Kutta formula with a trapezoidal rule first stage, and a second stage consisting of a backward differentiation formula of order two. By construction, the same iteration matrix is used in evaluating both stages.

ode23tb is more efficient than ode15s at crude tolerances, and can solve stiff problems for which ode15s is ineffective.

odeN (Nonadaptive)

Uses an N^{th} order fixed step integration formula to compute the model state as an explicit function of the current value of the state and the state derivatives approximated at intermediate points.

While the solver itself is a fixed step solver, Simulink will reduce the step size at zero crossings for accuracy.

daessc (Solver for Simscape)

Computes the model's state at the next time step by solving systems of differential-algebraic equations resulting from Simscape models. `daessc` provides robust algorithms specifically designed to simulate differential-algebraic equations arising from modeling physical systems.

`daessc` is only available with Simscape products.

Tips

- Identifying the optimal solver for a model requires experimentation. For an in-depth discussion, see “Solver Selection Criteria”.
- With fast restart, you do not need to recompile the model if you change the solver. You can pick appropriate solvers during runtime without having to go through an expensive recompilation process.
- The optimal solver balances acceptable accuracy with the shortest simulation time.
- Simulink software uses a discrete solver for a model with no states or discrete states only, even if you specify a continuous solver.
- A smaller step size increases accuracy, but also increases simulation time.
- The degree of computational complexity increases for `oden`, as n increases.
- As computational complexity increases, the accuracy of the results also increases.

Dependencies

Selecting the `ode1` (Euler), `ode2` (Huen), `ode3` (Bogacki-Shampine), `ode4` (Runge-Kutta), `ode5` (Dormand-Prince), `ode8` (Dormand Prince RK8(7)) or `Discrete` (no continuous states) fixed-step solvers enables the following parameters:

- **Fixed-step size (fundamental sample time)**
- **Periodic sample time constraint**
- **Treat each discrete rate as a separate task**
- **Automatically handle rate transition for data transfers**
- **Higher priority value indicates higher task priority**

Selecting `odeN` (Nonadaptive) variable-step solver enables the following parameters:

- **Max step size**
- **Integration method**

Selecting `ode14x` (extrapolation) enables the following parameters:

- **Fixed-step size (fundamental sample time)**
- **Extrapolation order**
- **Number of Newton's iterations**
- **Periodic sample time constraint**
- **Treat each discrete rate as a separate task**
- **Automatically handle rate transition for data transfers**
- **Higher priority value indicates higher task priority**

Selecting `ode1be` (Backward Euler) enables the following parameters:

- **Fixed-step size (fundamental sample time)**
- **Number of Newton's iterations**
- **Periodic sample time constraint**
- **Treat each discrete rate as a separate task**
- **Automatically handle rate transition for data transfers**
- **Higher priority value indicates higher task priority**

Selecting the Discrete (no continuous states) variable-step solver enables the following parameters:

- **Max step size**
- **Automatically handle rate transition for data transfers**
- **Higher priority value indicates higher task priority**
- **Zero-crossing control**
- **Time tolerance**
- **Number of consecutive zero crossings**
- **Algorithm**

Selecting `ode45` (Dormand-Prince), `ode23` (Bogacki-Shampine), `ode113` (Adams), or `ode23s` (stiff/Mod. Rosenbrock) enables the following parameters:

- **Max step size**
- **Min step size**
- **Initial step size**
- **Relative tolerance**
- **Absolute tolerance**
- **Shape preservation**
- **Number of consecutive min steps**
- **Automatically handle rate transition for data transfers**
- **Higher priority value indicates higher task priority**
- **Zero-crossing control**
- **Time tolerance**
- **Number of consecutive zero crossings**

- **Algorithm**

Selecting ode15s (stiff/NDF), ode23t (Mod. stiff/Trapezoidal), or ode23tb (stiff/TR-BDF2) enables the following parameters:

- **Max step size**
- **Min step size**
- **Initial step size**
- **Solver reset method**
- **Number of consecutive min steps**
- **Relative tolerance**
- **Absolute tolerance**
- **Shape preservation**
- **Maximum order**
- **Automatically handle rate transition for data transfers**
- **Higher priority value indicates higher task priority**
- **Zero-crossing control**
- **Time tolerance**
- **Number of consecutive zero crossings**
- **Algorithm**

Command-Line Information

Parameter: SolverName or Solver

Value: 'VariableStepAuto' | 'VariableStepDiscrete' | 'ode45' | 'ode23' | 'ode113' | 'ode15s' | 'ode23s' | 'ode23t' | 'ode23tb' | 'daessc' | 'FixedStepAuto' | 'FixedStepDiscrete' | 'ode8' | 'ode5' | 'ode4' | 'ode3' | 'ode2' | 'ode1' | 'ode14x'

Default: 'VariableStepAuto'

See Also

Related Examples

- “Compare Solvers”
- “Solver Selection Criteria”
- “Purely Discrete Systems”
- “Solver Pane” on page 14-2

Max step size

Description

Specify the largest time step that the solver can take.

Category: Solver

Settings

Default: auto

- For the discrete solver, the default value (**auto**) is the model's shortest sample time.
- For continuous solvers, the default value (**auto**) is determined from the start and stop times. If the stop time equals the start time or is `inf`, Simulink chooses 0.2 seconds as the maximum step size. Otherwise, it sets the maximum step size to

$$h_{\max} = \frac{t_{\text{stop}} - t_{\text{start}}}{50}$$

- For Sine and Signal Generator source blocks, Simulink calculates the max step size using this heuristic:

$$h_{\max} = \min\left(\frac{t_{\text{stop}} - t_{\text{start}}}{50}, \left(\frac{1}{3}\right)\left(\frac{1}{\text{Freq}_{\max}}\right)\right)$$

where Freq_{\max} is the maximum frequency (Hz) of these blocks in the model.

Tips

- Generally, the default maximum step size is sufficient. If you are concerned about the solver missing significant behavior, change the parameter to prevent the solver from taking too large a step.
- Max step size determines the step size of the variable-step solver.
- If the time span of the simulation is very long, the default step size might be too large for the solver to find the solution.
- If your model contains periodic or nearly periodic behavior and you know the period, set the maximum step size to some fraction (such as 1/4) of that period.
- In general, for more output points, change the refine factor, not the maximum step size.

Dependencies

This parameter is enabled only if the solver **Type** is set to **Variable-step**.

Programmatic Use

Parameter: MaxStep

Type: character vector

Value: any valid value

Default: 'auto'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- "Purely Discrete Systems"
- "Solver Pane" on page 14-2

Initial step size

Description

Specify the size of the first time step that the solver takes.

Category: Solver

Settings

Default: auto

By default, the solver selects an initial step size by examining the derivatives of the states at the start time.

Tips

- Be careful when increasing the initial step size. If the first step size is too large, the solver might step over important behavior.
- The initial step size parameter is a *suggested* first step size. The solver tries this step size but reduces it if error criteria are not satisfied.

Dependencies

This parameter is enabled only if the solver **Type** is set to Variable-step.

Programmatic Use

Parameter: InitialStep

Type: character vector

Value: any valid value

Default: 'auto'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Purely Discrete Systems”

- “Improve Simulation Performance Using Performance Advisor”
- “Solver Pane” on page 14-2

Min step size

Description

Specify the smallest time step that the solver can take.

Category: Solver

Settings

Default: auto

- The default value (auto) sets an unlimited number of warnings and a minimum step size on the order of machine precision.
- You can specify either a real number greater than zero, or a two-element vector for which the first element is the minimum step size and the second element is the maximum number of minimum step size warnings before an error was issued.

Tips

- If the solver takes a smaller step to meet error tolerances, it issues a warning indicating the current effective relative tolerance.
- Setting the second element to zero results in an error the first time the solver must take a step smaller than the specified minimum. This is equivalent to changing the **Min step size violation** diagnostic to error on the **Diagnostics** pane (see “Min step size violation” on page 9-11).
- Setting the second element to -1 results in an unlimited number of warnings. This is also the default if the input is a scalar.
- Min step size determines the step size of the variable step ODE solver. The size is limited by the smallest discrete sample time in the model.

Dependencies

This parameter is enabled only if the solver **Type** is set to Variable-step.

Programmatic Use

Parameter: MinStep

Type: character vector

Value: any valid value

Default: 'auto'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact

Application	Setting
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Purely Discrete Systems”
- “Min step size violation” on page 9-11
- “Solver Pane” on page 14-2

Relative tolerance

Description

Specify the largest acceptable solver error, relative to the size of each state during each time step. If the relative error exceeds this tolerance, the solver reduces the time step size.

Category: Solver

Settings

Default: 1e-3

- Setting the relative tolerance to `auto` is actually the default value of 1e-3.
- The relative tolerance is a percentage of the state's value.
- The default value (1e-3) means that the computed state is accurate to within 0.1%.

Tips

- The acceptable error at each time step is a function of both the **Relative tolerance** and the **Absolute tolerance**. For more information about how these settings work together, see “Error Tolerances for Variable-Step Solvers”.
- During each time step, the solver computes the state values at the end of the step and also determines the local error – the estimated error of these state values. If the error is greater than the acceptable error for any state, the solver reduces the step size and tries again.
- The default relative tolerance value is sufficient for most applications. Decreasing the relative tolerance value can slow down the simulation.
- To check the accuracy of a simulation after you run it, you can reduce the relative tolerance to 1e-4 and run it again. If the results of the two simulations are not significantly different, you can feel confident that the solution has converged.

Dependencies

This parameter is enabled only if you set:

- Solver **Type** to `Variable-step`.
- **Solver** to a continuous variable-step solver.

This parameter works along with **Absolute tolerance** to determine the acceptable error at each time step. For more information about how these settings work together, see “Error Tolerances for Variable-Step Solvers”.

Programmatic Use

Parameter: `RelTol`

Type: character vector

Value: any valid value

Default: '1e-3'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Error Tolerances for Variable-Step Solvers”
- “Improve Simulation Performance Using Performance Advisor”
- “Solver Pane” on page 14-2

Absolute tolerance

Description

Specify the largest acceptable solver error, as the value of the measured state approaches zero. If the absolute error exceeds this tolerance, the solver reduces the time step size.

Category: Solver

Settings

Default: auto

- The default value (auto) initially sets the absolute tolerance for each state based on the relative tolerance alone. If the relative tolerance is larger than $1e-3$, then the initial absolute tolerance is set to $1e-6$. However, for relative tolerances smaller than $1e-3$, the absolute tolerance for the state is initialized to $reltol * 1e-3$. As the simulation progresses, the absolute tolerance for each state is reset to the maximum value that the state has reached until that point, times the relative tolerance for that state.

For example, if a state goes from 0 to 1 and the **Relative tolerance** is $1e-4$, then the **Absolute tolerance** is initialized at $1e-7$ and by the end of the simulation, the **Absolute tolerance** reaches $1e-4$.

If, on the other hand, the **Relative tolerance** is set to $1e-3$, the **Absolute tolerance** is set to $1e-6$ and by the end of the simulation, reaches $1e-3$.

- If the computed setting is not suitable, you can determine an appropriate setting yourself.
- If you do set your own value for **Absolute tolerance**, you can also select whether it adapts based on the value of the states by toggling the `AutoScaleAbsTol` parameter. For more information, see “Auto scale absolute tolerance” on page 14-71.

Tips

- The acceptable error at each time step is a function of both the **Relative tolerance** and the **Absolute tolerance**. For more information about how these settings work together, see “Error Tolerances for Variable-Step Solvers”.
- The Integrator, Second-Order Integrator, Variable Transport Delay, Transfer Fcn, State-Space, and Zero-Pole blocks allow you to specify absolute tolerance values for solving the model states that they compute or that determine their output. The absolute tolerance values that you specify in these blocks override the global setting in the Configuration Parameters dialog box.
- You might want to override the **Absolute tolerance** setting using blocks if the global setting does not provide sufficient error control for all your model states, for example, if they vary widely in magnitude.
- If you set the **Absolute tolerance** too low, the solver might take too many steps around near-zero state values, and thus slow the simulation.
- To check the accuracy of a simulation after you run it, you can reduce the absolute tolerance and run it again. If the results of the two simulations are not significantly different, you can feel confident that the solution has converged.

- If your simulation results do not seem accurate, and your model has states whose values approach zero, the **Absolute tolerance** may be too large. Reduce the **Absolute tolerance** to force the simulation to take more steps around areas of near-zero state values.

Dependencies

This parameter is enabled only if you set:

- Solver **Type** to Variable-step.
- **Solver** to a continuous variable-step solver.

This parameter works along with **Relative tolerance** to determine the acceptable error at each time step. For more information about how these settings work together, see “Error Tolerances for Variable-Step Solvers”.

Programmatic Use

Parameter: AbsTol

Type: character vector | numeric value

Value: 'auto' | positive real scalar

Default: 'auto'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Error Tolerances for Variable-Step Solvers”
- “Improve Simulation Performance Using Performance Advisor”
- “Solver Pane” on page 14-2

Shape preservation

Description

At each time step use derivative information to improve integration accuracy.

Category: Solver

Settings

Default: Disable all

Disable all

Do not perform Shape preservation on any signals.

Enable all

Perform Shape preservation on all signals.

Tips

- The default setting (Disable all) usually provides good accuracy for most models.
- Setting to Enable all will increase accuracy in those models having signals whose derivative exhibits a high rate of change, but simulation time may be increased.

Dependencies

This parameter is enabled only if you use a continuous-step solver.

Programmatic Use

Parameter: ShapePreserveControl

Value: 'EnableAll' | 'DisableAll'

Default: 'DisableAll'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Zero-Crossing Detection”

- “Solver Pane” on page 14-2

Maximum order

Description

Select the order of the numerical differentiation formulas (NDFs) used in the `ode15s` solver.

Category: Solver

Settings

Default: 5

5

Specifies that the solver uses fifth order NDFs.

1

Specifies that the solver uses first order NDFs.

2

Specifies that the solver uses second order NDFs.

3

Specifies that the solver uses third order NDFs.

4

Specifies that the solver uses fourth order NDFs.

Tips

- Although the higher order formulas are more accurate, they are less stable.
- If your model is stiff and requires more stability, reduce the maximum order to 2 (the highest order for which the NDF formula is A-stable).
- As an alternative, you can try using the `ode23s` solver, which is a lower order (and A-stable) solver.

Dependencies

This parameter is enabled only if **Solver** is set to `ode15s`.

Programmatic Use

Parameter: MaxOrder

Type: integer

Value: 1 | 2 | 3 | 4 | 5

Default: 5

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Error Tolerances for Variable-Step Solvers”
- “Improve Simulation Performance Using Performance Advisor”
- “Solver Pane” on page 14-2

Solver reset method

Description

Select how the solver behaves during a reset, such as when it detects a zero crossing.

Category: Solver

Settings

Default: Fast

Fast

Specifies that the solver will not recompute the Jacobian matrix at a solver reset.

Robust

Specifies that the solver will recompute the Jacobian matrix needed by the integration step at every solver reset.

Tips

- Selecting **Fast** speeds up the simulation. However, it can result in incorrect solutions in some cases.
- If you suspect that the simulation is giving incorrect results, try the **Robust** setting. If there is no difference in simulation results between the fast and robust settings, revert to the fast setting.

Dependencies

This parameter is enabled only if you select one of the following solvers:

- ode15s (Stiff/NDF)
- ode23t (Mod. Stiff/Trapezoidal)
- ode23tb (Stiff/TR-BDF2)

Programmatic Use

Parameter: SolverResetMethod

Value: 'Fast' | 'Robust'

Default: 'Fast'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Choose a Solver”
- “Solver Pane” on page 14-2

Number of consecutive min steps

Description

Specify the maximum number of consecutive minimum step size violations allowed during simulation.

Category: Solver

Settings

Default: 1

- A minimum step size violation occurs when a variable-step continuous solver takes a smaller step than that specified by the **Min step size** property (see “Min step size” on page 14-20).
- Simulink software counts the number of consecutive violations that it detects. If the count exceeds the value of **Number of consecutive min steps**, Simulink software displays either a warning or error message as specified by the **Min step size violation** diagnostic (see “Min step size violation” on page 9-11).

Dependencies

This parameter is enabled only if you set:

- Solver **Type** to Variable-step.
- **Solver** to a continuous variable step solver.

Programmatic Use

Parameter: MaxConsecutiveMinStep

Type: character vector

Value: any valid value

Default: '1'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Choose a Solver”

- “Min step size violation” on page 9-11
- “Min step size” on page 14-20
- “Solver Pane” on page 14-2

Solver Jacobian Method

Description

Category: Solver

Settings

Default: auto

auto

Sparse perturbation

Full perturbation

Sparse analytical

Full analytical

Tips

- The default setting (auto) usually provides good accuracy for most models.

Dependencies

This parameter is enabled only if an implicit solver is used.

Programmatic Use

Parameter: SolverJacobianMethodControl

Value: 'auto' | 'SparsePerturbation' | 'FullPerturbation' | 'SparseAnalytical' | 'FullAnalytical'

Default: 'auto'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Choose a Solver”

- “Solver Pane” on page 14-2

Daessc mode

Description

Category: Solver

Settings

Default: auto

auto

Automatically selects the optimal daessc solver mode.

Fast

The most efficient mode in terms of computation cost, but less robust.

Balanced

Provides a balance between computational costs and robustness.

Robust

More robust, but also more costly than Fast or Balanced.

Quick debug

Updates the solver Jacobian at every integration step, and is therefore even more costly than Robust. Recommended only for interactive model development, to quickly find issues with equations.

Full debug

Updates the solver Jacobian at every integration step and every Newton iteration. This mode is the most expensive in terms of computational cost. Recommended only for interactive model development, to thoroughly check equations and find possible issues.

Tips

- The default setting (auto) usually provides a good balance between speed and robustness for most models.
- Robust mode tends to have a higher computational cost.
- Debug modes are the most expensive and they are recommended only for interactive model development.

Dependencies

This parameter is enabled only if the daessc (DAE solver for Simscape) solver is used.

Programmatic Use

Parameter: DaesscMode

Value: 'auto' | 'Fast' | 'Balanced' | 'Robust' | 'QuickDebug' | 'FullDebug'

Default: 'auto'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Choose a Solver”
- “Solver Pane” on page 14-2

Enable zero-crossing detection for fixed-step solver

Description

Specify whether to enable zero-crossing detection when using a fixed-step solver.

Category: Solver

Settings

Default: Off

On

Enables zero-crossing detection for fixed-step simulation.

Off

Disables zero-crossing detection for fixed-step simulation.

Dependencies

To enable this parameter, set the solver **Type** to Fixed-step and set **Solver** to any value except discrete (no continuous states).

Enabling **Enable zero-crossing detection for fixed-step solver** enables these parameters:

- **Zero-crossing control**
- **Maximum number of bracketing iterations**
- **Maximum number of zero-crossings per step**

Programmatic Use

Parameter: EnableFixedStepZeroCrossing

Value: 'on' | 'off'

Default: 'off'

See Also

Related Examples

- “Zero-Crossing Detection”
- “Zero-Crossing Detection with Fixed-Step Simulation”
- “Zero-Crossing Algorithms”
- “Use Fixed-Step Zero-Crossing Detection for Faster Simulations”

Maximum number of bracketing iterations

Description

Specify the number of iterations for the root-finding algorithm to perform when trying to locate each zero crossing.

Category: Solver

Settings

Default: 10

More iterations produce a more accurate solution but are more computationally intensive.

Dependencies

To enable this parameter:

- Set solver **Type** to Fixed-step.
- Set **Solver** to any value except discrete (no continuous states).
- Select **Enable zero-crossing detection for fixed-step solver**.

Programmatic Use

Parameter: MaxZcBracketingIterations

Type: integer

Value: real positive whole number

Default: 10

See Also

Related Examples

- [“Zero-Crossing Detection”](#)
- [“Zero-Crossing Detection with Fixed-Step Simulation”](#)
- [“Zero-Crossing Algorithms”](#)
- [“Use Fixed-Step Zero-Crossing Detection for Faster Simulations”](#)

Maximum number of zero-crossings per step

Description

Specify the maximum number of zero crossings to try to locate in each step

Category: Solver

Settings

Default: 2

Detecting more zero crossings produces a more accurate solution but is more computationally intensive.

Dependencies

To enable this parameter:

- Set the solver **Type** to Fixed-step.
- Set **Solver** to any value except discrete (no continuous states).
- Select **Enable zero-crossing detection for fixed-step solver**.

Programmatic Use

Parameter: MaxZcPerStep

Type: integer

Value: real positive whole number

Default: 1

See Also

Related Examples

- “Zero-Crossing Detection”
- “Zero-Crossing Detection with Fixed-Step Simulation”
- “Zero-Crossing Algorithms”
- “Use Fixed-Step Zero-Crossing Detection for Faster Simulations”

Allow multiple tasks to access inputs and outputs

Description

Enable multi-tasked branched root inputs and merged root outputs.

Category: Solver

Settings

Default: Off

On

Specifies that multiple tasks can access the inputs and outputs of a model block or a root Inport block by assigning them a union sample time of those tasks. Union sample time implies that the input or output is accessed in all the component sample times.

Off

Specifies that greatest common denominator sample time with an implicit task is used when multiple tasks access inputs and outputs a model block, or a root Inport block.

Command-Line Information

Parameter: AllowMultiTaskInputOutput

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	On
Safety precaution	No recommendation

See Also

Rate Transition

Related Examples

- “Solver Pane” on page 14-2

Treat each discrete rate as a separate task

Description

Specify whether Simulink executes blocks with periodic sample times individually or in groups.

Category: Solver

Settings

Default: Off

On

Selects multitasking execution for models operating at different sample rates. Specifies that groups of blocks with the same execution priority are processed through each stage of simulation (for example, calculating output and updating discrete states) based on task priority. The multitasking mode helps to create valid models of real-world multitasking systems, where sections of your model represent concurrent tasks.

Off

Specifies that all blocks are processed through each stage of simulation together (for example, calculating output and updating discrete states). Use single-tasking execution if:

- Your model contains one sample time.
- Your model contains a continuous and a discrete sample time, and the fixed-step size is equal to the discrete sample time.

Tips

- A multirate model with multitasking mode enabled cannot reference another multirate model that has the single-tasking mode enabled.
- The **Single task data transfer** and **Multitask data transfer** parameters on the **Diagnostics > Sample Time** pane allow you to adjust error checking for sample rate transitions between blocks that operate at different sample rates.

Dependency

- This parameter is enabled by selecting the Fixed-step solver type.
- Use `local solver when referencing model` must be disabled for this parameter to be enabled.

Command-Line Information

Parameter: EnableMultiTasking

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact for simulation or during development Off for production code generation
Efficiency	No impact
Safety precaution	No recommendation

See Also

Rate Transition

Related Examples

- “Time-Based Scheduling” (Simulink Coder)
- “Model Execution and Rate Transitions” (Simulink Coder)
- “Handle Rate Transitions” (Simulink Coder)
- “Solver Pane” on page 14-2
- “Use local solver when referencing model” on page 12-25

Automatically handle rate transition for data transfer

Description

Specify whether Simulink software automatically inserts hidden Rate Transition blocks between blocks that have different sample rates to ensure: the integrity of data transfers between tasks; and optional determinism of data transfers for periodic tasks.

Category: Solver

Settings

Default: Off

On

Inserts hidden Rate Transition blocks between blocks when rate transitions are detected. Handles rate transitions for asynchronous and periodic tasks. Simulink software adds the hidden blocks configured to ensure data integrity for data transfers. Selecting this option also enables the parameter **Deterministic data transfer**, which allows you to control the level of data transfer determinism for periodic tasks.

Off

Does not insert hidden Rate Transition blocks when rate transitions are detected. If Simulink software detects invalid transitions, you must adjust the model such that the sample rates for the blocks in question match or manually add a Rate Transition block.

See “Rate Transition Block Options” (Simulink Coder) for further details.

Tips

- Selecting this parameter allows you to handle rate transition issues automatically. This saves you from having to manually insert Rate Transition blocks to avoid invalid rate transitions, including invalid asynchronous-to-periodic and asynchronous-to-asynchronous rate transitions, in multirate models.
- For asynchronous tasks, Simulink software configures the inserted blocks to ensure data integrity but not determinism during data transfers.

Programmatic Use

Parameter: AutoInsertRateTranBlk

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	No impact for simulation or during development Off for production code generation
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- “Rate Transition Block Options” (Simulink Coder)
- “Solver Pane” on page 14-2

Deterministic data transfer

Description

Control whether the Rate Transition block parameter **Ensure deterministic data transfer (maximum delay)** is set for auto-inserted Rate Transition blocks

Default: Whenever possible

Always

Specifies that the block parameter **Ensure deterministic data transfer (maximum delay)** is always set for auto-inserted Rate Transition blocks.

If Always is selected and if a model needs to auto-insert a Rate Transition block to handle a rate transition that is *not* between two periodic sample times related by an integer multiple, Simulink errors out.

Whenever possible

Specifies that the block parameter **Ensure deterministic data transfer (maximum delay)** is set for auto-inserted Rate Transition blocks whenever possible. If an auto-inserted Rate Transition block handles data transfer between two periodic sample times that are related by an integer multiple, **Ensure deterministic data transfer (maximum delay)** is set; otherwise, it is cleared.

Never (minimum delay)

Specifies that the block parameter **Ensure deterministic data transfer (maximum delay)** is never set for auto-inserted Rate Transition blocks.

Note Clearing the Rate Transition block parameter **Ensure deterministic data transfer (maximum delay)** can provide reduced latency for models that do not require determinism. See the description of **Ensure deterministic data transfer (maximum delay)** on the Rate Transition block reference page for more information.

Category: Solver

Dependencies

This parameter is enabled only if **Automatically handle rate transition for data transfer** is checked.

Programmatic Use

Parameter: InsertRTBMode

Value: 'Always' | 'Whenever possible' | 'Never (minimum delay)'

Default: 'Whenever possible'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- “Rate Transition Block Options” (Simulink Coder)
- “Solver Pane” on page 14-2

Higher priority value indicates higher task priority

Description

Specify whether the real-time system targeted by the model assigns higher or lower priority values to higher priority tasks when implementing asynchronous data transfers

Category: Solver

Settings

Default: Off

On

Real-time system assigns higher priority values to higher priority tasks, for example, 8 has a higher task priority than 4. Rate Transition blocks treat asynchronous transitions between rates with lower priority values and rates with higher priority values as low-to-high rate transitions.

Off

Real-time system assigns lower priority values to higher priority tasks, for example, 4 has a higher task priority than 8. Rate Transition blocks treat asynchronous transitions between rates with lower priority values and rates with higher priority values as high-to-low rate transitions.

Programmatic Use

Parameter: PositivePriorityOrder

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Rate Transitions and Asynchronous Blocks” (Simulink Coder)
- “Solver Pane” on page 14-2

Zero-crossing control

Description

Enables zero-crossing detection during model simulation. For most models, zero-crossing detection speeds up simulation by enabling the solver to take larger time steps.

Category: Solver

Settings

Default: Use local settings

Use local settings

Zero-crossing detection is enabled on a block-by-block basis. For a list of applicable blocks, see “Simulation Phases in Dynamic Systems”

To enable zero-crossing detection for a block, open the Block Parameters dialog box and select **Enable zero-crossing detection**.

Enable all

Enables zero-crossing detection for all blocks in the model.

Disable all

Disables zero-crossing detection for all blocks in the model.

Tips

- For most models, enabling zero-crossing detection speeds up simulation by allowing the solver to take larger time steps.
- If a model has extreme dynamic changes, disabling zero-crossing detection can speed up the simulation but can also decrease the accuracy of simulation results. See “Zero-Crossing Detection” for more information.
- Selecting **Enable all** or **Disable all** overrides the zero-crossing detection setting for individual blocks.

Dependencies

Variable-Step Solver

This parameter is always enabled when the solver **Type** is **Variable-step**.

When you use a variable-step solver, setting **Zero-crossing control** to either **Use local settings** or **Enable all** enables these parameters:

- **Time tolerance**
- **Number of consecutive zero crossings**
- **Algorithm**

Fixed-Step Solver

To enable this parameter when the solver **Type** is Fixed - step, select **Enable zero-crossing detection for fixed-step solver**.

Programmatic Use

Parameter: ZeroCrossControl

Value: 'UseLocalSettings' | 'EnableAll' | 'DisableAll'

Default: 'UseLocalSettings'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Zero-Crossing Detection”
- “Number of consecutive zero crossings” on page 14-53
- “Consecutive zero-crossings violation” on page 9-13
- “Time tolerance” on page 14-51
- “Solver Pane” on page 14-2

Time tolerance

Description

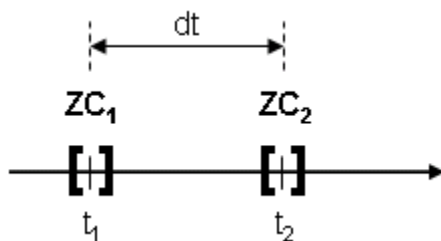
Specify a tolerance factor that controls how closely zero-crossing events must occur to be considered consecutive.

Category: Solver

Settings

Default: $10 \times 128 \times \text{eps}$

- Simulink software defines zero crossings as consecutive if the time between events is less than a particular interval. The following figure depicts a simulation timeline during which Simulink software detects zero crossings ZC_1 and ZC_2 , bracketed at successive time steps t_1 and t_2 .



Simulink software determines that the zero crossings are consecutive if

$$dt < \text{RelTolZC} * t_2$$

where dt is the time between zero crossings and RelTolZC is the **Time tolerance**.

- Simulink software counts the number of consecutive zero crossings that it detects. If the count exceeds the value of **Number of consecutive zero crossings** allowed, Simulink software displays either a warning or error as specified by the **Consecutive zero-crossings violation** diagnostic (see “Consecutive zero-crossings violation” on page 9-13).

Tips

- Simulink software resets the counter each time it detects nonconsecutive zero crossings (successive zero crossings that fail to meet the relative tolerance setting); therefore, decreasing the relative tolerance value may afford your model's behavior more time to recover.
- If your model experiences excessive zero crossings, you can also increase the **Number of consecutive zero crossings** to increase the threshold at which Simulink software triggers the **Consecutive zero-crossings violation** diagnostic.

Dependencies

To enable this parameter, set the solver **Type** to Variable-step and set **Zero-crossing control** to either Use local settings or Enable all.

Programmatic Use

Parameter: ConsecutiveZCsStepRelTol

Type: character vector

Value: any valid value

Default: '10*128*eps'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Zero-Crossing Detection”
- “Number of consecutive zero crossings” on page 14-53
- “Zero-crossing control” on page 14-49
- “Consecutive zero-crossings violation” on page 9-13
- “Solver Pane” on page 14-2

Number of consecutive zero crossings

Description

Specify the number of consecutive zero crossings that can occur before Simulink software displays a warning or an error.

Category: Solver

Settings

Default: 1000

- Simulink software counts the number of consecutive zero crossings that it detects. If the count exceeds the specified value, Simulink software displays either a warning or an error as specified by the **Consecutive zero-crossings violation** diagnostic (see “Consecutive zero-crossings violation” on page 9-13).
- Simulink software defines zero crossings as consecutive if the time between events is less than a particular interval (see “Time tolerance” on page 14-51).

Tips

- If your model experiences excessive zero crossings, you can increase this parameter to increase the threshold at which Simulink software triggers the **Consecutive zero-crossings violation** diagnostic. This may afford your model's behavior more time to recover.
- Simulink software resets the counter each time it detects nonconsecutive zero crossings; therefore, decreasing the relative tolerance value may also afford your model's behavior more time to recover.

Dependencies

To enable this parameter, set the solver **Type** to Variable-step and set **Zero-crossing control** to either Use local settings or Enable all.

Programmatic Use

Parameter: MaxConsecutiveZCs

Type: character vector

Value: any valid value

Default: '1000'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact

Application	Setting
Safety precaution	No impact

See Also

Related Examples

- “Zero-Crossing Detection”
- “Zero-crossing control” on page 14-49
- “Consecutive zero-crossings violation” on page 9-13
- “Time tolerance” on page 14-51
- “Solver Pane” on page 14-2

Algorithm

Description

Specifies the algorithm to detect zero crossings when a variable-step solver is used.

Category: Solver

Settings

Default: Nonadaptive

Adaptive

Use an improved zero-crossing algorithm which dynamically activates and deactivates zero-crossing bracketing. With this algorithm you can set a zero-crossing tolerance. See “Signal threshold” on page 14-57 to learn how to set the zero-crossing tolerance.

Nonadaptive

Use the nonadaptive zero-crossing algorithm present in the Simulink software prior to Version 7.0 (R2008a). This option detects zero-crossings accurately, but might cause longer simulation run times for systems with strong “chattering” or Zeno behavior.

Tips

- The adaptive zero-crossing algorithm is especially useful in systems having strong “chattering”, or Zeno behavior. In such systems, this algorithm yields shorter simulation run times compared to the nonadaptive algorithm. See “Zero-Crossing Detection” for more information.

Dependencies

To enable this parameter, set the solver **Type** to **Variable-step** and set **Zero-crossing control** to either **Use local settings** or **Enable all**.

Selecting the Adaptive algorithm enables the **Signal threshold** parameter.

Programmatic Use

Parameter: ZeroCrossAlgorithm

Value: 'Nonadaptive' | 'Adaptive'

Default: 'Nonadaptive'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact

Application	Setting
Safety precaution	No impact

See Also

Related Examples

- “Zero-Crossing Detection”
- “Zero-crossing control” on page 14-49
- “Consecutive zero-crossings violation” on page 9-13
- “Time tolerance” on page 14-51
- “Number of consecutive zero crossings” on page 14-53
- “Solver Pane” on page 14-2

Signal threshold

Description

Specifies the deadband region used during the detection of zero crossings. Signals falling within this region are defined as having crossed through zero.

The signal threshold is a real number, greater than or equal to zero.

Category: Solver

Settings

Default: auto

- By default, the zero-crossing signal threshold is determined automatically by the adaptive algorithm.
- You can also specify a value for the signal threshold. The value must be a real number equal to or greater than zero.

Tips

- Entering too small of a value for the **Signal Threshold** parameter will result in long simulation run times.
- Entering a large **Signal Threshold** value may improve the simulation speed (especially in systems having extensive chattering). However, making the value too large may reduce the simulation accuracy.

Dependencies

To enable this parameter:

- Set the solver **Type** to Variable-step.
- Set **Zero-crossing control** to either Use local settings or Enable all.
- Set **Algorithm** to Adaptive.

Programmatic Use

Parameter: ZCThreshold

Value: 'auto' | real number greater than or equal to zero

Default: 'auto'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact

Application	Setting
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Zero-Crossing Detection”
- “Zero-crossing control” on page 14-49
- “Consecutive zero-crossings violation” on page 9-13
- “Time tolerance” on page 14-51
- “Number of consecutive zero crossings” on page 14-53
- “Solver Pane” on page 14-2

Periodic sample time constraint

Description

Select constraints on the sample times defined by this model. If the model does not satisfy the specified constraints during simulation, Simulink software displays an error message.

Category: Solver

Settings

Default: Unconstrained

Unconstrained

Specifies no constraints. Selecting this option causes Simulink software to display a field for entering the solver step size.

Use the **Fixed-step size (fundamental sample time)** option to specify solver step size.

Ensure sample time independent

Specifies that Model blocks inherit sample time from the context in which they are used. You cannot use a referenced model that has intrinsic sample times in a triggered subsystem or iterator subsystem. If you plan on referencing this model in a triggered or iterator subsystem, you should select `Ensure sample time independent` so that Simulink can detect sample time problems while unit testing this model.

- “Referenced Model Sample Times”
- “S-Functions That Specify Sample Time Inheritance Rules” (Simulink Coder)
- “Conditionally Execute Referenced Models”

Simulink software checks to ensure that this model can inherit its sample times from a model that references it without altering its behavior. Models that specify a step size (i.e., a base sample time) cannot satisfy this constraint. For this reason, selecting this option causes Simulink software to hide the group's step size field (see “Fixed-step size (fundamental sample time)” on page 14-61).

Specified

Specifies that Simulink software check to ensure that this model operates at a specified set of prioritized periodic sample times. Use the **Sample time properties** option to specify and assign priorities to model sample times.

“Execute Multitasking Models” (Simulink Coder) explains how to use this option for multitasking models.

Tips

During simulation, Simulink software checks to ensure that the model satisfies the constraints. If the model does not satisfy the specified constraint, then Simulink software displays an error message.

Dependencies

This parameter is enabled only if the solver **Type** is set to Fixed-step.

Selecting Unconstrained enables the following parameters:

- **Fixed-step size (fundamental sample time)**
- **Treat each discrete rate as a separate task**
- **Higher priority value indicates higher task priority**
- **Automatically handle rate transitions for data transfers**

Selecting Specified enables the following parameters:

- **Sample time properties**
- **Treat each discrete rate as a separate task**
- **Higher priority value indicates higher task priority**
- **Automatically handle rate transitions for data transfers**

Programmatic Use

Parameter: SampleTimeConstraint

Value: 'unconstrained' | 'STIndependent' | 'Specified'

Default: 'unconstrained'

Recommended Settings

Application	Setting
Debugging	Update optimize using the specified minimum and maximum values to Off
Traceability	No impact
Efficiency	No impact
Safety precaution	Specified or Ensure sample time independent

See Also

Related Examples

- “Referenced Model Sample Times”
- “S-Functions That Specify Sample Time Inheritance Rules” (Simulink Coder)
- “Conditionally Execute Referenced Models”
- “Function-Call Models”
- “Fixed-step size (fundamental sample time)” on page 14-61
- “Execute Multitasking Models” (Simulink Coder)
- “Solver Pane” on page 14-2

Fixed-step size (fundamental sample time)

Description

Specify the step size used by the selected fixed-step solver.

Category: Solver

Settings

Default: auto

- Entering **auto** (the default) in this field causes Simulink to choose the step size.
- If the model specifies one or more periodic sample times, Simulink chooses a step size equal to the greatest common divisor of the specified sample times. This step size, known as the fundamental sample time of the model, ensures that the solver will take a step at every sample time defined by the model.
- If the model does not define any periodic sample times, Simulink chooses a step size that divides the total simulation time into 50 equal steps.
- If the model specifies no periodic rates and the stop time is **Inf**, Simulink uses 0.2 as the step size. Otherwise, it sets the fixed-step size to

$$h_{\max} = \frac{t_{\text{stop}} - t_{\text{start}}}{50}$$

- For Sine and Signal Generator source blocks, if the stop time is **Inf**, Simulink calculates the step size using this heuristic: $h_{\max} = \min\left(0.2, \left(\frac{1}{3}\right)\left(\frac{1}{\text{Freq}_{\max}}\right)\right)$ Otherwise, the step size is:

$$h_{\max} = \min\left(\frac{t_{\text{stop}} - t_{\text{start}}}{50}, \left(\frac{1}{3}\right)\left(\frac{1}{\text{Freq}_{\max}}\right)\right)$$

where Freq_{\max} is the maximum frequency (Hz) of these blocks in the model.

Dependencies

This parameter is enabled only if the **Periodic sample time constraint** is set to **Unconstrained**.

Programmatic Use

Parameter: FixedStep

Type: character vector

Value: any valid value

Default: 'auto'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Solver Pane” on page 14-2

Sample time properties

Description

Specify and assign priorities to the sample times that this model implements.

Category: Solver

Settings

No Default

- Enter an Nx3 matrix with rows that specify the model's discrete sample time properties in order from fastest rate to slowest rate.
- Faster sample times must have higher priorities.

Format

[period, offset, priority]

period	The time interval (sample rate) at which updates occur during the simulation.
offset	A time interval indicating an update delay. The block is updated later in the sample interval than other blocks operating at the same sample rate.
priority	Execution priority of the real-time task associated with the sample rate.

See “Specify Sample Time” for more details and options for specifying sample time.

Example

```
[[0.1, 0, 10]; [0.2, 0, 11]; [0.3, 0, 12]]
```

- Declares that the model should specify three sample times.
- Sets the fundamental sample time period to 0.1 second.
- Assigns priorities of 10, 11, and 12 to the sample times.
- Assumes higher priority values indicate lower priorities — the **Higher priority value indicates higher task priority** option is not selected.

Tips

- If the model's fundamental rate differs from the fastest rate specified by the model, specify the fundamental rate as the first entry in the matrix followed by the specified rates, in order from fastest to slowest. See “Purely Discrete Systems”.
- If the model operates at one rate, enter the rate as a three-element vector in this field — for example, [0.1, 0, 10].
- When you update a model, Simulink software displays an error message if what you specify does not match the sample times defined by the model.
- If **Periodic sample time constraint** is set to Unconstrained, Simulink software assigns priority 40 to the model base sample rate. If **Higher priority value indicates higher task**

priority is selected, Simulink software assigns priorities 39, 38, 37, and so on, to subrates of the base rate. Otherwise, it assigns priorities 41, 42, 43, and so on, to the subrates.

- Continuous rate is assigned a higher priority than is the discrete base rate regardless of whether **Periodic sample time constraint** is Specified or Unconstrained.

Dependencies

This parameter is enabled by selecting Specified from the **Periodic sample time constraint** list.

Programmatic Use

Parameter: SampleTimeProperty

Type: structure

Value: any valid matrix

Default: []

Note If you specify SampleTimeProperty, you must enter the sample time properties as a structure with the following fields:

- SampleTime
 - Offset
 - Priority
-

See Also

Related Examples

- “Purely Discrete Systems”
- “Specify Sample Time”
- “Solver Pane” on page 14-2

Extrapolation order

Description

Select the extrapolation order used by the `ode14x` solver to compute a model's states at the next time step from the states at the current time step.

Category: Solver

Settings

Default: 4

- 1
Specifies first order extrapolation.
- 2
Specifies second order extrapolation.
- 3
Specifies third order extrapolation.
- 4
Specifies fourth order extrapolation.

Tip

Selecting a higher order produces a more accurate solution, but is more computationally intensive per step size.

Dependencies

This parameter is enabled by selecting `ode14x (extrapolation)` from the **Solver** list.

Programmatic Use

Parameter: `ExtrapolationOrder`

Type: integer

Value: 1 | 2 | 3 | 4

Default: 4

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Fixed Step Solvers in Simulink”
- “Solver Pane” on page 14-2

Number of Newton's iterations

Description

Specify the number of Newton's method iterations used by the `ode14x` and `ode1be` solvers to compute a model's states at the next time step from the states at the current time step.

Category: Solver

Settings

Default: 1

Minimum: 1

Maximum: 2147483647

More iterations produce a more accurate solution, but are more computationally intensive per step size.

Dependencies

This parameter is enabled by selecting any of the following solvers from the **Solver** list:

- `ode14x` (extrapolation)
- `ode1be` (Backward Euler)

Programmatic Use

Parameter: `NumberNewtonIterations`

Type: integer

Value: any valid number

Default: 1

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- "Fixed Step Solvers in Simulink"
- "Purely Discrete Systems"

- “Solver Pane” on page 14-2

Allow tasks to execute concurrently on target

Description

Enable concurrent tasking behavior for model.

Category: Solver

Settings

Default: Off

On

Enable the model to be configured for concurrent tasking.

Off

Disable the model from being configured for concurrent tasking.

Tip

- If the referenced model has a single rate, you do not need to select this check box to enable concurrent tasking behavior.

Dependencies

This option is visible only if the **Solver Type** is set to Fixed Step and the **Periodic sample time constraint** is set to Unconstrained or Specified. You can toggle it in the **Additional Parameters** section of the Solver Configuration Settings.

- When you select this parameter check box, clicking the **Configure Tasks** button displays the Concurrent Execution dialog.
- If you clear this parameter check box, these parameters are enabled:
 - **Periodic sample time constraint**
 - **Treat each discrete rate as a separate task**
 - **Automatically handle rate transition for data transfer**
 - **Higher priority value indicates higher task priority**

Programmatic Use

Parameter: ConcurrentTasks

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- “Concurrent Execution Window: Main Pane” on page 19-2
- “Solver Pane” on page 14-2

Auto scale absolute tolerance

Description

Allow the solver to dynamically adjust the absolute tolerance value for each state.

Settings

Default: On

- On
Allow the solver to adjust the absolute tolerance
- Off
Keep the absolute tolerance value fixed.

Dependencies

This option is visible when the following conditions are satisfied:

- Solver **Type** is set to **Variable-step**.
- **Solver** is not set to **discrete (no continuous states)**.

If the **Absolute Tolerance** is set to **auto**, the `AutoScaleAbsTol` parameter is turned on by default and cannot be disabled. To disable this parameter, you must first specify a finite non-negative value for `abstol`.

Programmatic Use

Parameter: `AutoScaleAbsTol`

Value: 'on' | 'off'

Default: 'on'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Absolute tolerance” on page 14-24
- “Solver Pane” on page 14-2

- "Absolute Tolerances"

Integration method

Description

Specify integration method of odeN solver

Settings

Default: ode3

ode1 (Euler)

Use the ode1 solver with a first order of accuracy

ode2 (Heun)

Use the ode2 solver with a second order of accuracy

ode3 (Bogacki-Shampine)

Use the ode3 solver with a third order of accuracy

ode4 (Runge-Kutta)

Use the ode4 solver with a fourth order of accuracy

ode5 (Dormand-Prince)

Use the ode5 solver with a fifth order of accuracy

ode8 (Dormand-Prince)

Use the ode8 solver with an eighth order of accuracy

ode14x (extrapolation)

Use the ode14x **implicit** solver.

ode1be (Backward Euler)

Use the ode1be solver. This solver uses the first-order Backward Euler integration method.

Dependencies

This option is visible when the following conditions are satisfied:

- Solver **Type** is set to `Variable-step`.
- **Solver** is set to `odeN (Nonadaptive)`.

Command-Line Information

Parameter: ODEIntegrationMethod

Value:ode1|ode2|ode3|ode4|ode5|ode8|ode14x|ode1be

Default: ode3

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Max step size” on page 14-16
- “Solver Pane” on page 14-2
- “Solver” on page 14-10

More About

- “Fixed-Step Continuous Solvers”
- “Fixed-Step Versus Variable-Step Solvers”

Hardware Implementation Parameters

Hardware Implementation Pane

The **Hardware Implementation** category includes parameters for configuring a hardware board to run a model. Hardware implementation parameters specify different options for building models to run on hardware boards or devices including communication connections and hardware specific parameters. **Hardware Implementation** pane parameters do not control hardware or compiler behavior. The parameters describe hardware and compiler properties for the MATLAB software.

- Specifying hardware characteristics enables simulation of the model to detect error conditions that can arise when executing code, such as hardware overflow.
- MATLAB uses the information to generate code for the platform that runs as efficiently as possible. MATLAB software also uses the information to give bit-true agreement for the results of integer and fixed-point operations in simulation and generated code.

Parameter	Description
“Hardware board” on page 15-5	Select the hardware board upon which to run your model.
“Code Generation system target file” on page 15-7	System target file that you select on the Code Generation pane.
“Device vendor” on page 15-8	Select the manufacturer of the hardware board to use to implement the system that this model represents.
“Device type” on page 15-10	Select the type of hardware to use to implement the system that this model represents.

These configuration parameters are in the **Device details** section.

Parameter	Description
“Number of bits: char” on page 15-21	Describe the character bit length for the hardware.
“Number of bits: short” on page 15-23	Describe the data bit length for the hardware.
“Number of bits: int” on page 15-25	Describe the data integer bit length for the hardware.
“Number of bits: long” on page 15-27	Describe the data bit lengths for the hardware.
“Number of bits: long long” on page 15-29	Describe the length in bits of the C <code>long long</code> data type that the hardware supports.
“Number of bits: float” on page 15-31	Describe the bit length of floating-point data for the hardware (read only).
“Number of bits: double” on page 15-32	Describe the bit-length of <code>double</code> data for the hardware (read only).
“Number of bits: native” on page 15-33	Describe the microprocessor native word size for the hardware.
“Number of bits: pointer” on page 15-35	Describe the bit-length of pointer data for the hardware.

Parameter	Description
“Number of bits: size_t” on page 15-37	Describe the bit-length of <code>size_t</code> data for the hardware.
“Number of bits: ptrdiff_t” on page 15-39	Describe the bit-length of <code>ptrdiff_t</code> data for the hardware.
“Largest atomic size: integer” on page 15-41	Specify the largest integer data type that can be atomically loaded and stored on the hardware.
“Largest atomic size: floating-point” on page 15-43	Specify the largest floating-point data type that can be atomically loaded and stored on the hardware.
“Byte ordering” on page 15-45	Describe the byte ordering for the hardware board.
“Signed integer division rounds to” on page 15-47	Describe how your compiler for the hardware rounds the result of dividing two signed integers.
“Shift right on a signed integer as arithmetic shift” on page 15-49	Describe how your compiler for the hardware fills the sign bit in a right shift of a signed integer.
“Support long long” on page 15-51	Specify that your C compiler supports the C <code>long long</code> data type. Most C99 compilers support <code>long long</code> .

These configuration parameters are in the **Advanced parameters** section.

Parameter	Description
“Test hardware is the same as production hardware” on page 2-2	Specify whether the test hardware differs from the production hardware.
“Test device vendor and type” on page 2-3	Select the manufacturer and type of the hardware to use to test the code generated from the model.
“Number of bits: char” on page 2-15	Describe the character bit length for the hardware that you use to test code.
“Number of bits: short” on page 2-17	Describe the data bit length for the hardware that you use to test code.
“Number of bits: int” on page 2-19	Describe the data integer bit length of the hardware that you use to test code.
“Number of bits: long” on page 2-21	Describe the data bit lengths for the hardware that you use to test code.
“Number of bits: long long” on page 2-23	Describe the length in bits of the C <code>long long</code> data type that the test hardware supports.
“Number of bits: float” on page 2-25	Describe the bit length of floating-point data for the hardware that you use to test code (read only).
“Number of bits: double” on page 2-26	Describe the bit-length of <code>double</code> data for the hardware that you use to test code (read only).
“Number of bits: native” on page 2-27	Describe the microprocessor native word size for the hardware that you use to test code.

Parameter	Description
"Number of bits: pointer" on page 2-29	Describe the bit-length of pointer data for the hardware that you use to test code.
"Number of bits: size_t" on page 2-30	Describe the bit-length of size_t data for the hardware that you use to test code.
"Number of bits: ptrdiff_t" on page 2-32	Describe the bit-length of ptrdiff_t data for the hardware that you use to test code.
"Largest atomic size: integer" on page 2-34	Specify the largest integer data type that can be atomically loaded and stored on the hardware that you use to test code.
"Largest atomic size: floating-point" on page 2-36	Specify the largest floating-point data type that can be atomically loaded and stored on the hardware that you use to test code.
"Byte ordering" on page 2-38	Describe the byte ordering for the hardware that you use to test code.
"Signed integer division rounds to" on page 2-40	Describe how your compiler for the test hardware rounds the result of dividing two signed integers.
"Shift right on a signed integer as arithmetic shift" on page 2-42	Describe how your compiler for the test hardware fills the sign bit in a right shift of a signed integer.
"Support long long" on page 2-44	Specify that your C compiler supports the C long long data type.
"Use Simulink Coder Features" (Simulink Coder)	Enable "Simulink Coder" features for models deployed to "Simulink Supported Hardware".
"Use Embedded Coder Features" (Embedded Coder)	Enable "Embedded Coder" features for models deployed to "Simulink Supported Hardware".

The following model configuration parameters have no other documentation.

Parameter	Description
TargetPreprocMaxBitsSint int - 32	Specify the maximum number of bits that the target C preprocessor can use for signed integer math.
TargetPreprocMaxBitsUInt int - 32	Specify the maximum number of bits that the target C preprocessor can use for unsigned integer math.

See Also

Related Examples

- "Simulink Supported Hardware"

Hardware board

Select the hardware board upon which to run your model.

Changing this parameter updates the dialog box display so that it displays parameters that are relevant to your hardware board.

To install support for a hardware board, start the Support Package Installer by selecting **Get Hardware Support Packages**. Alternatively, in the MATLAB Command Window, enter `supportPackageInstaller`.

After installing support for a hardware board, reopen the Configuration Parameters dialog box and select the hardware board.

Settings

Default: None if the specified system target file is `ert.tlc`, `realtime.tlc`, or `autosar.tlc`. Otherwise, the default is `Determine by Code Generation system target file`.

None

No hardware board is specified. The system target file specified for the model is `ert.tlc`, `realtime.tlc`, or `autosar.tlc`.

`Determine by Code Generation system target file`

Specifies that the system target file setting determines the hardware board.

`Get Hardware Support Packages`

Invokes the Support Package Installer. After you install a hardware support package, the list includes relevant hardware board names.

Hardware board name

Specifies the hardware board to use to implement the system this model represents.

Tips

- When you select a hardware board, parameters for board settings appear in the dialog box display.
- After you select a hardware board, you can select a device vendor and type.

Dependencies

The **Device vendor** and **Device type** parameter values reflect available device support for the selected hardware board.

When you select a hardware board, the selection potentially changes the `Toolchain` parameter value and other configuration parameter values. For example, if you change the hardware board selection to `ARM Cortex-A9 (QEMU)`, the `Toolchain` parameter value changes to a supported toolchain, such as `Linaro Toolchain v4.8`.

Command-Line Information

Parameter: `HardwareBoard`

Type: character array

Default: 'Determine by Code Generation system target file'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Code Generation system target file

System target file that you select on the **Code Generation** pane.

See Also

“Hardware Implementation Pane” on page 15-2

Device vendor

Select the manufacturer of the hardware board to use to implement the system that this model represents.

Settings

Default: Intel

If you have installed target support packages, the list of settings can include additional manufacturers.

- AMD
- ARM Compatible
- Altera
- Analog Devices
- Apple
- Atmel
- Freescale
- Infineon
- Intel
- Microchip
- NXP
- Renesas
- STMicroelectronics
- Texas Instruments
- ASIC/FPGA
- Custom Processor

Tips

- The **Device vendor** and **Device type** fields share the command-line parameter `ProdHWDeviceType`. When specifying this parameter at the command line, separate the device vendor and device type values by using the characters `->`. For example: `'Intel->x86-64 (Linux 64)'`.
- If you have a Simulink Coder license and you want to add **Device vendor** and **Device type** values to the default set, see “Register New Hardware Devices” (Simulink Coder).

Dependencies

The **Device vendor** and **Device type** parameter values reflect available device support for the selected hardware board.

Command-Line Information

Parameter: ProdHWDeviceType

Type: string

Value: any valid value (see tips)

Default: 'Intel'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	Select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Device type

Select the type of hardware to use to implement the system that this model represents.

Settings

Default: x86–64 (Windows64)

If you have installed target support packages, the list of settings includes additional types of hardware.

AMD options:

- Athlon 64
- K5/K6/Athlon
- x86–32 (Windows 32)
- x86–64 (Linux 64)
- x86–64 (macOS)
- x86–64 (Windows64)

ARM options:

- ARM 10
- ARM 11
- ARM 7
- ARM 8
- ARM 9
- ARM Cortex-A
- ARM Cortex-M
- ARM Cortex-R
- ARM Cortex
- ARM 64-bit (LP64)
- ARM 64-bit (LLP64)

Altera options:

- SoC (ARM CortexA)

Analog Devices options:

- ADSP–CM40x (ARM Cortex-M)
- Blackfin
- SHARC
- TigerSHARC

Apple options:

- ARM64

Atmel options:

- AVR
- AVR (32-bit)
- AVR (8-bit)

Freescale options:

- 32-bit PowerPC
- 68332
- 68HC08
- 68HC11
- ColdFire
- DSP563xx (16-bit mode)
- HC(S)12
- MPC52xx
- MPC5500
- MPC55xx
- MPC5xx
- MPC7xxx
- MPC82xx
- MPC83xx
- MPC85xx
- MPC86xx
- MPC8xx
- S08
- S12x
- StarCore

Infineon options:

- C16x, XC16x
- TriCore

Intel options:

- x86-32 (Windows32)
- x86-64 (Linux 64)
- x86-64 (macOS)
- x86-64 (Windows64)

Microchip options:

- PIC18
- dsPIC

NXP options:

- Cortex-M0/M0+
- Cortex-M3
- Cortex-M4

Renesas options:

- M16C
- M32C
- R8C/Tiny
- RH850
- RL78
- RX
- RZ
- SH-2/3/4
- V850

STMicroelectronics:

- ST10/Super10

Texas Instruments options:

- C2000
- C5000
- C6000
- MSP430
- Stellaris Cortex-M3
- TMS470
- TMS570 Cortex-R4

ASIC/FPGA options:

- ASIC/FPGA

Tips

- Before you specify the device type, select the device vendor.
- To view parameters for a device type, click the arrow button to the left of **Device details**.
- Selecting a device type specifies the hardware device to define system constraints:
 - Default hardware properties appear as the initial values.
 - You cannot change parameters with only one possible value.

- Parameters with more than one possible value provide a list of valid values.

The following table lists values for each device type.

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
AMD															
Athlon 64	8	16	32	64	64	64	64	64	64	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
K5/K6/Athlon	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
x86-32 (Windows32)	8	16	32	32	64	32	32	32	32	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
x86-64 (Linux 64)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
x86-64 (macOS)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
x86-64 (Windows64)	8	16	32	32	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
ARM Compatible															
ARM 7/8/9/10	8	16	32	32	64	32	32	32	32	Long	Float	Little Endian	Zero	✓	<input type="checkbox"/>
ARM 11	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
ARM Cortex	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
ARM 64-bit (LP64)	8	16	32	64	64	64	64	64	64	Long	Double	Little Endian	Zero	✓	✓
ARM 64-bit (LLP64)	8	16	32	32	64	64	64	64	64	Long	Double	Little Endian	Zero	✓	✓
Altera															
SoC (ARM Cortex A)	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
Analog Devices															
ADSP-CM40x(ARM Cortex-M)	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
Blackfin	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
SHARC	32	32	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	<input type="checkbox"/>
TigerSHARC	32	32	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
Apple															
ARM64	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
Atmel															
AVR	8	16	16	32	64	8	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
AVR (32-bit)	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
AVR (8-bit)	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
Freescale															
32-bit PowerPC	8	16	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	<input type="checkbox"/>
68332	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>
68HC08	8	16	16	32	64	8	8	16	8	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>
68HC11	8	16	16	32	64	8	8	16	16	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>
ColdFire	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>
DSP563xx (16-bit mode)	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
DSP5685x	8	16	16	32	64	16	16	16	16	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
HC(S)12	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
MPC52xx, MPC5500, MPC55xx, MPC5xx, PC5xx, MPC7xxx, MPC82xx, MPC83xx, MPC86xx, MPC8xx	8	16	32	32	64	32	32	32	32	Long	None	Big Endian	Zero	✓	<input type="checkbox"/>
MPC85xx	8	16	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	<input type="checkbox"/>
S08	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>
S12x	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>
StarCore	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
Infineon															
C16x, XC16x	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
TriCore	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
Intel															
x86-32 (Windows32)	8	16	32	32	64	32	32	32	32	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
x86-64 (Linux 64)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
x86-64 (macOS)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
x86-64 (Windows64)	8	16	32	32	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	<input type="checkbox"/>
Microchip															
PIC18	8	16	16	32	64	8	8	24	24	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
dsPIC	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
NXP															
Cortex-M0/M0+	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
Cortex-M3	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
Cortex-M4	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
Renesas															
M16C	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
M32C	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
R8C/Tiny	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
RH850	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
RL78	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
RX	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
RZ	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
SH-2/3/4	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	<input type="checkbox"/>
V850	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
STMicroelectronics															
ST10/Super10	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
Texas Instruments															
C2000	16	16	16	32	64	16	32	16	16	Int	None	Little Endian	Zero	✓	<input type="checkbox"/>
C5000	16	16	16	32	64	16	16	16	16	Int	None	Big Endian	Zero	✓	<input type="checkbox"/>

Key:	float and double (not listed) always equal 32 and 64, respectively														
	Round to = Signed integer division rounds to														
	Shift right = Shift right on a signed integer as arithmetic shift														
	Long long = Support long long														
Device vendor / Device type	Number of bits									Largest atomic size		Byte ordering	Round to	Shift right	Long long
	char	short	int	long	long long	native	pointer	size_t	ptrdiff_t	int	float				
C6000	8	16	32	40	64	32	32	32	32	Int	None	Little Endian	Zero	✓	<input type="checkbox"/>
MSP430	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	<input type="checkbox"/>
Stellaris Cortex-M3	8	16	32	32	6	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
TMS470	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	<input type="checkbox"/>
TMS570 Cortex-R4	8	16	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	<input type="checkbox"/>
ASIC/FPGA															
ASIC/FPGA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

- The **Device vendor** and **Device type** fields share the command-line parameter `ProdHWDeviceType`. When specifying this parameter at the command line, separate the device vendor and device type values by using the characters `->`. For example: `'Intel->x86-64 (Linux 64)'`.
- If you have a Simulink Coder license and you want to add **Device vendor** and **Device type** values to the default set, see “Register New Hardware Devices” (Simulink Coder).

Dependencies

The **Device vendor** and **Device type** parameter values reflect available device support for the selected hardware board.

Menu options that are available in the menu depend on the **Device vendor** parameter setting.

With the exception of device vendor ASIC/FPGA, selecting a device type sets the following parameters:

- **Number of bits: char**
- **Number of bits: short**
- **Number of bits: int**
- **Number of bits: long**
- **Number of bits: long long**
- **Number of bits: float**
- **Number of bits: double**
- **Number of bits: native**
- **Number of bits: pointer**
- **Largest atomic size: integer**
- **Largest atomic size: floating-point**
- **Byte ordering**
- **Signed integer division rounds to**
- **Shift right on a signed integer as arithmetic shift**
- **Support long long**

Whether you can modify the setting of a device-specific parameter varies according to device type.

Command-Line Information

Parameter: ProdHWDeviceType

Type: string

Value: any valid value (see tips)

Default: 'Intel->x86-64 (Windows64)'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Number of bits: char

Description

Describe the character bit length for the selected hardware.

Category: Hardware Implementation

Settings

Default: 8

Minimum: 8

Maximum: 32

Enter a value from 8 through 32.

Tip

All values must be a multiple of 8.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: ProdBitPerChar

Type: integer

Value: any valid value

Default: 8

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific

Application	Setting
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Number of bits: short

Description

Describe the data bit length for the selected hardware.

Category: Hardware Implementation

Settings

Default: 16

Minimum: 8

Maximum: 32

Enter a value from 8 through 32.

Tip

All values must be a multiple of 8.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: ProdBitPerShort

Type: integer

Value: any valid value

Default: 16

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific

Application	Setting
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Number of bits: int

Description

Describe the data integer bit length of the selected hardware.

Category: Hardware Implementation

Settings

Default: 32

Minimum: 8

Maximum: 32

Enter a number from 8 through 32.

Tip

All values must be a multiple of 8.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: ProdBitPerInt

Type: integer

Value: any valid value

Default: 32

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific

Application	Setting
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Number of bits: long

Description

Describe the data bit lengths for the selected hardware.

Category: Hardware Implementation

Settings

Default: 32

Minimum: 32

Maximum: 128

Enter a value from 32 through 128.

Tip

All values must be a multiple of 8 and from 32 through 128.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: ProdBitPerLong

Type: integer

Value: any valid value

Default: 32

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific

Application	Setting
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Number of bits: long long

Description

Describe the length in bits of the C `long long` data type for the selected hardware.

Category: Hardware Implementation

Settings

Default: 64

Minimum: 64

Maximum: 128

The number of bits that represent the C `long long` data type.

Tips

- Use the C `long long` data type only if your C compiler supports `long long`.
- You can change the value of this parameter for custom targets only. For custom targets, all values must be a multiple of 8 and be between 64 and 128.

Dependencies

- **Enable long long** enables use of this parameter.
- The value of this parameter must be greater than or equal to the value of **Number of bits: long**.
- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: `ProdBitPerLongLong`

Type: integer

Value: any valid value

Default: 64

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific

Application	Setting
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Number of bits: float

Description

Describe the bit length of floating-point data for the selected hardware (read only).

Category: Hardware Implementation

Settings

Default: 32

Always equals 32.

Command-Line Information

Parameter: ProdBitPerFloat

Type: integer

Value: 32 (read-only)

Default: 32

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Number of bits: double

Description

Describe the bit-length of double data for the selected hardware (read only).

Category: Hardware Implementation

Settings

Default: 64

Always equals 64.

Command-Line Information

Parameter: ProdBitPerDouble

Type: integer

Value: 64 (read only)

Default: 64

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Number of bits: native

Description

Describe the microprocessor native word size for the selected hardware.

Category: Hardware Implementation

Settings

Default: 64

Minimum: 8

Maximum: 64

Enter a value from 8 through 64.

Tip

All values must be a multiple of 8.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: ProdWordSize

Type: integer

Value: any valid value

Default: 64

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific

Application	Setting
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Number of bits: pointer

Description

Describe the bit-length of pointer data for the selected hardware.

Category: Hardware Implementation

Settings

Default: 64

Minimum: 8

Maximum: 64

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: ProdBitPerPointer

Type: integer

Value: any valid value

Default: 64

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)

- Specifying Production Hardware Characteristics (Simulink Coder)

Number of bits: size_t

Description

Describe the bit-length of `size_t` data for the selected hardware.

If `ProdEqTarget` is off, an Embedded Coder processor-in-the-loop (PIL) simulation checks this setting with reference to the target hardware. If `ProdEqTarget` is on, the PIL simulation checks the `ProdBitPerSizeT` setting.

Category: Hardware Implementation

Settings

Default: 64

Value must be 8, 16, 24, 32, 40, 64, or 128 *and* greater or equal to the value of `int`.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: `ProdBitPerSizeT`

Type: integer

Value: any valid value

Default: 64

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- “Verification of Code Generation Assumptions” (Embedded Coder)

Number of bits: ptrdiff_t

Description

Describe the bit-length of `ptrdiff_t` data for the selected hardware.

If `ProdEqTarget` is off, an Embedded Coder processor-in-the-loop (PIL) simulation checks this setting with reference to the target hardware. If `ProdEqTarget` is on, the PIL simulation checks the `ProdBitPerPtrDiffT` setting.

Category: Hardware Implementation

Settings

Default: 64

Value must be 8, 16, 24, 32, 40, 64, or 128 *and* greater or equal to the value of `int`.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: `ProdBitPerPtrDiffT`

Type: integer

Value: any valid value

Default: 64

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- “Verification of Code Generation Assumptions” (Embedded Coder)

Largest atomic size: integer

Description

Specify the largest integer data type that can be atomically loaded and stored on the selected hardware.

Category: Hardware Implementation

Settings

Default: Char

Char

Specifies that `char` is the largest integer data type that can be atomically loaded and stored on the hardware.

Short

Specifies that `short` is the largest integer data type that can be atomically loaded and stored on the hardware.

Int

Specifies that `int` is the largest integer data type that can be atomically loaded and stored on the hardware.

Long

Specifies that `long` is the largest integer data type that can be atomically loaded and stored on the hardware.

LongLong

Specifies that `long long` is the largest integer data type that can be atomically loaded and stored on the hardware.

Tip

Use this parameter, where possible, to remove unnecessary double-buffering or unnecessary semaphore protection, based on data size, in generated multirate code.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.
- You can set this parameter to LongLong only if the hardware supports the C `long long` data type and you have selected **Enable long long**.

Command-Line Information

Parameter: ProdLargestAtomicInteger

Type: string

Value: 'Char' | 'Short' | 'Int' | 'Long' | 'LongLong'

Default: 'Char'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Target specific
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Largest atomic size: floating-point

Description

Specify the largest floating-point data type that can be atomically loaded and stored on the selected hardware.

Category: Hardware Implementation

Settings

Default: Float

Float

Specifies that `float` is the largest floating-point data type that can be atomically loaded and stored on the hardware.

Double

Specifies that `double` is the largest floating-point data type that can be atomically loaded and stored on the hardware.

None

Specifies that there is no applicable setting or not to use this parameter in generating multirate code.

Tip

Use this parameter, where possible, to remove unnecessary double-buffering or unnecessary semaphore protection, based on data size, in generated multirate code.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: ProdLargestAtomicFloat

Type: string

Value: 'Float' | 'Double' | 'None'

Default: 'Float'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact

Application	Setting
Efficiency	Target specific
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- “Verification of Code Generation Assumptions” (Embedded Coder)

Byte ordering

Description

Describe the byte ordering for the selected hardware.

Category: Hardware Implementation

Settings

Default: Little Endian

Unspecified

Specifies that the code determines the endianness of the hardware. This choice is the least efficient.

Big Endian

The most significant byte appears first in the byte ordering.

Little Endian

The least significant byte appears first in the byte ordering.

Dependencies

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: ProdEndianness

Type: string

Value: 'Unspecified' | 'LittleEndian' | 'BigEndian'

Default: 'Little Endian'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact

Application	Setting
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Signed integer division rounds to

Description

Describe how your compiler for the hardware rounds the result of dividing two signed integers.

Category: Hardware Implementation

Settings

Default: Zero

Undefined

Choose this option if neither Zero nor Floor describes the compiler behavior, or if that behavior is unknown.

Zero

If the quotient is between two integers, the compiler chooses the integer that is closer to zero as the result.

Floor

If the quotient is between two integers, the compiler chooses the integer that is closer to negative infinity.

Tips

- To simulate rounding behavior of the C compiler that you use to compile generated code, use the **Integer rounding mode** parameter for blocks. This setting appears on the **Signal Attributes** pane of the parameter dialog boxes of blocks that can perform signed integer arithmetic, such as the Product block.
- For most blocks, the value of **Integer rounding mode** completely defines rounding behavior. For blocks that support fixed-point data and the Simplest rounding mode, the value of **Signed integer division rounds to** also affects rounding. For details, see "Rounding" (Fixed-Point Designer).
- For more information on how this parameter affects code generation, see Hardware Implementation Options (Simulink Coder).
- This table lists the compiler behavior described by the options for this parameter.

N	D	Ideal N/D	Zero	Floor	Undefined
33	4	8.25	8	8	8
-33	4	-8.25	-8	-9	-8 or -9
33	-4	-8.25	-8	-9	-8 or -9
-33	-4	8.25	8	8	8 or 9

Dependency

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: ProdIntDivRoundTo

Type: string

Value: 'Floor' | 'Zero' | 'Undefined'

Default: 'Zero'

Recommended settings

Application	Setting
Debugging	No impact for simulation or during development. Undefined for production code generation.
Traceability	No impact for simulation or during development. Zero or Floor for production code generation.
Efficiency	No impact for simulation or during development. Zero for production code generation.
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Shift right on a signed integer as arithmetic shift

Description

Describe how your compiler for the hardware fills the sign bit in a right shift of a signed integer.

Category: Hardware Implementation

Settings

Default: On

On

Generates simple, efficient code whenever the Simulink model performs arithmetic shifts on signed integers.

Off

Generates fully portable but less efficient code to implement right arithmetic shifts.

Tips

- Select this parameter if the C compiler implements a signed integer right shift as an arithmetic right shift.
- An arithmetic right shift fills bits vacated by the right shift with the value of the most significant bit. The most significant bit indicates the sign of the number in twos complement notation.

Dependency

- Selecting a device by using the **Device vendor** and **Device type** parameters sets a device-specific value for this parameter.
- This parameter is enabled only if you can modify it for the selected hardware.

Command-Line Information

Parameter: ProdShiftRightIntArith

Type: string

Value: 'on' | 'off'

Default: 'on'

Recommended settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	On

Application	Setting
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)

Support long long

Description

Specify that your C compiler supports the C long long data type. Most C99 compilers support long long.

Category: Hardware Implementation

Settings

Default: Off

On

Enables use of C long long data type for simulation and code generation on the hardware.

Off

Disables use of C long long data type for simulation or code generation on the hardware.

Tips

- This parameter is enabled only if the selected hardware supports the C long long data type.
- If your compiler does not support C long long, do not select this parameter.

Dependencies

This parameter enables **Number of bits: long long**.

Command-Line Information

Parameter: ProdLongLongMode

Type: string

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	On (execution, ROM)

Application	Setting
Safety precaution	No recommendation for simulation without code generation. For simulation with code generation, select your Device vendor and Device type if they are available in the drop-down list. If your Device vendor and Device type are not available, set device-specific values by using Custom Processor.

See Also

- “Hardware Implementation Pane” on page 15-2
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- “Number of bits: long long” on page 15-29

Signal Properties Dialog Box

Data Transfer Options for Concurrent Execution

In this section...
“Specify data transfer settings” on page 16-2
“Data transfer handling option” on page 16-2
“Extrapolation method (continuous-time signals)” on page 16-2
“Initial condition” on page 16-2

This tab displays the data transfer options for configuring models for targets with multicore processors. To enable this tab, in the Model Explorer for the model, right-click **Configuration**, then select the **Show Concurrent Execution** option.

Specify data transfer settings

Enable custom data transfer settings. For more information, see “Configure Data Transfer Settings Between Concurrent Tasks”.

Data transfer handling option

Select a data transfer handling option. For more information, see “Configure Data Transfer Settings Between Concurrent Tasks”.

Extrapolation method (continuous-time signals)

Select a data transfer extrapolation method. For more information, see “Configure Data Transfer Settings Between Concurrent Tasks”.

Initial condition

For discrete signals, this parameter specifies the initial input on the reader side of the data transfer. It applies for data transfer types **Ensure Data Integrity Only** and **Ensure deterministic transfer (maximum delay)**. Simulink does not allow this value to be Inf or NaN.

For continuous signals, the extrapolation method of the initial input on the reader side of the data transfer uses this parameter. It applies for data transfer types **Ensure Data Integrity Only** and **Ensure deterministic transfer (maximum delay)**. Simulink does not allow this value to be Inf or NaN.

For more information, see “Configure Data Transfer Settings Between Concurrent Tasks”.

See Also

Signal Properties

Simulink Preferences Window

Font Styles for Models

Font Styles Overview

Configure font options for blocks, lines, and annotations.

Configuration

New models use these styles. For details, see “Specify Fonts in Models”.

- 1 Use the lists to specify font types, styles, and sizes to apply to new block diagrams.
- 2 Click **OK**.

Simulink Mask Editor

- “Mask Editor Overview” on page 18-2
- “Dialog Control Operations” on page 18-30
- “Specify Data Types Using DataTypeStr Parameter” on page 18-33
- “Design a Mask Dialog Box” on page 18-39

Mask Editor Overview

In this section...
“Parameters & Dialog Pane” on page 18-2
“Code Pane” on page 18-16
“Icon Pane” on page 18-19
“Constraints” on page 18-27
“Additional Options” on page 18-28

A mask is a custom user interface for a block that hides the block's contents, making it appear to the user as an atomic block with its own icon and parameter dialog box.

The **Mask Editor** dialog box helps you create and customize the block mask. The **Mask Editor** dialog box opens when you create or edit a mask. You can access the **Mask Editor** dialog box by any of these options:

To create mask,

- In the **Modeling** tab, under **Component**, click **Create System Mask**.
- Select the block and on the **Block** tab, in the **Mask** group, click **Create Mask**. The Mask Editor opens.

To edit mask,

- On the **Block** tab, in the **Mask** group, click **Edit Mask**.
- Right-click the block and select **Mask > Edit Mask**.

Note You can also use the keyboard shortcut **CTRL + M** to open Mask Editor.

The **Mask Editor** dialog box contains a set of tabbed panes, each of which enables you define a feature of the mask. These tabs are:

- “Parameters & Dialog Pane” on page 18-2: To design mask dialog boxes.
- “Code Pane” on page 18-16: To initialize a masked block using MATLAB code.
- “Icon Pane” on page 18-19: To create block mask icons.
- “Constraints” on page 18-27: To create constraints.

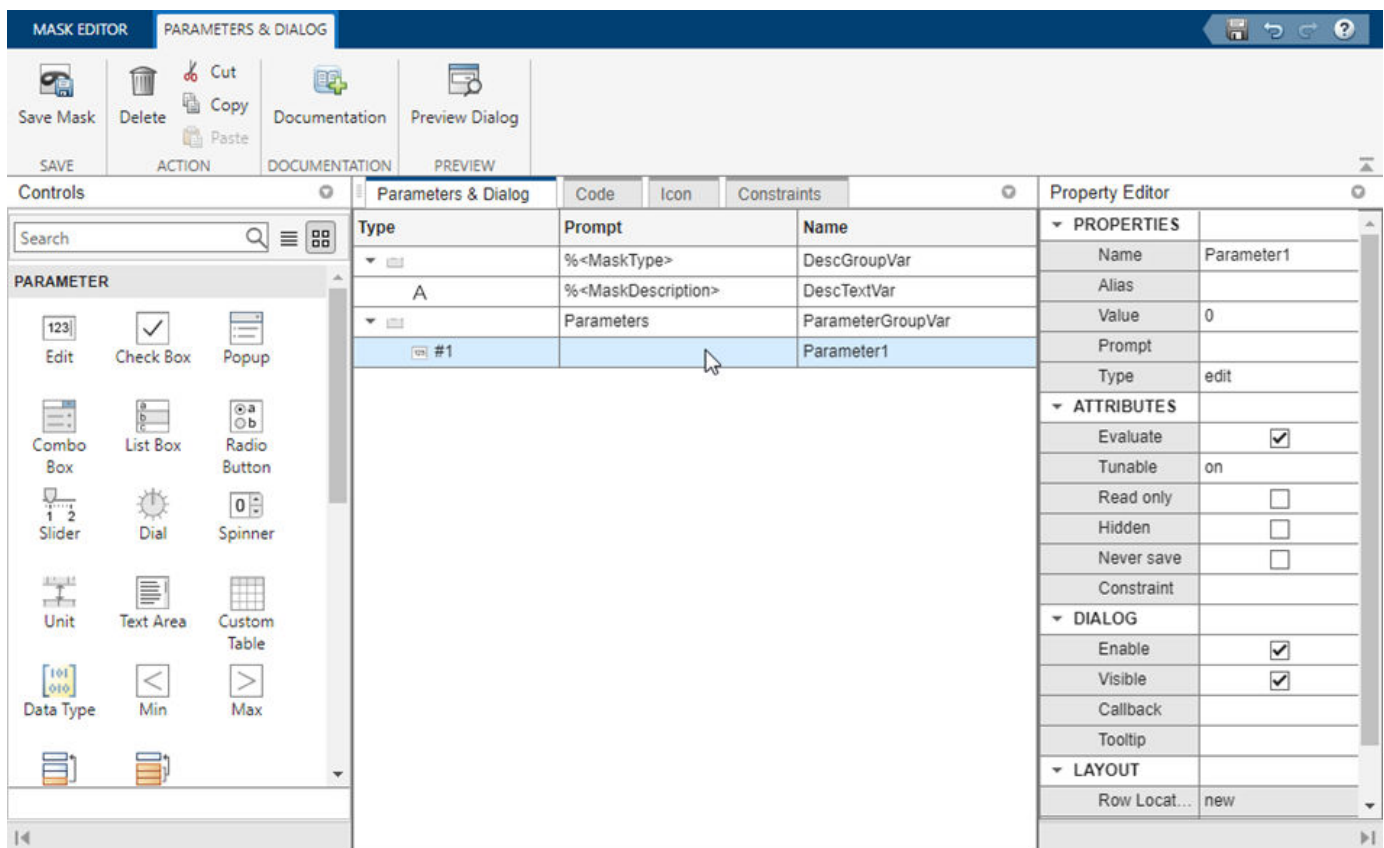
Note For information on creating and editing a block mask from command line, see “Control Masks Programmatically”.

Parameters & Dialog Pane

- “Controls” on page 18-4

- “Dialog box” on page 18-9
- “Property editor” on page 18-10
- “Documentation Pane” on page 18-14
- “Type” on page 18-15
- “Description” on page 18-15
- “Help” on page 18-15
- “Provide a URL” on page 18-15
- “Provide a web Command” on page 18-15
- “Provide an eval Command” on page 18-15
- “Provide Literal or HTML Text” on page 18-16

The **Parameters & Dialog** pane enables you to design mask dialog boxes using the dialog controls in the **Parameters**, **Display**, and **Action** palettes.



The **Parameters & Dialog** pane divided into these sections:







Parameter & Dialog Pane



Section	Section Description	Sub-Section	Sub-Section Description
"Controls" on page 18-4	Parameters are elements in a mask dialog box that users can interact with to add or manipulate data.	Parameter	Parameters are user inputs that take part in simulation. The Parameters palette has a set of parameter dialog controls that you can add to a mask dialog box.
		Container	
		Display	Controls on the Display palette allow you to group dialog controls in the mask dialog box and display text and images
		Action	Action controls allow you to perform some actions in the mask dialog box. For example, you can click a hyperlink or a button in the mask dialog box.
"Dialog box" on page 18-9	You can click or drag and drop dialog controls from the palettes to the Dialog box to create a mask dialog box.	NA	NA
"Property editor" on page 18-10	The Property editor allows you to view and set the properties for the Parameters , Display , Container , and Action controls.	Properties	Defines basic information on all dialog controls, such as Name , Value , Prompt , and Type .
		Attributes	Defines how a mask dialog control is interpreted. Attributes are related only to parameters.
		Dialog	Defines how dialog controls are displayed in the mask dialog box.
		Layout	Defines how dialog controls are laid out on the mask dialog box.







Controls







The controls section is sub divided into Parameters, Display, and Action sections. The Controls Table lists the different controls and their description.










Controls Table

Controls	Description	
Parameters		
	Edit	<p>Allows you to enter a parameter value by typing it into the field.</p> <p>You can associate constraints to an Edit parameter.</p>
	Check box	Accepts a Boolean value.
	Popup	<p>Allows you to select a parameter value from a list of possible values. When you select the Evaluate check box, the associated variable holds the index of the selected item. Note that the index starts from 1, and not 0. When Evaluate is disabled, the associated variable holds the string of the selected item.</p>
	Combo box	<p>Allows you to select a parameter value from a list of possible values. You can also type a value either from the list or from outside of the list. When you select the Evaluate check box, the associated variable holds the actual value of the selected item.</p> <p>You can associate constraints to a Combo box parameter.</p> <p>For more information, see the Combo box example in <code>slexMaskParameterOptionsExample</code>.</p>
	Listbox	Allows you to create a list of parameter values. All options of possible values are displayed on the mask dialog box. You can select multiple values from it.
	Radio button	Allows you to select a parameter value from a list of possible values. All options for a radio button are displayed on the mask dialog.

Controls		Description
	Slider	<p>Allows you to slide to values within a range defined by minimum and maximum values. A Slider parameter can accept input as a number or a variable name. If the specified variable is a base workspace or a model workspace variable, you can tune the variable value through the Slider.</p> <p>You can tune the values in the linear scale or logarithmic scale using the Scale drop-down menu</p> <p>You can also control the slider range dynamically. For more information, see <code>slexMaskParameterOptionsExample</code>.</p> <hr/> <p>Note Values specified for Slider are auto applied.</p>
	Dial	<p>Allows you to dial to values within a range defined by minimum and maximum values. A Dial parameter can accept input as a number or a variable name. If the specified variable is a base workspace or a model workspace variable, you can tune the variable value through the Dial.</p> <p>You can tune the values in the linear scale or logarithmic scale using the Scale drop-down menu.</p> <p>You can also control the dial range dynamically. For more information, see <code>slexMaskParameterOptionsExample</code>.</p> <hr/> <p>Note Values specified for Dial are auto applied.</p>

Controls		Description
	Spinbox	Allows you to spin through values within a range defined by minimum and maximum values. You can specify a step size for the values. Note Values specified for Spinbox are auto applied.
	DataType	Enables you to specify a data type for a mask parameter. You can associate the Min , Max , and Edit parameters with a data type parameter. For more details, see “Specify Data Types Using DataTypeStr Parameter” on page 18-33.
	Min	Specifies a minimum value for the DataTypeStr parameter.
	Max	Specifies a maximum value for the DataTypeStr parameter.
	Unit	Allows you to set the measurement units for output or input values of a masked block. The Unit parameter can accept any units of measurement as input. For example, rad/sec for angular velocity, meters/sec ² for acceleration, or distance in km or m. For more information, see slexMaskParameterOptionsExample .
	Custom Table	Allows you to add tables in the mask dialog box. You can add values as a nested cell array in the Values section of the Property editor. For more information, see slexMaskParameterOptionsExample .

Controls		Description
	Promote One-to-One	Allows you to selectively promote block parameters from underlying blocks to the mask. Click the Promote One-to-One to open the Promoted Parameter Selector dialog box. In this dialog box, you can select the block parameters that you want to promote. Click OK to close it.
	Promote Many-to-One	Allows you to promote all underlying block parameters to the mask. When you promote all parameters, the promote operation deletes parameters that have been promoted previously.
Container		
	Group box	Container to group other dialog controls and containers in the mask dialog box.
	Tab	Tab to group dialog controls in the mask dialog box. A tab is contained within a tab container. A tab container can have multiple tabs.
	Table	Container to group the Edit , Check box , and the Popup parameters in a tabular form. You can also search and sort the content listed within the Table container. For more information, see the Tables example in Dialog Layout Options and "Handling Large Number of Mask Parameters".
	Collapsible Panel	Container to group dialog controls similar to Panel . You can choose to expand or collapse the CollapsiblePanel dialog controls. For more information, see the Collapsible Panel example in Dialog Layout Options.

Controls		Description
	Panel	Container to group of dialog controls. You use a Panel for logical grouping of dialog controls.
Display		
	Text	Text displayed in the mask dialog box.
	Image	Image displayed in the mask dialog box.
	Text Area	Add a custom text or MATLAB code in the mask dialog box.
	Listbox Control	Allows you to select a value from a list of possible values. You can select multiple values (Ctrl + click).
	Tree Control	Allows you to select a value from a hierarchical tree of possible values. You can select multiple values (Ctrl + click).
	Lookup Table Control	Allows you to visualize n-dimensional table and breakpoint data
Action		
	Hyperlink	Hyperlink text displayed on the mask dialog box.
	Button	Button controls on the mask dialog box. You can program button for specific actions. You can also add an image on a button controls. For more information, see slexMaskParameterOptionsExample .

Dialog box

You can build a hierarchy of dialog controls by dragging them from a **Controls** section to the **Parameters and Dialog** tab. You can also click the palettes on the **Controls** section to add the required control to the **Parameters and Dialog** tab. You can add a maximum of 32 levels of hierarchy in the **Parameters and Dialog** tab.

The **Parameters and Dialog** displays three fields: **Type**, **Prompt**, and **Name**.

- The **Type** field shows the type of the dialog control and it cannot be edited. It also displays a sequence number for parameter dialog controls.
- The **Prompt** field shows the prompt text for the dialog control.

- The **Name** field is auto-populated and uniquely identifies the dialog controls. You can choose to add a different value (valid MATLAB name) in the **Name** field and must not match the built-in parameter name.

The **Parameter** controls are displayed in light blue background whereas the **Display** and **Action** controls are displayed in white background on the **Dialog box**.

You can move a dialog control in the hierarchy, you can copy and paste a dialog control, you can also delete a node. For more information, see “Dialog Control Operations” on page 18-30.

Property editor

The **Property editor** allows you to view and set the properties for **Parameter**, **Display**, **Container**, and **Action** dialog controls. The **Property editor** for **Parameter** is shown:

Property Editor	
▼ PROPERTIES	
Name	Parameter1
Alias	
Value	
Prompt	
Type	textarea
Text Type	Plain Text
▼ ATTRIBUTES	
Evaluate	<input type="checkbox"/>
Tunable	off
Read only	<input type="checkbox"/>
Hidden	<input type="checkbox"/>
Never save	<input type="checkbox"/>
▼ DIALOG	
Enable	<input checked="" type="checkbox"/>
Visible	<input checked="" type="checkbox"/>
Callback	
Tooltip	
▼ LAYOUT	
Row Location	new
Horizontal Stretch	<input checked="" type="checkbox"/>

You can set the following properties for **Parameter**, **Action**, and **Display** dialog controls. For more information, see the Property editor table.

Property editor

Property	Description
Properties	
Name	Uniquely identifies the dialog control in the mask dialog box. The Name property must be set for all dialog controls.
Value	Value of the Parameter . The Value property applies only to the Parameter dialog controls.
Prompt	Label text that identifies the parameters in a mask dialog box. The Prompt property applies to all dialog controls except Panel and Image dialog control.
Type	Type of the dialog control. You can use the Type field to change the Parameter and Container types. You cannot change any container type to Tab and vice versa.
Expand	Allows you to specify if the collapsible panel dialog control is expanded or collapsed, by default.
Type options	The Type options property allows you to set specific Parameter properties. The Type options property applies to the Popup , Radio button , DataTypeStr , and Promoted parameters.
File path	<p>You can add an image to a mask using the Image dialog control. You can also display an image on a Button dialog control. In either case, provide the path to the image in the File path property that is enabled for these two dialog controls. For the Button dialog control, specify an empty character vector for the Prompt property in order for the image to be displayed.</p> <p>Note that, when you provide the filepath, do not use the quote marks (' '). For example, if you want to add an image, provide the filepath as : C:\Users\User1\Image_Repository\motor.png</p>
Word wrap	The Word wrap property enables word wrapping for long text. The Word wrap property applies only for Text dialog control.
Maximum and Minimum	The Maximum and Minimum properties enable you to specify a range for controls like Spinbox , Slider , and Dial .
Step size	Allows you to specify a step size for the values. This property applies only for Spinbox dialog control.
Tooltip	Allows you to specify a tooltip for the selected dialog control type. The tooltip is visible when you hover the cursor over a dialog control on the mask dialog box. You can add tooltips for all dialog controls type except for Group box , Tab , CollapsiblePanel , and Panel .
Scale	Allows you to set the tuning scale as linear or log for Slider and Dial dialog controls.
Table Parameter	Specify table data for the Lookup Table parameter.
Table Unit	Specify units for the table data.
Table Display Name	Specify display name for the Lookup Table control.

Property	Description
Breakpoint Parameters	Specify breakpoint parameters for the Lookup Table control. For example, {'torque', 'engine speed'}
Breakpoint Units	Specify the units for breakpoint parameters. For example, {'Nm', 'rpm'}
Data Specification	You can specify data for table and breakpoint parameters by explicitly specifying the values in the parameters or through a data object
Lookup Table Object	Specify the name of the data object for the table and breakpoint parameters values
Text Type	Specify the type of text for the Text Area parameter. It can accept Plain Text , HTML Text , and MATLAB code . The Text Area parameter has the capability to process the HTML code and display the output in the mask dialog. Similarly, it can process the MATLAB code and display the output.
Attributes	
Evaluate	<p>If you enter a MATLAB expression as a mask parameter input, Simulink handles the entry in one of two ways:</p> <ol style="list-style-type: none"> 1 If the Evaluate option is selected, Simulink evaluates the expression and uses the final result of the calculation. To complete a successful evaluation, the variables of the expression must be initialized in the model or base workspace. For example, 'a + b' evaluates to 11 if the variables a and b hold the values 2 and 9, respectively. 2 If the Evaluate option is not selected, Simulink takes a literal reading of the input entry as you type it in the mask parameter dialog box. For example, 'a + b' is read as a + b. <p>The Evaluate option is selected by default for the Edit, Check Box and Popup mask parameters.</p>

Property	Description												
Tunable	<p>By default, you can change a mask parameter value during simulation. To prevent the changing of parameter value during simulation, clear the Tunable option. If the masked parameter does not support parameter tuning, Simulink ignores the Tunable option setting of a mask parameter. Such parameters are then disabled on the Mask dialog box when simulating. The available modes in Tunable are:</p> <ul style="list-style-type: none"> • off - you cannot change mask parameter values during simulation while in this mode. • on - you can change the mask parameter value during simulation. Each time you make a change the model is compiled. • run-to-run - you can change the mask parameter value during simulation but the model is not recompiled when you change any mask parameter value. While simulating elaborate models, this mode helps in reducing the model compilation time when simulated in fast restart. <p>You can also change the mask parameter value while simulating your model on fast restart mode. Depending on the value specified for the Tunable attribute and the simulation mode, the mask parameter can either be read-only or read-write.</p> <table border="1" data-bbox="678 1060 1484 1192"> <tr> <td></td> <td>off</td> <td>on</td> <td>run-to-run</td> </tr> <tr> <td>Normal</td> <td>read-only</td> <td>read-write</td> <td></td> </tr> <tr> <td>Fast Restart</td> <td>read-only</td> <td>read-write</td> <td>read-write</td> </tr> </table> <p>For information about parameter tuning and the blocks that support it, see "Tune and Experiment with Block Parameter Values".</p>		off	on	run-to-run	Normal	read-only	read-write		Fast Restart	read-only	read-write	read-write
	off	on	run-to-run										
Normal	read-only	read-write											
Fast Restart	read-only	read-write	read-write										
Read only	Indicates that the parameter cannot be modified.												
Hidden	Indicates that the parameter must not be displayed in the mask dialog box.												
Never save	Indicates that the parameter value never gets saved in the model file.												
Constraint	Allows you to add constraints to the selected parameter.												
Dialog box													
Enable	By default Enable is selected. If you clear this option, the selected control becomes unavailable for edit. Masked block users cannot set the value of the parameter.												
Visible	The selected control appears in the mask dialog box only if this option is selected.												
Callback	MATLAB code that you want Simulink to execute when a user applies a change to the selected control. Simulink uses a temporary workspace to execute the callback code.												

Property	Description
Layout	
Item location	Allows you to set the location for the dialog control to appear in the current row or a new row.
Align Prompts	Allows you to align the parameters on the mask dialog box. This option is supported for all the Display control types except Table . For more information, see Combo box Parameter.
Prompt location	Allows you to set the prompt location for the dialog control on either the top or to the left of the dialog control. You cannot set the Prompt location property for Check box , Dial , DataTypeStr , Collapsible Panel and Radiobutton .
Orientation	Allows you to specify horizontal or vertical orientation for sliders and radio buttons.
Horizontal Stretch	If this option is selected, the controls on the mask dialog box stretch horizontally when you resize the mask dialog box. By default, Horizontal Stretch check box is selected. For more information, see Horizontal Stretch Property.

Documentation Pane

The **Documentation** pane enables you to define or modify the type, description, and help text for a masked block.

The screenshot shows a dialog box titled "Documentation" with a close button (X) in the top right corner. It contains three text input areas: "Type", "Description", and "Help". The "Type" field is currently selected with a blue border. At the bottom right, there are "Cancel" and "Apply" buttons. A vertical scrollbar is visible on the right side of the "Description" and "Help" fields.

Type

The mask type is a block classification that appears in the mask dialog box and on all **Mask Editor** panes for the block. When Simulink displays a mask dialog box, it suffixes (**mask**) to the mask type. To define the mask type, enter it in the **Type** field. The text can contain any valid MATLAB character, but cannot contain line breaks.

Description

The mask description is summary help text that describe the block's purpose or function. By default, the mask description is displayed below the mask type in the mask dialog box. To define the mask description, enter it in the **Description** field. The text can contain any legal MATLAB character. Simulink automatically wraps long lines. You can force line breaks by using the **Enter** key.

Help

The Online Help for a masked block provides information in addition to that provided by the **Type** and **Description** fields. This information appears in a separate window when the masked block user clicks the **Help** button on the mask dialog box. To define the mask help, type one of these in the **Help** field:

- URL specification
- `web` or `eval` command
- Literal or HTML text

Provide a URL

If the first line of the **Help** field is a URL, Simulink passes the URL to your default web browser. The URL can begin with `https:`, `www:`, `file:`, `ftp:`, or `mailto:`. Examples:

```
https://www.mathworks.com
file:///c:/mydir/helpdoc.html
```

Once the browser is active, MATLAB and Simulink have no further control over its actions.

Provide a web Command

If the first line of the **Help** field is a `web` command, Simulink passes the command to MATLAB, which displays the specified file in the MATLAB Online Help browser. Example:

```
web([docroot '/MyBlockDoc/' get_param(gcb,'MaskType') '.html'])
```

See the MATLAB `web` command documentation for details. A `web` command used for mask help cannot return values.

Provide an eval Command

If the first line of the **Help** field is an `eval` command, Simulink passes the command to MATLAB, which performs the specified evaluation. Example:

```
eval('open My_Spec.doc')
```

See MATLAB `eval` command documentation for details. An `eval` command used for mask help cannot return values.

Provide Literal or HTML Text

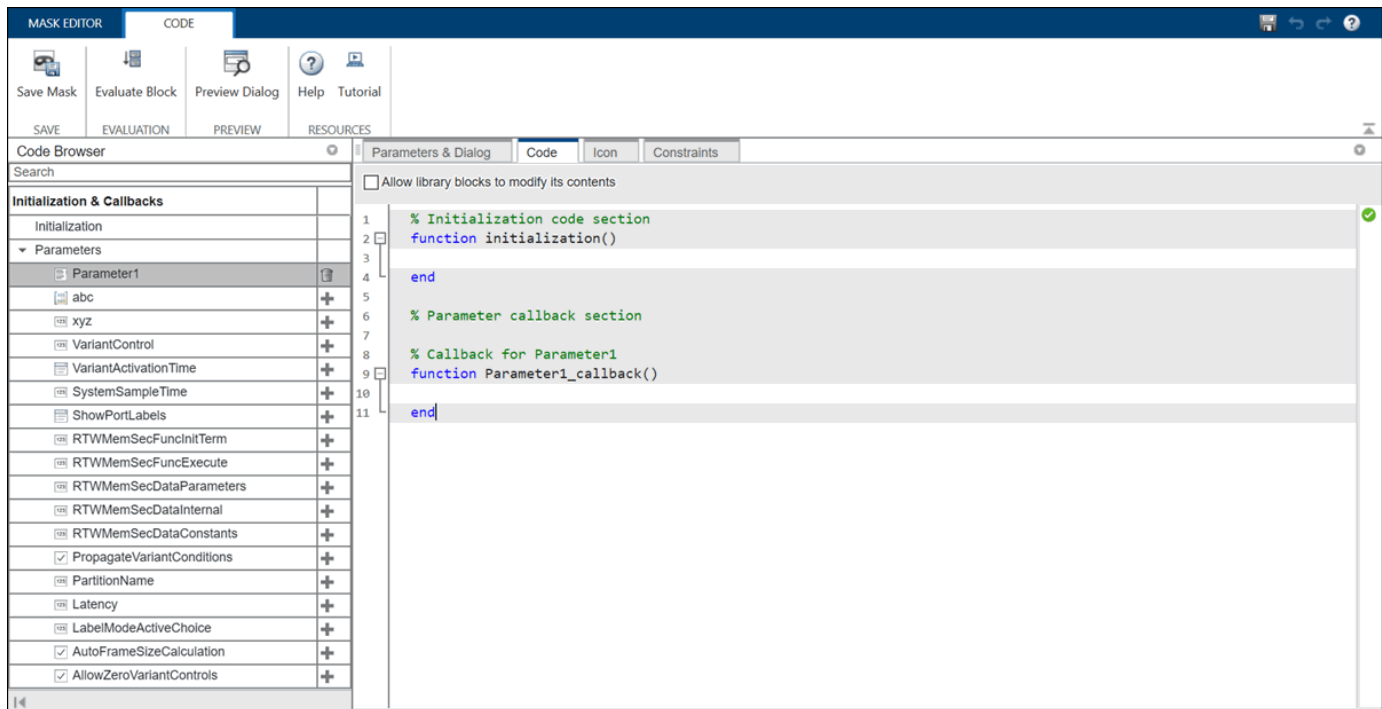
If the first line of the **Help** field is not a URL, or a `web` or an `eval` command, Simulink displays the text in the MATLAB Online Help browser under a heading that is the value of the **Mask type** field. The text can contain any legal MATLAB character, line breaks, and any standard HTML tag, including tags like `img` that display images.

Simulink first copies the text to a temporary folder, then displays the text using the `web` command. If you want the text to display an image, you can provide a URL path to the image file, or you can place the image file in the temporary folder. Use `tempdir` to find the temporary folder that Simulink uses for your system.

Code Pane

- “Dialog variables” on page 18-17
- “Initialization commands” on page 18-17
- “Rules for Initialization commands” on page 18-17
- “Allow library block to modify its contents” on page 18-18
- “Mask Parameter Callback” on page 18-18

The **Code** pane gives you an integrated view of block initialization and parameter callback code. The Mask Editor code functionalities are like those in the MATLAB Editor, with some limitations. For example, the autocomplete functionality is supported, but you cannot set a breakpoint in your code.



When you open a model, Simulink locates the visible masked blocks that reside at the top level of the model or in an open subsystem. Simulink only executes the initialization commands for these visible masked blocks if they meet either of the following conditions:

- The masked block has icon drawing commands.

Note Simulink does not initialize masked blocks that do not have icon drawing commands, even if they have initialization commands.

- The masked block belongs to a library and has the **Allow library block to modify its contents** enabled.

Initialization commands for all masked blocks in a model run when you:

- Update the diagram
- Start simulation
- Start code generation
- Click **Apply** on the dialog box

Initialization commands for an individual masked block run when you:

- Change any of the mask parameters that define the mask, such as `MaskDisplay` and `MaskInitialization`, by using the Mask Editor or the `set_param` command.
- Rotate or flip the masked block, if the icon depends on the initialization commands.
- Cause the icon to be drawn or redrawn, and the icon drawing depends on initialization code.
- Change the value of a mask parameter by using the block dialog box or the `set_param` command.
- Copy the masked block within the same model or between different models.

The **Code** pane contains the controls described in this section.

Dialog variables

The **Dialog variables** list displays the names of the dialog controls and associated mask parameters, which are defined in the **Parameters & Dialog** pane. You can also use the list to change the names of mask parameters. To change a name, double-click the name in the list. An edit field containing the existing name appears. Edit the existing name and click **Enter** or click outside the edit field to confirm your changes.

Initialization commands

Enter the initialization commands in this field. You can enter any valid MATLAB expression, consisting of MATLAB functions and scripts, operators, and variables defined in the mask workspace. Initialization commands run in the mask workspace, not the base workspace.

Rules for Initialization commands

Following rules apply for mask initialization commands:

- Do not use initialization code to create mask dialogs whose appearance or control settings change depending on changes made to other control settings. Instead, use the mask callbacks provided specifically for this purpose.
- Avoid prefacing variable names in initialization commands with `MaskParam_L_` and `MaskParam_M_`. These specific prefixes are reserved for use with internal variable names.

- Avoid using `set_param` commands to set parameters of blocks residing in masked subsystems that reside in the masked subsystem being initialized. See “Set Up Nested Masked Block Parameters” for details.

Allow library block to modify its contents

This check box is enabled only if the masked block resides in a library. Selecting this option allows you to modify the parameters of the masked block. If the masked block is a masked subsystem, this option allows you to add or delete blocks and set the parameters of the blocks within that subsystem. If this option is not selected, an error is generated when a masked library block tries to modify its contents in any way.

Mask Parameter Callback

The Code pane provides you an integrated view of the mask initialization code and the mask callback code. To add parameter callback code, click on the plus button next to the parameter from the **Parameter** list, the skeleton for the callback code appears. Enter MATLAB commands for the callback.

The screenshot displays the Simulink Mask Editor interface. The top bar shows 'MASK EDITOR' and 'CODE' tabs. Below the top bar are icons for 'Save Mask', 'Evaluate Block', 'Preview Dialog', 'Help', and 'Tutorial'. The main area is divided into a 'Code Browser' on the left and a 'Code' pane on the right. The 'Code Browser' shows a list of parameters under 'Initialization & Callbacks' > 'Parameters'. A red box highlights the plus sign next to the 'VariantActivationTime' parameter. The 'Code' pane shows the following MATLAB code:

```

1  % Initialization code section
2  function initialization()
3
4  end
5
6  % Parameter callback section
7
8  % Callback for VariantControl
9  function VariantControl_callback()
10
11 end
12
13 % Callback for VariantActivationTime
14 function VariantActivationTime_callback()
15
16 end

```

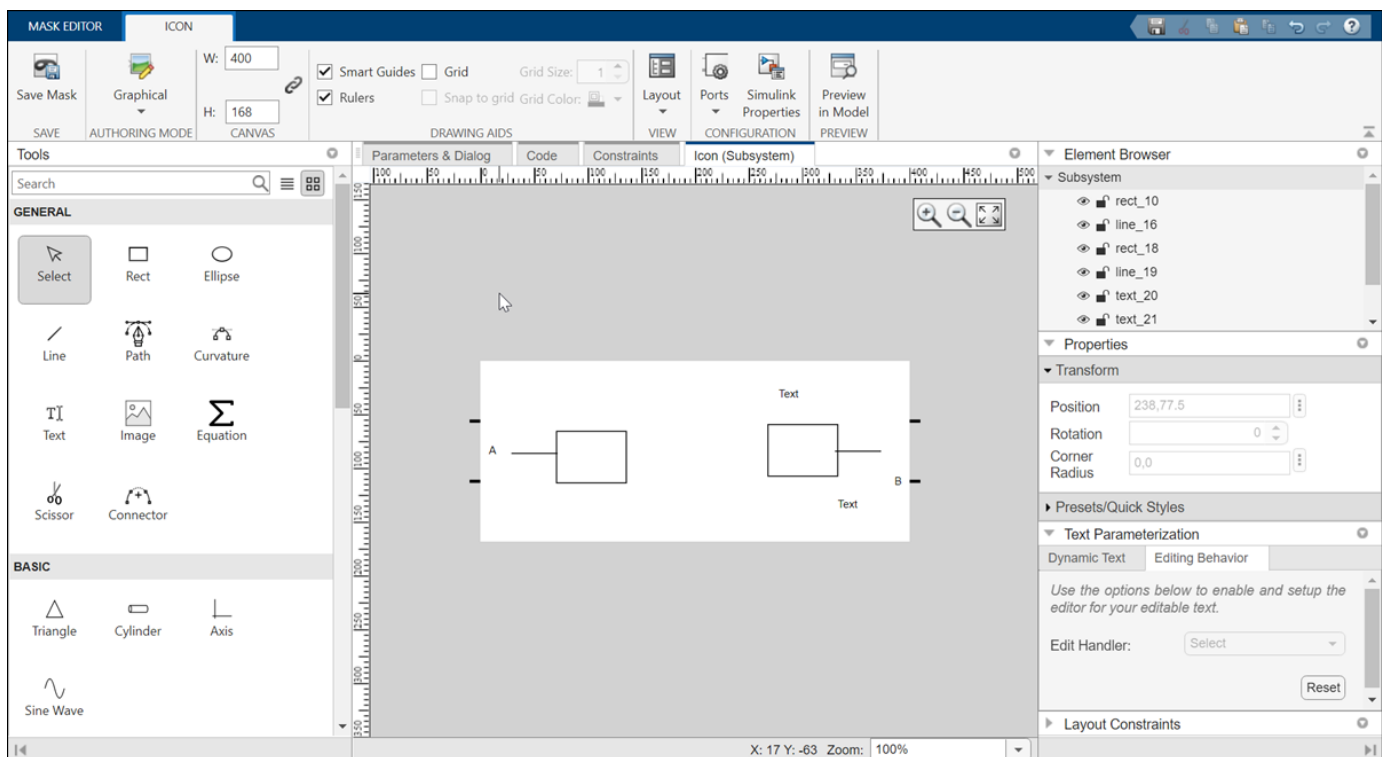
Icon Pane

- “Graphical Icon Editor” on page 18-19
- “Mask Icon Drawing Commands” on page 18-20

The **Icon** pane helps you to create a block icon that contains descriptive text, state equations, image, and graphics. You can author block icon using either Graphical Editor or Mask Drawing Commands.

Graphical Icon Editor

Graphical Editor: You can create and edit the mask icon of a block through a graphical environment. The various features in Graphical Icon Editor helps you to create icons with ease. Launch Graphical Icon Editor from Mask Editor.



- **Interactive graphical environment:** Use graphical tools like pen, curvature, text, scissor, connector, and equation (which supports LaTeX) to create rich graphical icons. Grids, smart guides, and rulers help you to create pixel-perfect icons. Apart from the drawing tools, a few built-in shapes, such as Resistor, Inductor, and Rotational Damper, are readily available
- **Element browser:** Element browser lists all the elements in the icon.
 - Hide or unhide an element in the icon.
 - Lock or unlock an element so that you do not accidentally change the shape or position of an element while working on other elements of the icon.
 - Name each element in the icon for easy identification.
- **Port binding/unbinding:** The number of ports on each block is pre-defined if you are creating or modifying the block using block context. For example, the number of ports for Simscape blocks or

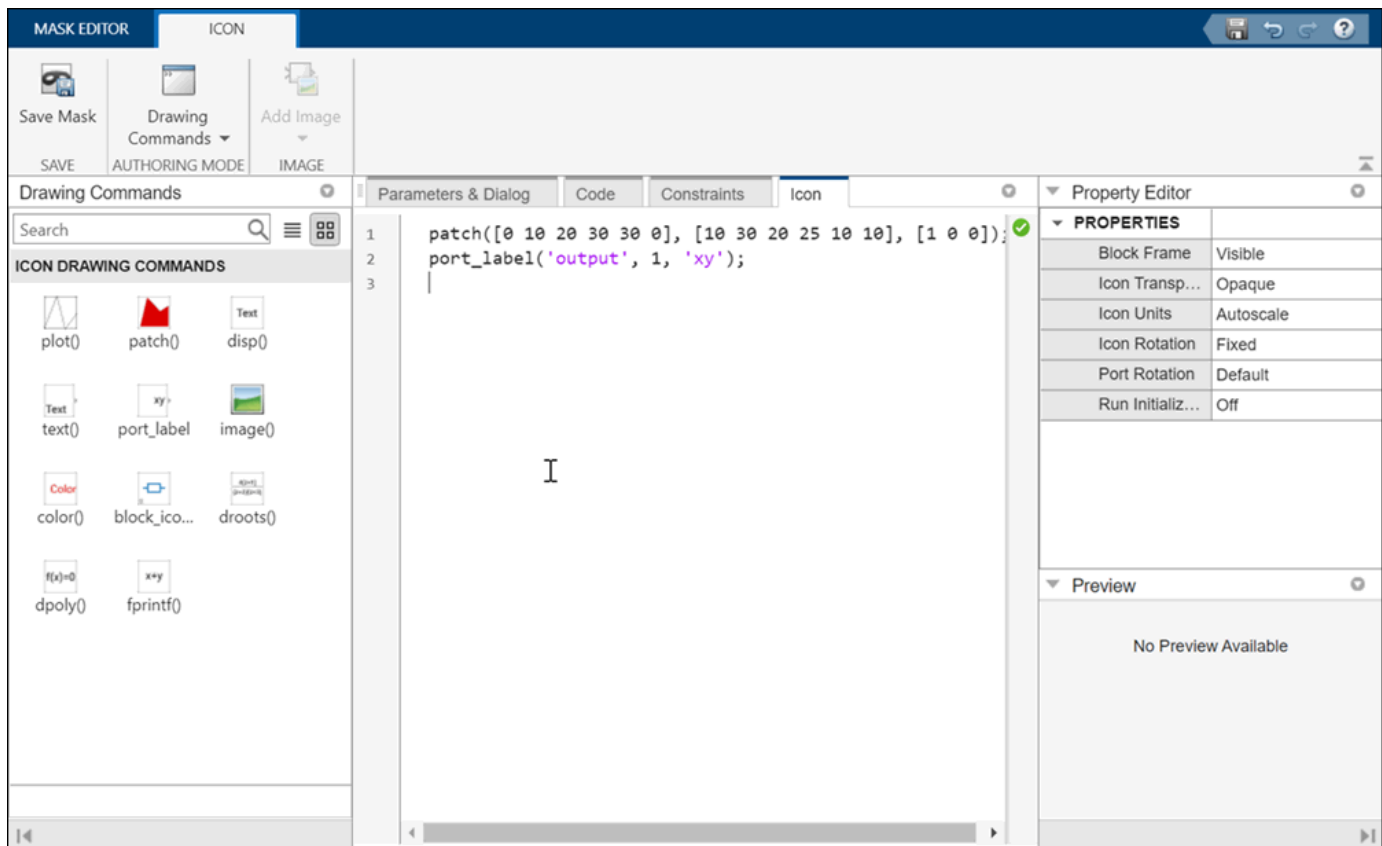
Aerospace blocks are pre-defined and they appear on the block icon. You can also define the number of ports on the block icon if you are creating or modifying a block without a block context.

- **Conditional visibility:** Hide or unhide an element of the block based on the block parameters or mask parameters.
- **Preview options:** Preview the icon in Simulink using preview options such as horizontal stretch, flip, or scale. You can also preview the icons with modified block parameters.
- **Display elements that fit the size of the icon:** The first-fit feature helps you to display only the elements that fit in the size of the icon when you resize the block.
- **Position elements relatively:** The auto layout constraint feature helps you to position each element relative to other elements on the canvas.
- **Text Parameterization:** You can view the evaluated value of a block parameter or mask parameter on the block icon. Enter the block parameter name or a placeholder in Parameter/Value that will return the text or value during runtime. To see the evaluated value of a block parameter on the block icon, preview the icon on Simulink canvas.

To know more about Graphical Icon Editor, see “Create and Edit Masked Block Icon Using Graphical Icon Editor”

Mask Icon Drawing Commands

Mask editor provides you the skeleton for each of the drawing commands. You can set an image for the mask icon. Click **Add Image** to import an image.



The Mask Icon Drawing Commands pane is divided into these sections:

- “Properties” on page 18-21: Provides a list of different controls that can be applied on the mask icon.
- “Preview” on page 18-25: Displays the preview of the block mask icon.
- “Icon drawing commands” on page 18-25: Enables you to draw mask icon by using MATLAB code.

Note You can create static and dynamic block mask icon. For more information, see “Draw Mask Icon” and `slexMaskDisplayAndInitializationExample`.

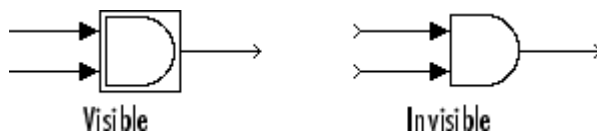
Properties

Properties available in the right pane are a list of controls that allow you to specify attributes on the mask icon. These options are,

- “Block Frame” on page 18-21
- “Icon Transparency” on page 18-21
- “Icon Units” on page 18-22
- “Icon Rotation” on page 18-23
- “Port Rotation” on page 18-23
- “Run Initialization” on page 18-25

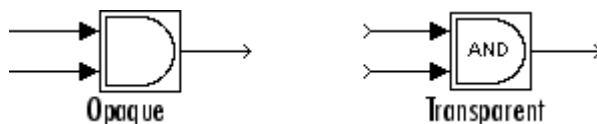
Block Frame

The block frame is the rectangle that encloses the block. You can choose to show or hide the frame by setting the **Block Frame** parameter to `Visible` or `Invisible`. The default is to make the block frame visible. For example, this figure shows visible and invisible block frames for an AND gate block.

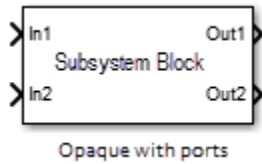


Icon Transparency

The icon transparency can be set to `Opaque`, `Opaque with ports`, or `Transparent`, based on whether you want to hide or show what is underneath the icon. The default option `Opaque` hides information such as port labels. The block frame is displayed for a transparent icon, and hidden for the opaque icon.



For a subsystem block, if you set the icon transparency to `Opaque with ports` the port labels are visible.

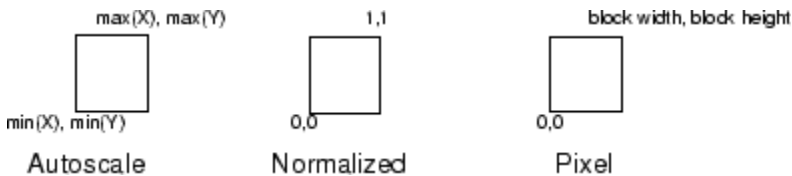


Note

- For the `Opaque` option to hide the port labels, there must be an icon drawing command added in the mask editor.
- If you set the icon transparency to `Transparent`, Simulink does not hide the block frame even if you set the **Block Frame** property to `Invisible`.

Icon Units

This option controls the coordinate system used by the drawing commands. It applies only to the `plot`, `text`, and `patch` drawing commands. You can select from among these choices: `Autoscale`, `Normalized`, and `Pixel`.



- `Autoscale` scales the icon to fit the block frame. When the block is resized, the icon is also resized. For example, this figure shows the icon drawn using these vectors:

```
X = [0 2 3 4 9]; Y = [4 6 3 5 8];
```



The lower-left corner of the block frame is (0,3) and the upper-right corner is (9,8). The range of the x-axis is 9 (from 0 to 9), while the range of the y-axis is 5 (from 3 to 8).

- `Normalized` draws the icon within a block frame whose bottom-left corner is (0,0) and whose top-right corner is (1,1). Only X and Y values from 0 through 1 appear. When the block is resized, the icon is also resized. For example, this figure shows the icon drawn using these vectors:

```
X = [.0 .2 .3 .4 .9]; Y = [.4 .6 .3 .5 .8];
```



- `Pixel` draws the icon with X and Y values expressed in pixels. The icon is not automatically resized when the block is resized. To force the icon to resize with the block, define the drawing commands in terms of the block size.

Icon Rotation

When the block is rotated or flipped, you can choose whether to rotate or flip the icon or to have it remain fixed in its original orientation. The default is not to rotate the icon. The icon rotation is consistent with block port rotation. This figure shows the results of choosing **Fixed** and **Rotates** icon rotation when the AND gate block is rotated.



Port Rotation

This option enables you to specify a port rotation type for the masked block. The choices are:

- `default`

Ports are reordered after a clockwise rotation to maintain a left-to-right port numbering order for ports along the top and bottom of the block and a top-to-bottom port numbering order for ports along the left and right sides of the block.

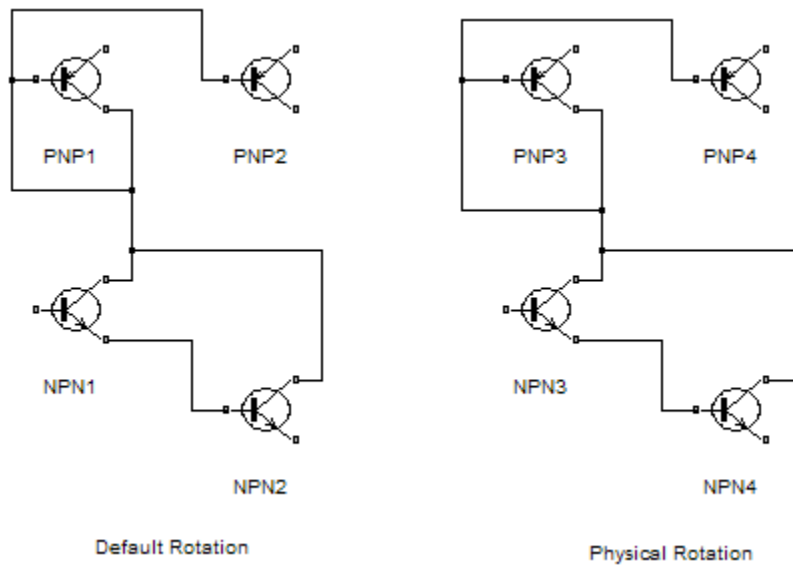
- `physical`

Ports rotate with the block without being reordered after a clockwise rotation.

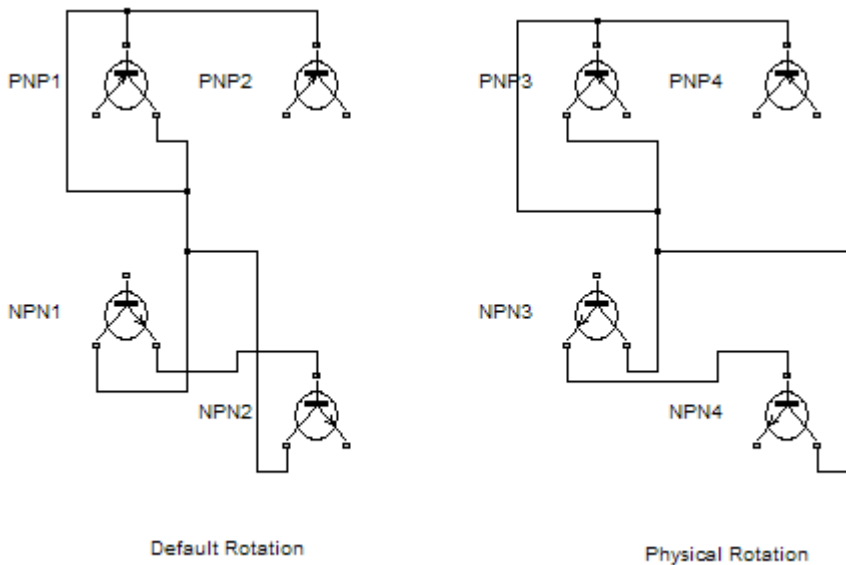
The default rotation option is appropriate for control systems and other modeling applications where block diagrams typically have a top-down and left-right orientation. It simplifies editing of diagrams, by minimizing the need to reconnect blocks after rotations to preserve the standard orientation.

Similarly, the physical rotation option is appropriate for electronic, mechanical, hydraulic, and other modeling applications where blocks represent physical components and lines represent physical connections. The physical rotation option more closely models the behavior of the devices represented (that is, the ports rotate with the block as they would on a physical device). In addition, the option avoids introducing line crossings as the result of rotations, making diagrams easier to read.

For example, the following figure shows two diagrams representing the same transistor circuit. In one, the masked blocks representing transistors use default rotation and in the other, physical rotation.



Both diagrams avoid line crossings that make diagrams harder to read. The next figure shows the diagrams after a single clockwise rotation.



Note The rotation introduces a line crossing the diagram that uses default rotation but not in the diagram that uses physical rotation. Also that there is no way to edit the diagram with default rotation to remove the line crossing. See “Flip or Rotate Blocks” for more information.

Run Initialization

The **Run initialization** option enables you to control the execution of the mask initialization commands. The choices are:

- **Off** (Default): Does not execute the mask initialization commands. When the mask drawing commands do not have dependency on the mask workspace, it is recommended to specify the value of **Run initialization** as **Off**. Setting the value to **Off** helps in optimizing Simulink performance as the mask initialization commands are not executed.
- **On**: Executes the mask initialization commands if the mask workspace is not up-to-date. When this option is specified, the mask initialization commands are executed before executing the mask drawing commands irrespective of the mask workspace dependency of the mask drawing commands.
- **Analyze**: Executes the mask initialization commands only if there is mask workspace dependency. When this option is specified, Simulink executes the mask initialization commands before executing the mask icon drawing commands. The **Analyze** option is for backward compatibility and is not recommended otherwise. It is recommended that the Simulink models from R2016b or before are upgraded using the Upgrade Advisor.

For more information, see `slexMaskDrawingExamples`.

Preview










This section displays the preview of block mask icon. Block mask preview is available only if the mask contains an icon drawing.


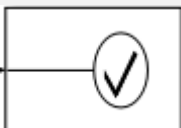
When you add an icon drawing command and click **Apply**, the preview image refreshes and is displayed in the **Preview** section of **Icon** pane.

Icon drawing commands


Add code to the editor to draw a block icon. You can use the list of commands in the left pane to draw a block icon.

Mask icon drawing commands

Drawing Command	Description	Syntax Example	Preview
color	Change drawing color of subsequent mask icon drawing commands	<code>color('red'); port_label('output',1,'Text')</code>	
disp	Display text on the masked icon.	<code>disp('Gain')</code>	
dpoly	Display transfer function on masked icon	<code>dpoly([0 0 1], [1 2 1], 'z')</code>	
droots	Display transfer function on masked icon	<code>droots([-1], [-2 -3], 4)</code>	
fprintf	Display variable text centered on masked icon	<code>fprintf('Sum = %d', 7)</code>	
image	<p>Display RGB image on masked icon</p> <hr/> <p>Note To add mask icon image from the user interface, click Mask > Add Mask Icon in the context menu.</p>	<code>image('b747.jpg')</code>	
<div data-bbox="553 1171 1130 1472" data-label="Image"> </div>			
patch	Draw color patch of specified shape on masked icon	<code>patch([0 10 20 30 30 0], [10 30 20 25 10 10],[1 0 0])</code>	
plot	Draw graph connecting series of points on masked icon	<code>plot([10 20 30 40], [10 20 10 15])</code>	
port_label	Draw port label on masked icon	<code>port_label('output', 1, 'xy')</code>	

Drawing Command	Description	Syntax Example	Preview
text	Display text at specific location on masked icon . You must select Pixels in the Icon units box.	text(5,10, 'Gain')	
block_icon	Promote icon of a block contained in a Subsystem to the Subsystem mask	block_icon(BlockName) Here, the icon of block is promoted to its Subsystem block. For more information, see slexMaskDrawingExamples.	

Note Simulink does not support mask drawing commands within anonymous functions.

The drawing commands execute in the same sequence as they are added in the text box. Drawing commands have access to all variables in the mask workspace. If any drawing command cannot successfully execute, the block icon displays question marks .

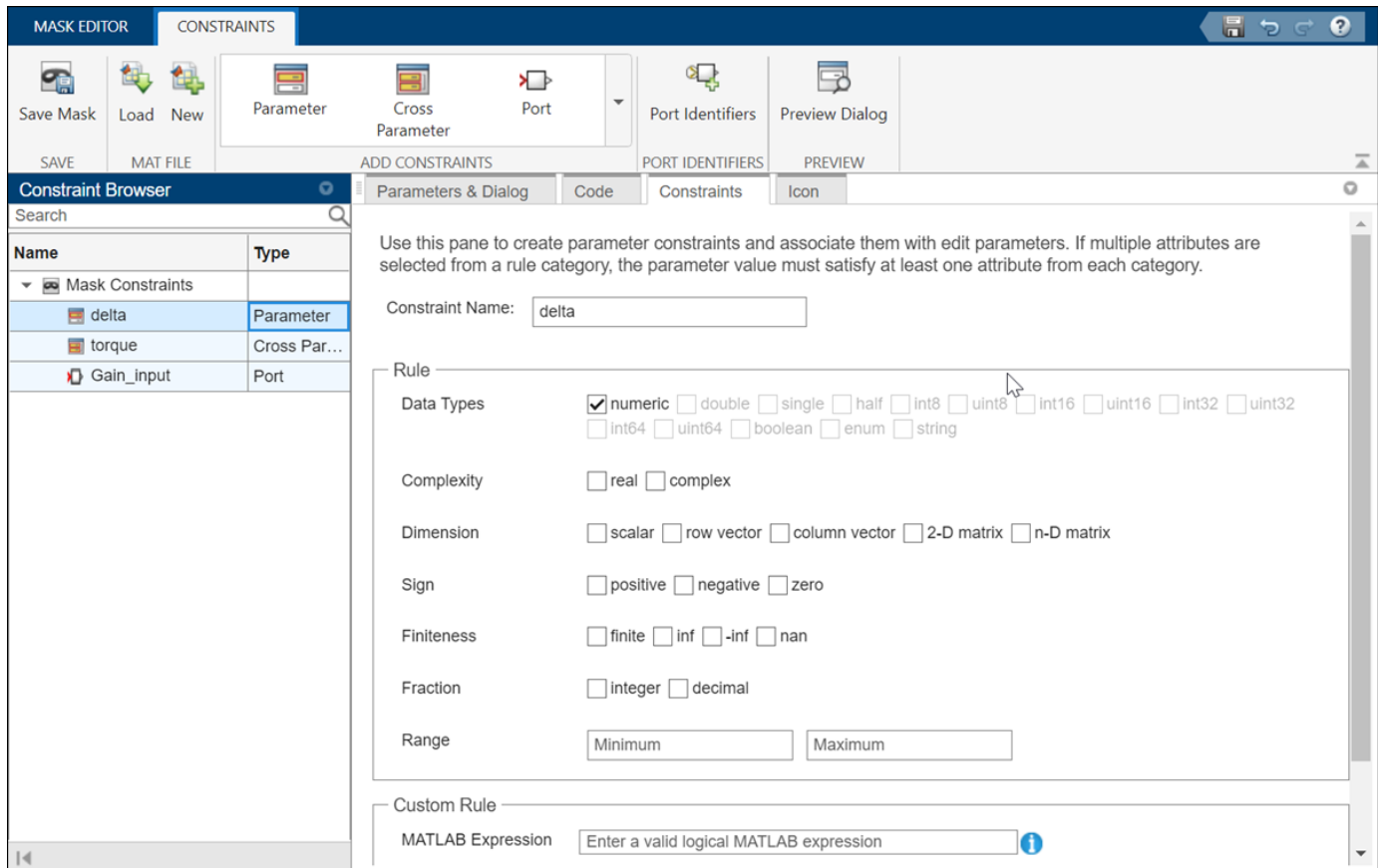
The drawing commands execute after the block is drawn in these cases:

- Changes are made and applied in the mask dialog box.
- Changes are made in the Mask Editor.
- Changes are done to the block diagram that affects the block appearance, such as rotating the block.

Constraints

Mask parameter constraints help you to create validations on a mask parameter without having to write your own validation code. There are three types of constraints, Parameter Constraint, Cross

Parameter Constraints, and Port Constraints.



Parameter Constraint: A mask can contain parameters that accept user input values. You can provide input values for mask parameters using the mask dialog box. Constraints ensure that the input for the mask parameter is within a specified range. For example, consider a masked Gain block. You can set a constraint where the input value must be between 1 and 10. If you provide an input that is outside the specified range, an error displays. Constraint Browser on the left pane helps you to manage Shared Constraints.

Cross Parameter Constraint: Cross-parameter constraints are applied among two or more **Edit** or **Combobox** type mask parameters. You can use a cross parameter constraint when you want to specify scenarios such as, Parameter1 must be greater than Parameter2.

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

Additional Options

Following buttons appear on the **Mask Editor**:

- **Save Mask** applies the mask settings and leaves the **Mask Editor** open.
- The **Preview Dialog** applies the changes you made, and opens the mask dialog box.
- The **Delete Mask** deletes the mask and closes the **Mask Editor**. To create the mask again, select the block and on the **Block** tab, in the **Mask** group, click **Create Mask**.

- **Copy Mask** copies the mask definitions from Simulink library blocks. Search for the desired block and click **Copy Mask** to import the mask definition from an existing block.
- **Evaluate Block** evaluates the callback and initialization code.

See Also

More About

- “Masking Fundamentals”
- “Create a Simple Mask”
- “Create Block Masks”
- Creating a Mask: Parameters and Dialog Pane

Dialog Control Operations

In this section...

“Moving dialog controls in the Dialog box” on page 18-30

“Cut, Copy, and Paste Controls” on page 18-30

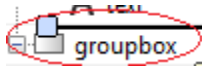
“Delete nodes” on page 18-31

“Error Display” on page 18-31

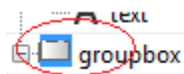
Moving dialog controls in the Dialog box

You can move dialog controls up and down in the hierarchy using drag and drop. When you drag a control, a cue line indicates the level in the hierarchy. Based on the type of dialog control, you can drag and drop controls as indicated:

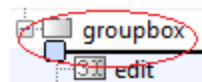
- **Drag and drop on the container dialog control in the Dialog box**
 - **Drop before it:** Adds the dialog control as a sibling before the current dialog control.



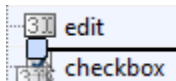
- **Drop on it:** Adds to the container as a child at the end.



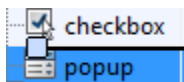
- **Drop after it:** Adds the dialog control as a sibling after the current dialog control.



- **Drag and drop on the non-container dialog control in the Dialog box**
 - **Drop before it:** Adds the dialog control before the current dialog control.



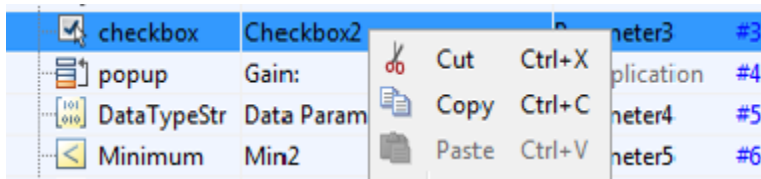
- **Drop after it:** Adds the dialog control after the current dialog control.




- **Drag and drop into Dialog box blank area**
 - The element is added to the root level node.

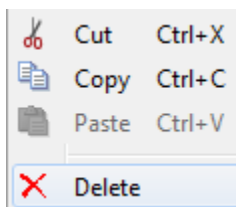
Cut, Copy, and Paste Controls

You can cut, copy, and paste dialog controls on the **Dialog box** using the context menu.



Delete nodes

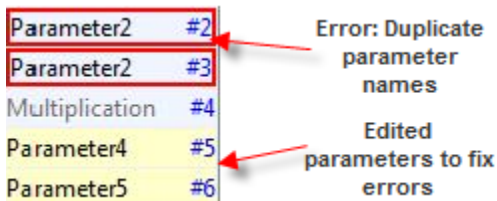
Right-click the control that you want to delete in the **Dialog box**. Select,  **Delete** from the context menu. For example, to delete a **Check box** dialog control, right-click and select **Delete**:

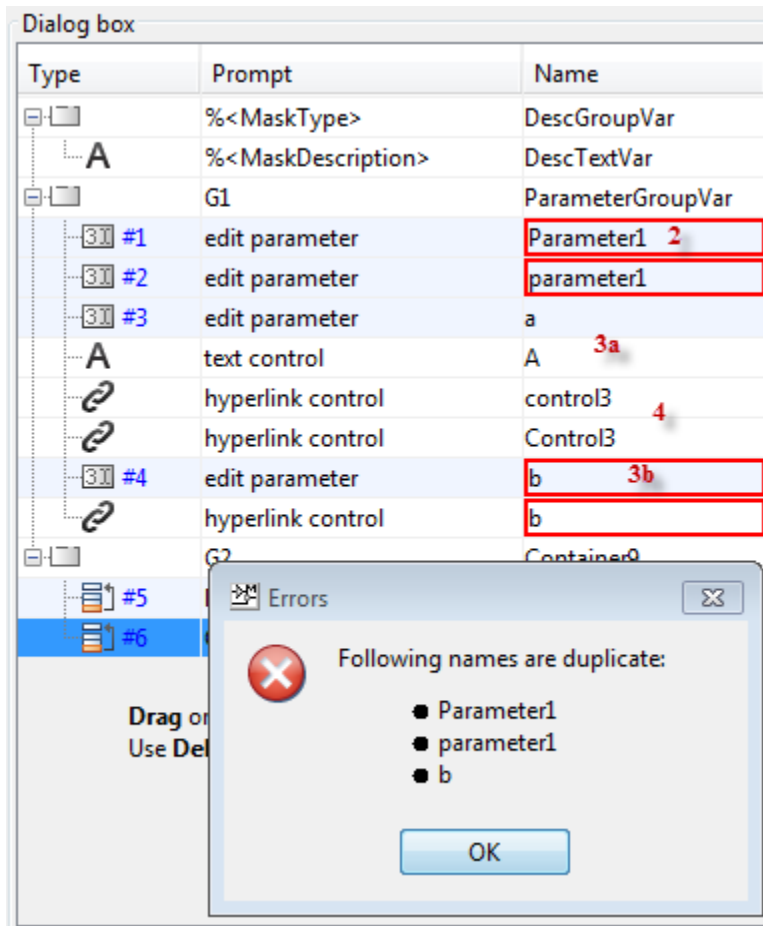


You can also use the **Delete** menu option to delete a dialog control.

Error Display

If you have errors in parameters names, such as, duplicate, invalid parameter names, or empty names, the mask editor displays the parameter names in red outline. When you edit the parameters to fix errors, the modified fields are identified by a yellow background.





- 1 Duplicate **Parameter**, **Display**, and **Action** control names are not allowed.
- 2 **Parameter** names must be unique and are case insensitive. Names varying only in lowercase and uppercase letters, are treated as duplicates. For example, Parameter1 and parameter1 are not allowed.
- 3 **Parameter**, **Display**, and **Action** control names can be same as long as different lowercase and uppercase characters are used. For example, while a and A are allowed, b and b are not allowed.
- 4 **Action** and **Display** control names are case-sensitive. For example, while Control3 and control3 are allowed, control3 and control3 are not allowed.

See Also

"Create Block Masks"

Specify Data Types Using DataTypeStr Parameter

Similar to any mask parameter, the **DataType** parameter can be added on a mask dialog box from the Mask Editor. Adding the **DataType** parameter to the mask dialog box allows the end user of the block to specify the acceptable data types for the associated **Edit** type parameter. While defining the mask, you can specify single or multiple data types for the **Edit** parameter. The end user of the block can select from one of these data types. Specifying a data type for the **Edit** parameter defines a rule for the input value that can be provided through the mask dialog box.

The **DataType** parameter also allows you to specify a minimum and maximum value for the **Edit** parameter. You can do so by using the **Min** and **Max** mask parameters and associating these parameters to the **DataType** parameter. **DataType** parameter can be used to do fixed-point analysis.

Associate Data Types to Edit Parameter

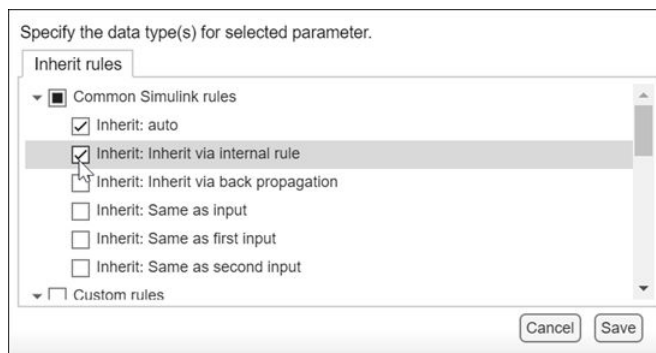
- 1 Open the model in which you want to mask a block. For example, open the `DataTypeStr` model in Mask Parameters.
- 2 Select the Subsystem block on the **Subsystem Block** tab, in the **Mask** group, click **Create Mask**.

Note If you are editing an existing mask, to open the Mask Editor, on the **Subsystem Block** tab, in the **Mask** group, click **Edit Mask**.

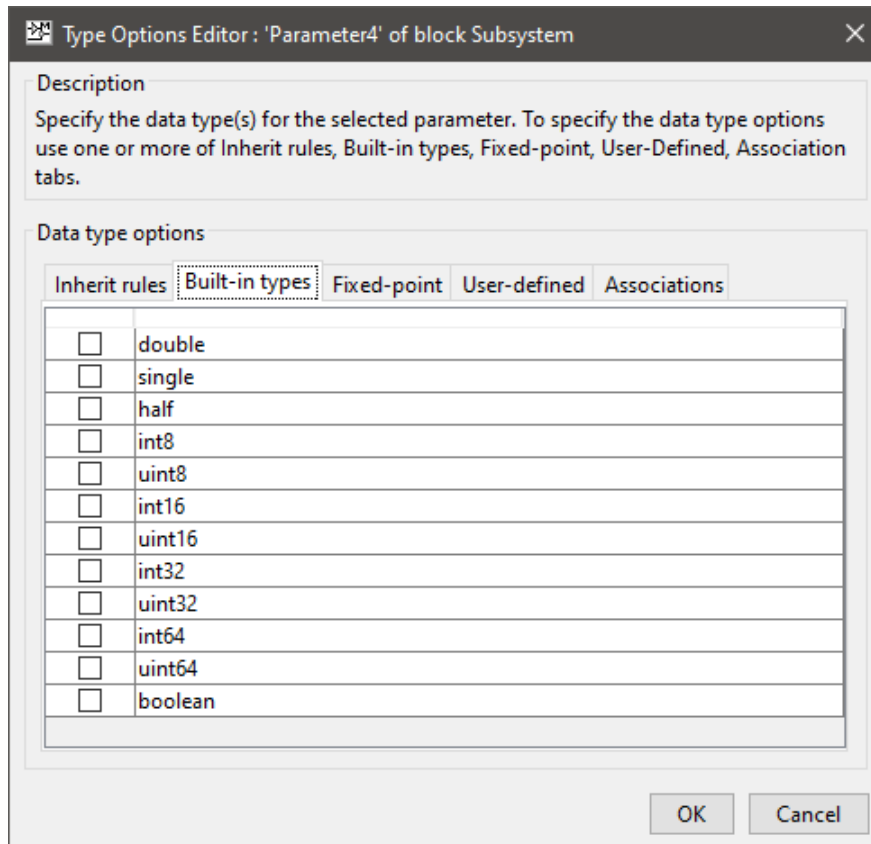
- 3 In the Mask Editor, click the **Parameters & Dialog** pane and add the **Edit**, **Min**, **Max**, **DataType** parameters.

Type	Prompt	Name
	%<MaskType>	DescGroupVar
A	%<MaskDescription>	DescTextVar
Parameters		ParameterGroupVar
#1	Gain	Parameter1
#2	Output Min	Parameter2
#3	Output Max	Parameter3
#4	Output data type	Parameter4

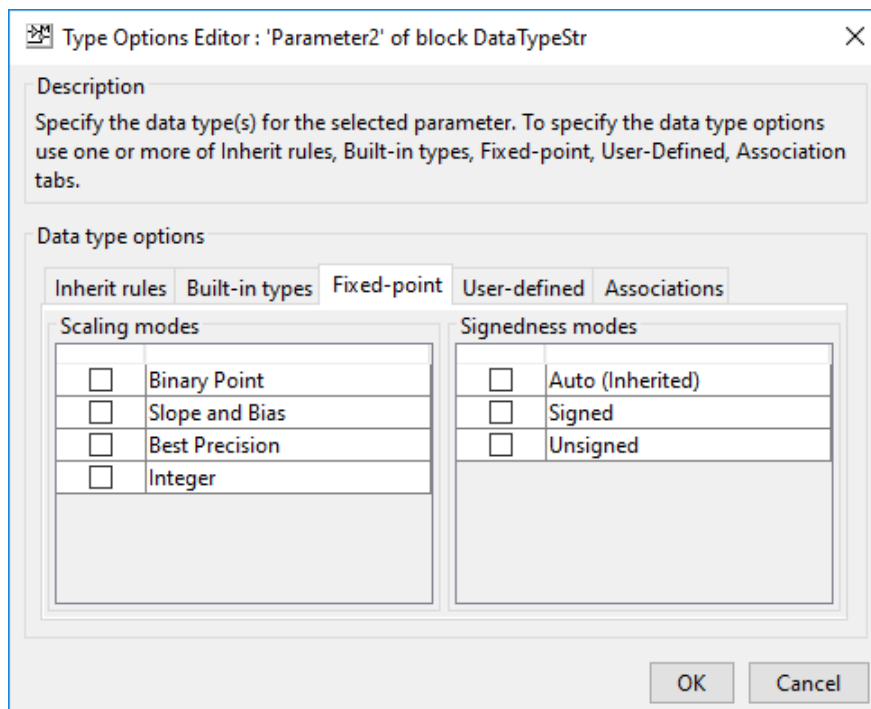
- 4 To specify data types for the **Edit** parameter, select **Data Type** in the **Dialog box** section of the Mask Editor and click the button next to **Type options** in the **Property editor** pane. The **Type options** editor has a tabbed user interface containing these tabs for data type rules.
 - a **Inherit rules-** Specify inheritance rules for determining the data types. The inherit rules are grouped under three categories: Common Simulink rules, Custom rules, and Advanced Simulink rules. By default, the Common Simulink rules and Advanced Simulink rules are available under **Inherit rules** tab. The Advanced rules section allows you to inherit rules from breakpoint data, constant value, gain, table data, logic data, accumulator, product output, and Simulink. It also allows you to have same word length as input and have same data types for all ports. The Custom rules are listed under **Inherit rules** tab only if there are any custom inheritance rules registered on the MATLAB search path. For definitions of some Inherit rules, see “Data Type Inheritance Rules”.



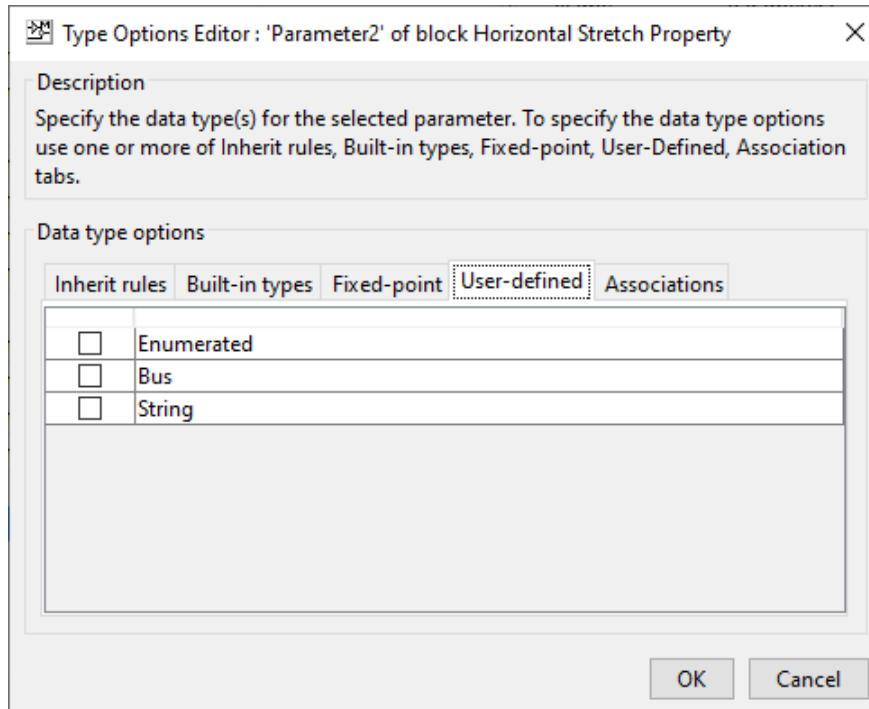
- b **Built-in types:** Specify one or more built-in Simulink data types, such as `double` or `single`. For more information, see “Data Types Supported by Simulink”.



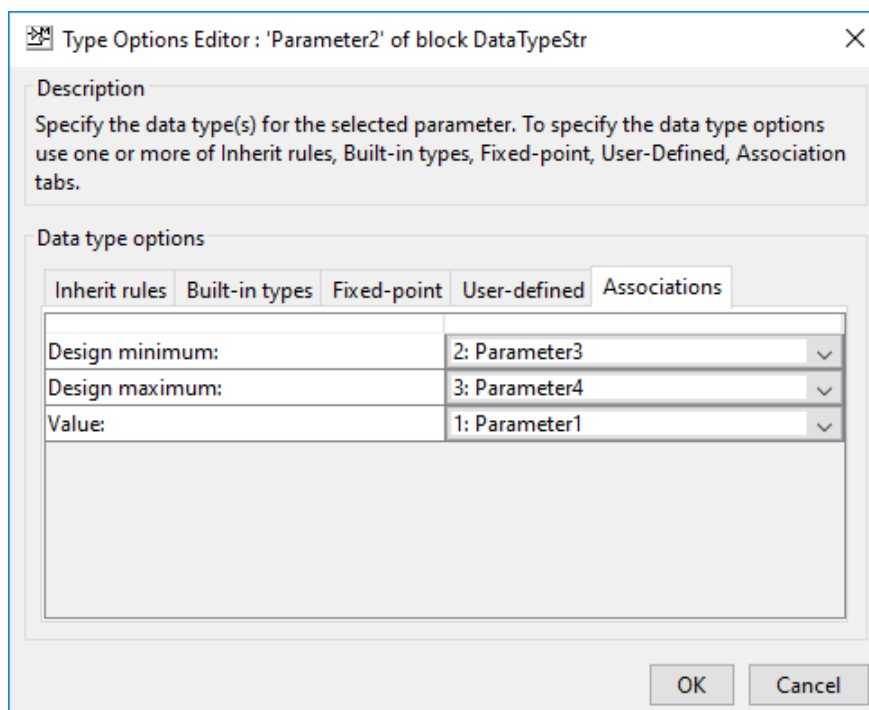
- c **Fixed-point:** Specify the scaling and signed modes for a fixed-point data type. For more information, see “Specifying a Fixed-Point Data Type”.



- d **User-defined:** Specify a bus object, an enumerated (enum) data type, or a string. For more information, see “Specify an Enumerated Data Type”, “Specify a Bus Object Data Type”, and “Simulink Strings”.



- e **Associations:** Associate a data type parameter with an **Edit** parameter. You can also associate the **Min** and **Max** parameters to the **Edit** parameter.



- 5 To save the rules selection, click **OK** in the **Type Options Editor**.
- 6 To save changes and exit the Mask Editor, click **OK**.

View DataTypeStr Programmatically

You can use the `Simulink.Mask.get` command in the MATLAB command window to view the data type values specified for a block mask. MATLAB uses a predefined nomenclature to represent the data type information in the command line.

This example shows how to view the `DataTypeStr` Parameter for the example model `Mask Parameters` programmatically.

```
maskobj = Simulink.Mask.get(gcb)
```

```
maskobj =
```

```
Mask with properties:
```

```

    Type: ''
    Description: ''
    Help: ''
    Initialization: ''
    SelfModifiable: 'off'
    Display: ''
    IconFrame: 'on'
    IconOpaque: 'opaque'
    RunInitForIconRedraw: 'off'
    IconRotate: 'none'
    PortRotate: 'default'
    IconUnits: 'autoscale'
    Parameters: [1x4 Simulink.MaskParameter]
    BaseMask: [0x0 Simulink.Mask]
    ParameterConstraints: [0x0 Simulink.Mask.Constraints]
    BlockConstraintRules: [0x0 Simulink.Mask.BlockConstraints]
    ConstraintParamAssociator: [0x0 Simulink.Mask.ConstraintParamAssociator]

```

```
maskobj.getParameter('DataTypeStrParameter')
```

```
ans =
```

```
MaskParameter with properties:
```

```

Type: 'unidt({a=4|2|3|1}{i=Inherit: auto|Inherit: Inherit via internal rule}{b=double|single})'
TypeOptions: {0x1 cell}
    Name: 'DataTypeStrParameter'
    Prompt: 'Output data type'
    Value: 'Inherit: auto'
    Evaluate: 'on'
    Tunable: 'off'
    NeverSave: 'off'
    Hidden: 'off'
    ReadOnly: 'off'
    Enabled: 'on'
    Visible: 'on'
    ShowTooltip: 'on'
    Callback: ''
    Alias: ''

```

The result displays the properties that are defined for the `DataTypeStr` parameter. This example defines the nomenclature for the specified type options:

```
Type: 'unidt({a=4|2|3|1}{i=Inherit: auto|Inherit: Inherit via internal rule}{b=double|single})'
```

Here, `Type` displays the values specified for the **DataTypeStr** parameter and has these definitions:

- a defines **Associations** and its corresponding values are 4, 2, 3, 1. These values are index numbers for the parameter and represent the **DataTypeStr**, **Min**, **Max**, and **Edit** parameters sequentially.
- i defines the **Inherit rules** and its corresponding value as `Inherit: Same as first input`.
- b defines the **Built-in types** and its corresponding value as `double` and `single`.

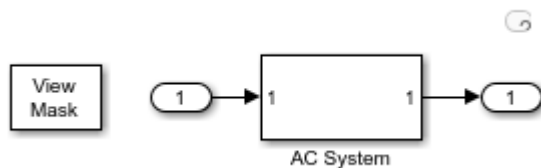
See Also

"Create Block Masks"

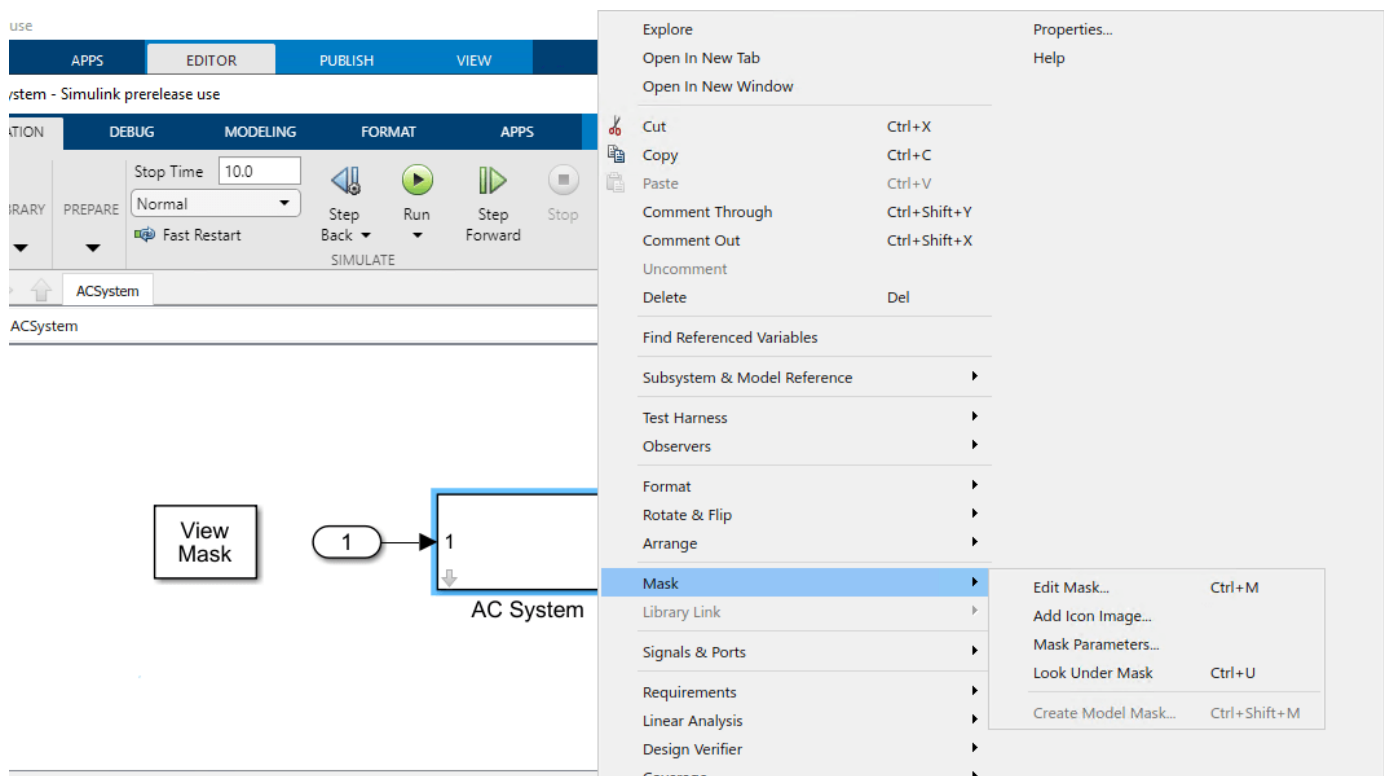
Design a Mask Dialog Box

This example shows how to create a mask dialog box using the Parameters & Dialog pane of the Mask Editor. When you mask a block, you encapsulate the details of the block logic and create a custom interface for the block.

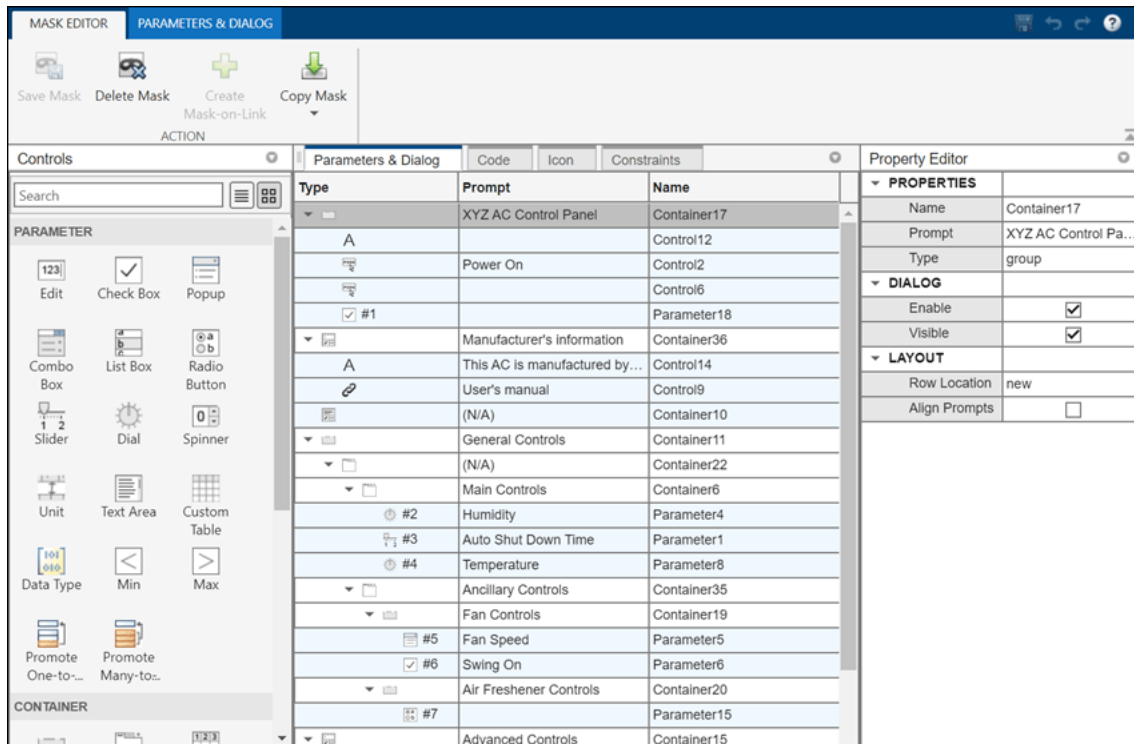
Consider this model containing a masked Subsystem block called AC System. The AC System block contains an air conditioning system. For more on masking subsystems, see “Create a Simple Mask”.



To open the Mask Editor, right-click the AC System block, then select **Mask > Edit Mask**.

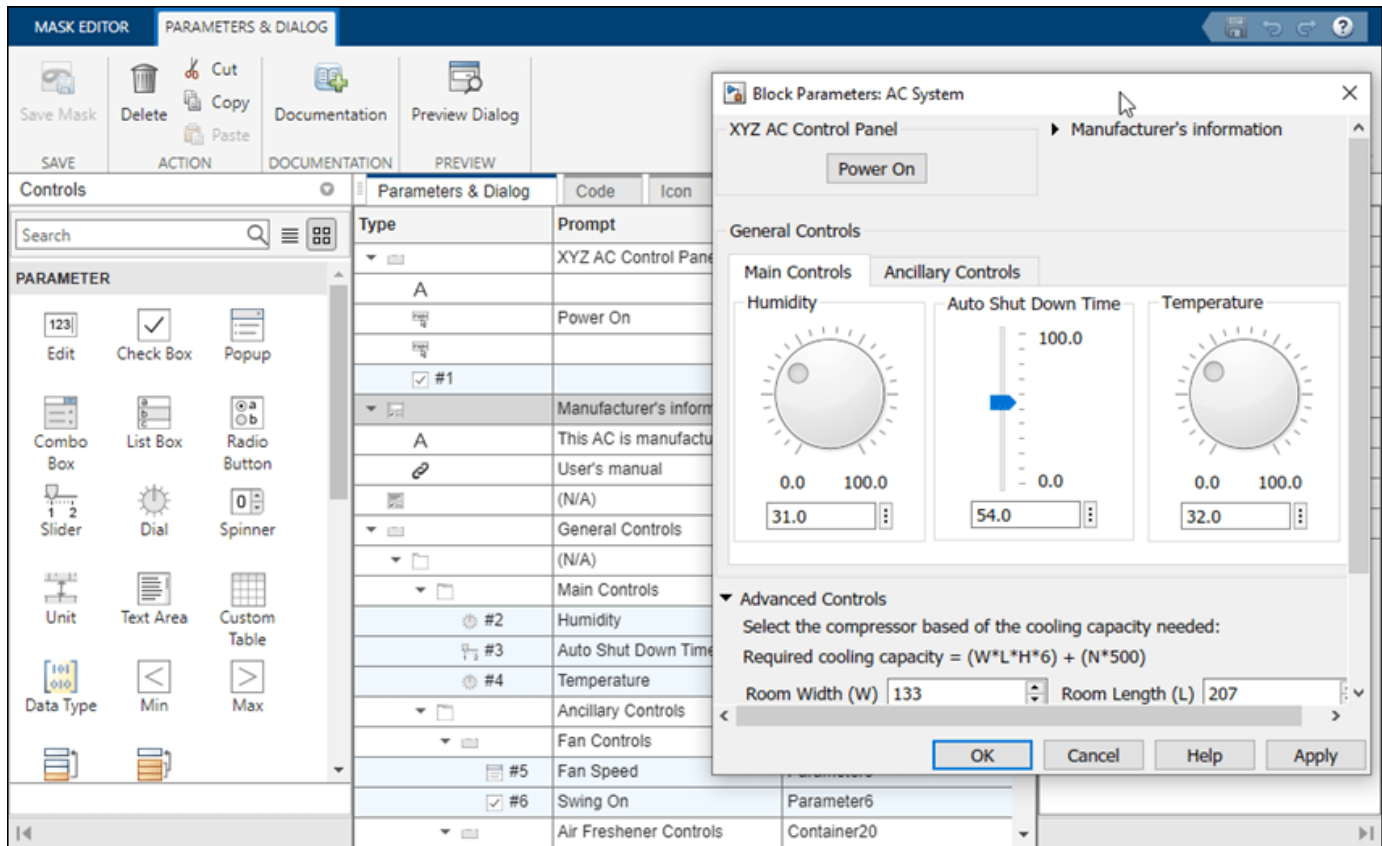


In the Mask Editor, use the Parameters & Dialogs pane to add controls on the mask dialog box and manage the mask dialog box layout. Select items from the Controls section to add parameters to the mask dialog box. Use the Property editor section to edit parameter properties.



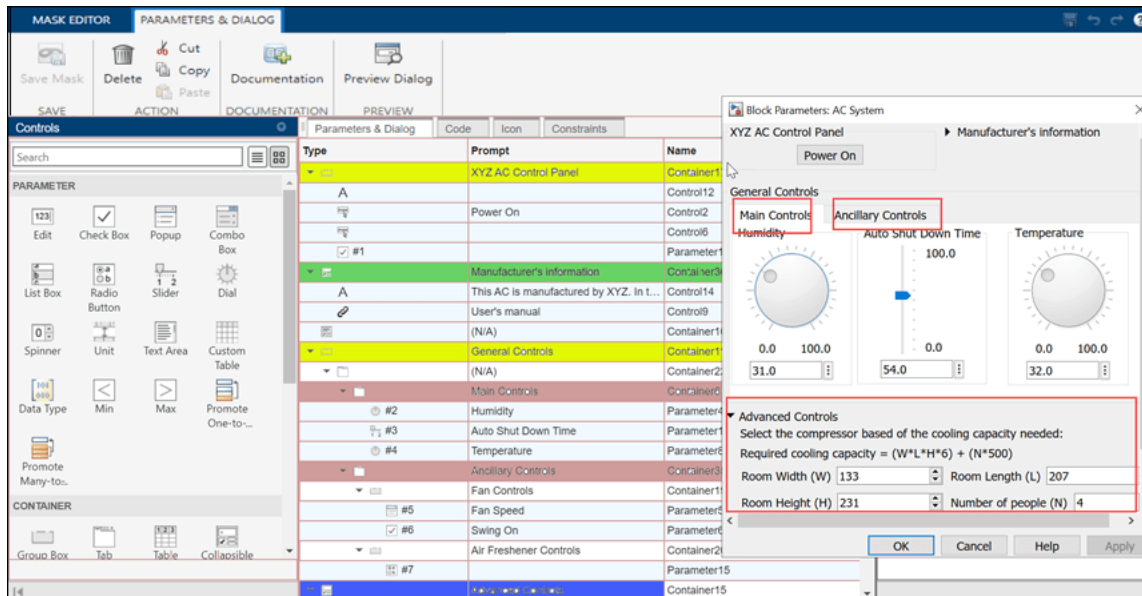
For example, in the **Controls** panel, click **Collapsible Panel**. Observe that a collapsible panel container is now added in the **Dialog box** section. In the **Prompt** column, type a value to be displayed on the mask dialog box. For example, **Manufacturer's Information**. The **Name** column gets populated automatically when you add a control. You can change this value. You can change the name and the type of this parameter from the **Property editor**.

Edit the properties of collapsible panel in the Property editor. Click **Preview** to view the mask dialog box as you build it.

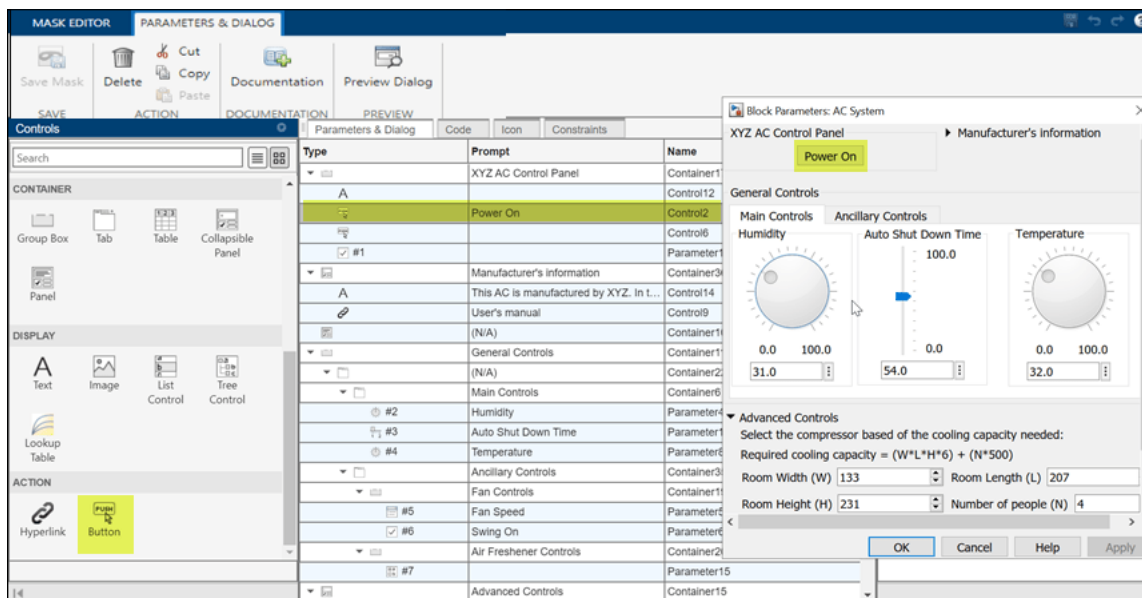


Similarly, you can add and configure various controls from the Mask Editor to build the mask dialog box.

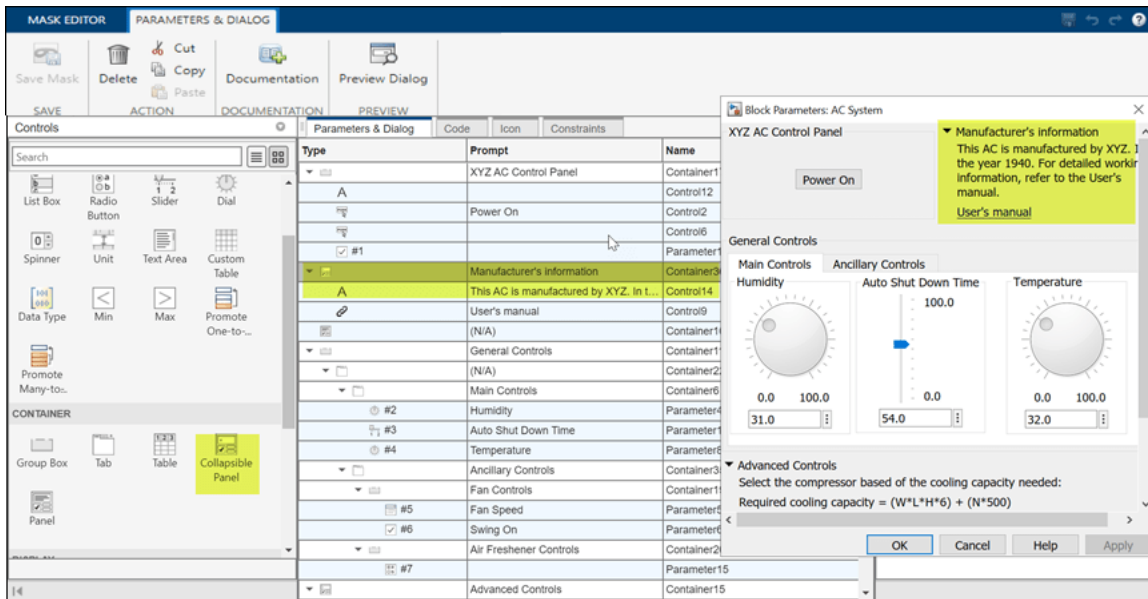
Observe the mask layout. Containers like group boxes, collapsible panels, and tabs group the controls together. Here, yellow represents **Group Box**, pink represents **Tab**, and green represents the **Collapsible Panel**.



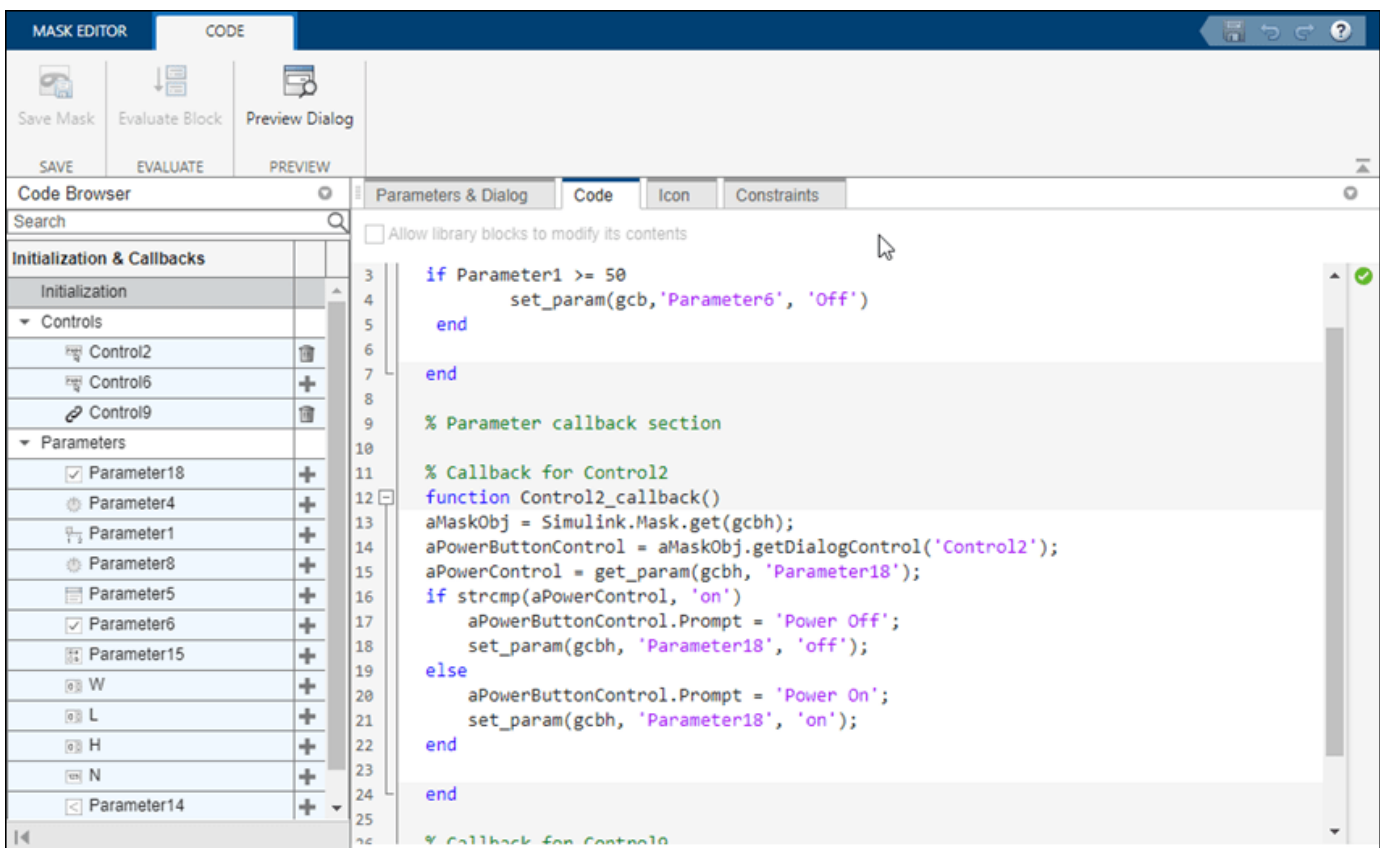
The **Button** controls type is used to create the **Power On button** on the mask dialog box. To manage the button placement, apply the **Horizontal Stretch** property. You can also add callback code to execute when the button is pressed. You can view a sample callback code for the **Button** controls type in the attached model.



The collapsible panel for **Manufacturer's information** contains **Text** and **Hyperlink** control types.

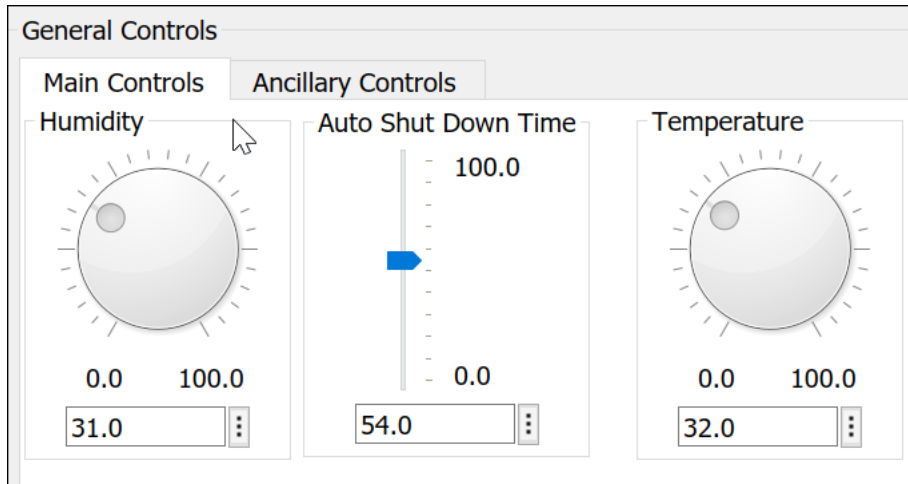


You can add MATLAB code as a callback for the hyperlink.

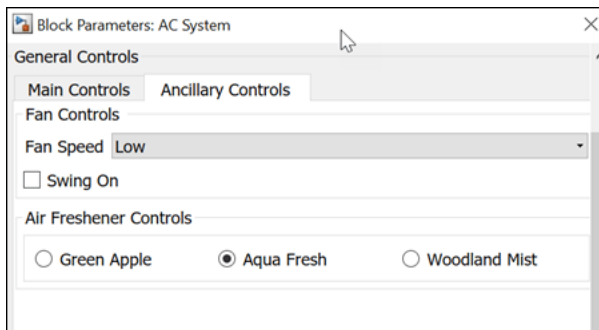


The **General Controls** section contains tabs to segregate and categorize information under **Main Controls** and **Ancillary Controls**. The **Main Controls** tab uses **Dials** and **Slider** to accept inputs

for air conditioner parameters. You can edit the property of dial and slider in the property editor section of Mask Editor to place them horizontally or vertically.



The **Ancillary Controls** use **Popup**, **Check Box**, and **Radio Buttons**.



The **Advanced Controls** section is a collapsible panel that contains spinbox, minimum, and maximum parameters to accept inputs.

▼ Advanced Controls
Select the compressor based of the cooling capacity needed:
Required cooling capacity = $(W*L*H*6) + (N*500)$

Room Width (W) Room Length (L)

Room Height (H) Number of people (N)

Minimum cooling capacity Maximum cooling capacity

See Also

More About

- “Mask Editor Overview” on page 18-2

Concurrent Execution Window

- “Concurrent Execution Window: Main Pane” on page 19-2
- “Data Transfer Pane” on page 19-5
- “CPU Pane” on page 19-8
- “Hardware Node Pane” on page 19-9
- “Periodic Pane” on page 19-11
- “Task Pane” on page 19-13
- “Interrupt Pane” on page 19-15
- “System Tasks Pane” on page 19-19
- “System Task Pane” on page 19-20
- “System Interrupt Pane” on page 19-22
- “Profile Report Pane” on page 19-24

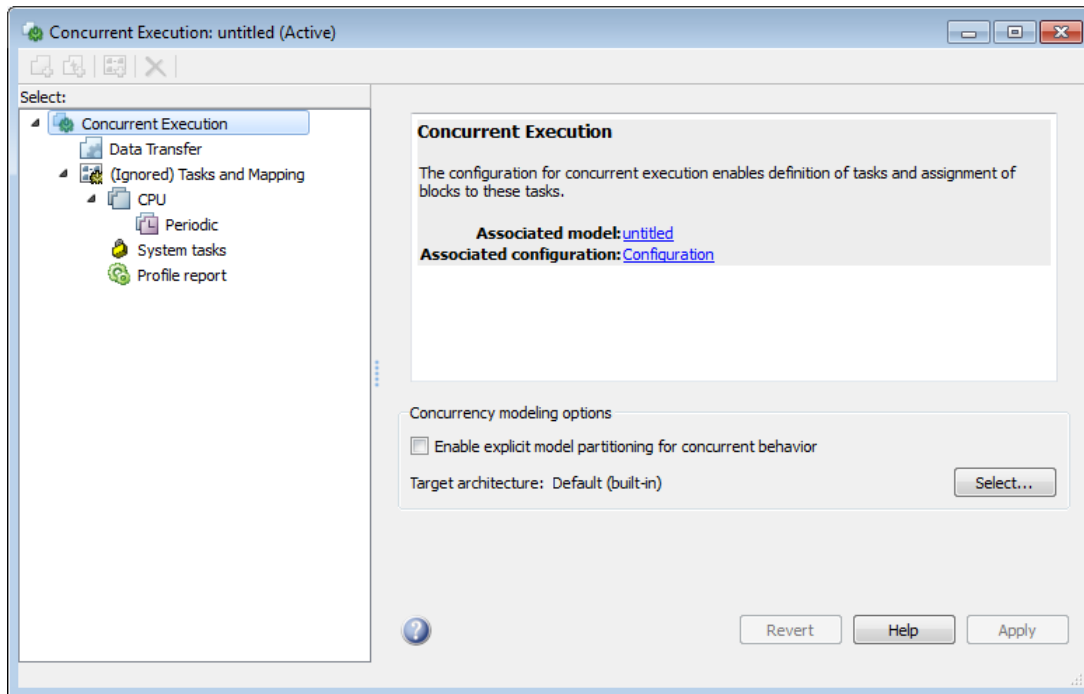
Concurrent Execution Window: Main Pane

In this section...

“Concurrent Execution Window Overview” on page 19-2

“Enable explicit model partitioning for concurrent behavior” on page 19-3

Concurrent Execution Window Overview



The Concurrent Execution window comprises the following panes:

- Concurrent Execution (root level)

Display general information for the model, including model name, configuration set name, and status of configuration set.

- Data Transfer on page 19-5

Configure data transfer methods between tasks.

- Tasks and Mapping

Map blocks to tasks.

- “CPU Pane” on page 19-8

Set up software nodes.

- Periodic on page 19-11

Name periodic tasks.

- Task on page 19-13

Define and configure a periodic task that the target operating system executes.

- Interrupt on page 19-15

Define aperiodic event handler that executes in response to hardware or software interrupts.

- System Task Pane on page 19-19

Display system tasks.

- System Task on page 19-20

Display periodic system tasks.

- System Interrupt on page 19-22

Display interrupt system tasks.

- “Profile Report Pane” on page 19-24

Generate and examine profile report for model.

Click items in the tree to select panes.

Configuration

This pane appears only if you select **Allow tasks to execute concurrently on target** in the **Solver** pane in the Configuration Parameters dialog box.

- 1 Open the Model Settings from the **Modeling** tab of the current model and select **Solver**.
- 2 Expand the **Solver details** section and select **Allow tasks to execute concurrently on target**.

This option is visible only if the **Solver Type** is Fixed Step and the **Periodic sample time constraint** is set to Unconstrained.

- 3 Click **Configure Tasks**.

The concurrent execution dialog box is displayed.

See Also

“Configure Your Model for Concurrent Execution”

Enable explicit model partitioning for concurrent behavior

Specify whether you want to manually map tasks (explicit mapping) or use the rate-based tasks.

Settings

Default: On

On

Enable manual mapping of tasks to blocks.

Off

Allow implicit rate-based tasks.

Command-Line Information

Parameter: ExplicitPartitioning

Value: 'on' | 'off'

Default: 'off'

See Also

“Configure Your Model for Concurrent Execution”

Dependencies

Selecting this check box:

- Allows custom task-to-block mappings. The node name changes to **Tasks and Mapping** label and the icon changes.
- Disables the **Automatically handle rate transition for data transfer** check box on the Data Transfer pane.

Clearing this check box

- Causes the software to ignore the task-to-block mappings. The node name changes to **(Ignored) Tasks and Mapping**.
- Enables the **Automatically handle rate transition for data transfer** check box on the Data Transfer pane.

Data Transfer Pane

In this section...

“Data Transfer Pane Overview” on page 19-5

“Periodic signals” on page 19-5

“Continuous signals” on page 19-6

“Extrapolation method” on page 19-6

“Automatically handle rate transition for data transfer” on page 19-7

Data Transfer Pane Overview

Data Transfer Options

Defaults

Periodic signals:

Continuous signals:

Extrapolation method:

Automatically handle rate transition for data transfer

Edit options to define data transfer between tasks.

See Also

“Configure Your Model for Concurrent Execution”

Periodic signals

Select the data transfer mode of synchronous signals.

Settings

Default: Ensure deterministic transfer (maximum delay)

Ensure deterministic transfer (maximum delay)

Ensure maximum capacity during data transfer.

Ensure data integrity only

Ensure maximum data integrity during data transfer.

Dependency

This parameter is enabled if the **Enable explicit task mapping to override implicit rate-based tasks** check box on the Concurrent Execution pane is selected.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Continuous signals

Select the data transfer mode of continuous signals.

Settings

Default: Ensure deterministic transfer (maximum delay)

Ensure deterministic transfer (maximum delay)

Ensure maximum capacity during data transfer.

Ensure data integrity only

Ensure maximum data integrity during data transfer.

Dependency

This parameter is enabled if the **Enable explicit task mapping to override implicit rate-based tasks** check box on the Concurrent Execution pane is cleared.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Extrapolation method

Select the extrapolation method of data transfer to configure continuous-to-continuous task transitions.

Settings

Default: None

None

Do not use any extrapolation method for task transitions.

Zero Order Hold

User zero order hold extrapolation method for task transitions.

Linear

User linear extrapolation method for task transitions.

Quadratic

User quadratic extrapolation method for task transitions.

Dependency

This parameter is enabled if the **Enable explicit task mapping to override implicit rate-based tasks** check box on the Concurrent Execution pane is selected.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Automatically handle rate transition for data transfer

Select the extrapolation method of data transfer to configure continuous-to-continuous task transitions.

Settings

Default: Off

On

Enable the software to handle rate transitions for data transfers automatically, without user intervention.

Off

Disable the software from handling rate transitions for data transfers automatically.

Dependencies

This parameter is enabled if the Concurrent Execution pane **Enable explicit task mapping to override implicit rate-based tasks** check box is cleared.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

CPU Pane

CPU Pane Overview

Configure software nodes.

See Also

“Configure Your Model for Concurrent Execution”

Name

Specify a unique name for software node.

Settings

Default: CPU

- Alternatively, enter a unique character vector to identify the software node. This value must be a valid MATLAB variable.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Hardware Node Pane

Hardware Node Pane Overview

Configure hardware nodes.

Name

Specify name of hardware node.

Settings

Default: FPGAN

- Alternatively, enter a unique character vector to identify the hardware node. This value must be a valid MATLAB variable.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Clock Frequency [MHz]

Specify clock frequency of hardware node.

Settings

Default: 33

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Color

Specify the color for the hardware node icon.

Settings

Default: Next color in basic color sequence

Tips

The hardware node icon appears in the tree.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Periodic Pane

In this section...

“Periodic Pane Overview” on page 19-11

“Name” on page 19-11

“Periodic Trigger” on page 19-11

“Color” on page 19-12

“Template” on page 19-12


Periodic Pane Overview

Periodic Trigger: Periodic

Properties

Name: Periodic

Period: 1

Color: 

Configure periodic (synchronous) tasks.

See Also

“Configure Your Model for Concurrent Execution”

Name

Specify a unique name for the periodic task trigger configuration.

Settings

Default: Periodic

- Alternatively, enter a unique character vector to identify the periodic task trigger configuration. This value must be a valid MATLAB variable.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Periodic Trigger

Specify the period of a periodic trigger

Settings

Default:

- Change `ERTDefaultEvent` to the actual trigger source event.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Color

Specify a color for the periodic trigger icon.

Settings

Default: Blue

- Click the color picker icon to select a color for the periodic trigger icon.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Template

Specify the XML-format custom architecture template file that code generation properties use for the task, periodic trigger or aperiodic triggers.

Settings

Default: None

The XML-format custom architecture template file defines these settings.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

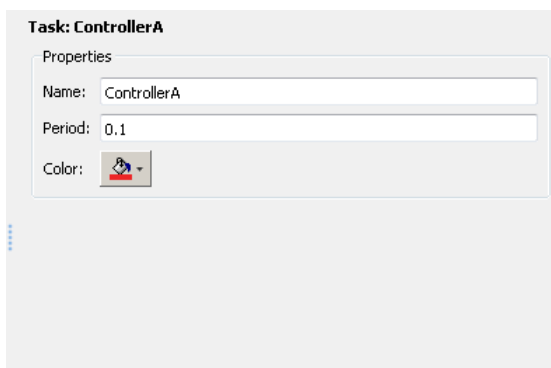
- “Define a Custom Architecture File”
- “Configure Your Model for Concurrent Execution”

Task Pane

In this section...

“Task Pane Overview” on page 19-13
“Name” on page 19-13
“Period” on page 19-14
“Color” on page 19-14

Task Pane Overview



Specify concurrent execution tasks. You can add tasks for periodic and interrupt-driven (aperiodic) tasks.

See Also

“Configure Your Model for Concurrent Execution”

Name

Specify a unique name for the task configuration.

Settings

Default: Task

- Alternatively, enter a unique character vector to identify the periodic task trigger configuration. This value must be a valid MATLAB variable.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Period

Specify the period for the task.

Settings

Default: 1

Minimum: 0

- Enter a positive real or ratio value.

Tip

You can parameterize this value by using MATLAB expression character vectors as values.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Color

Specify a color for the task icon.

Settings

Default: Blue

- Click the color picker icon to select a color for the task icon.

Tips

The task icon appears on the top left of the Model block. It indicates the task to which the Model block is assigned.

- As you add a task, the software automatically assigns a color to the task icon, up to six colors. When the current list of colors is exhausted, the software reassigns previously used colors to the new tasks, starting with the first color assigned.
- If you select a different color for an icon and then use the software to automatically assign colors, the software assigns a preselected color.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Interrupt Pane

In this section...

“Interrupt Pane Overview” on page 19-15

“Name” on page 19-15

“Color” on page 19-16

“Aperiodic trigger source” on page 19-16

“Signal number [2,SIGRTMAX-SIGRTMIN-1]” on page 19-17

“Event name” on page 19-17

Interrupt Pane Overview

Configure interrupt-driven (aperiodic) tasks.

See Also

“Configure Your Model for Concurrent Execution”

Name

Specify a unique name for the interrupt-driven task configuration.

Settings

Default: Interrupt

- Enter a unique character vector to identify the interrupt-driven task configuration. This value must be a valid MATLAB variable.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Color

Specify a color for the interrupt icon.

Settings

Default: Blue

- Click the color picker icon to select a color for the interrupt icon.

Tips

The interrupt icon appears on the top left of the Model block. It indicates the task to which the Model block is assigned.

- As you add an interrupt, the software automatically assigns a color to the interrupt icon, up to six colors. When the current list of colors is exhausted, the software reassigns previously used colors to the new interrupts, starting with the first color assigned.
- If you select a different color for an icon and then use the software to automatically assign colors, the software assigns a preselected color.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Aperiodic trigger source

Specify the trigger source for the interrupt-driven task.

Settings

Default: Posix Signal (Linux/VxWorks 6.x)

Posix Signal (Linux/VxWorks 6.x)

For Linux[®] or VxWorks[®] systems, select Posix Signal (Linux/VxWorks 6.x).

Event (Windows)

For Windows systems, select Event (Windows).

Dependencies

This parameter enables either **Signal number [2,SIGRTMAX-SIGRTMIN-1]** or **Event name**.

- Selecting Posix Signal (Linux/VxWorks 6.x) enables the following parameter:

Signal number [2,SIGRTMAX-SIGRTMIN-1]

- Selecting Event (Windows) enables the following parameter:

Event name

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Signal number [2,SIGRTMAX-SIGRTMIN-1]

Enter the POSIX® signal number as the trigger source.

Settings

Default: 2

Minimum: 2

Maximum: SIGRTMAX-SIGRTMIN-1

- Enter the POSIX signal number as the trigger source.

Dependencies

Aperiodic trigger source > Posix signal (Linux/VxWorks 6.x) enables this parameter.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Event name

Enter the name of the event as the trigger source.

Settings

Default: ERTDefaultEvent

- Change ERTDefaultEvent to the actual trigger source event.

Dependencies

Aperiodic trigger source > Event (Windows) enables this parameter.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

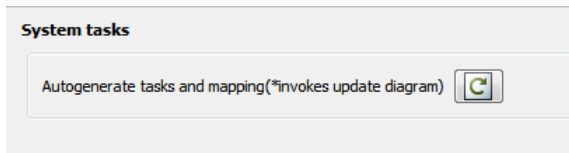
See Also

“Configure Your Model for Concurrent Execution”

System Tasks Pane

System Tasks Pane Overview

Display system tasks.



See Also

“Configure Your Model for Concurrent Execution”

System Task Pane

In this section...

“System Task Pane Overview” on page 19-20

“Name” on page 19-20

“Period” on page 19-20

“Color” on page 19-21

System Task Pane Overview

Display periodic system tasks.



The screenshot shows a configuration window titled "Task: Discrete1". Under the "Properties" section, there are three fields: "Name" with the value "Discrete1", "Period" with the value "0.1", and "Color" with a color selection icon.

See Also

“Configure Your Model for Concurrent Execution”

Name

Specify a default name for the periodic system task configuration.

Settings

Default: Discrete*N*

Tip

To change the name, period, or color of this task, right-click the task node and select **Convert to editable periodic task**.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Period

Specify the period for the task.

Settings

Default: 1

Minimum: 0

- Enter a positive real or ratio value.

Tip

- To change the name, period, or color of this task, right-click the task node and select **Convert to editable periodic task**.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Color

Specify the outline color for the task icon.

Settings

Default: Blue

Tips

The task icon appears on the top left of the Model block. It indicates the task the Model block is assigned to.

- To change the name, period, or color of this task, right-click the task node and select **Convert to editable periodic task**.

See Also

“Configure Your Model for Concurrent Execution”

System Interrupt Pane

In this section...

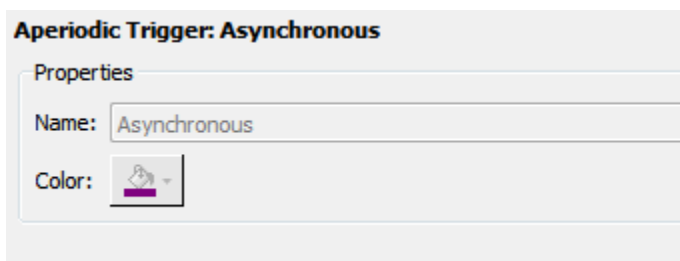
“System Interrupt Pane Overview” on page 19-22

“Name” on page 19-22

“Color” on page 19-22

System Interrupt Pane Overview

Display interrupt system tasks.



See Also

“Configure Your Model for Concurrent Execution”

Name

Specify a default name for the interrupt system task.

Settings

Default: Asynchronous

Tip

To change the name or color of this task, right-click the task node and select **Convert to editable aperiodic trigger**.

Command-Line Information

See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Color

Specify the outline color for the task icon.

Tips

The task icon appears on the top left of the Model block. It indicates the task the Model block is assigned to.

- To change the name or color of this task, right-click the task node and select **Convert to editable aperiodic task**.

See Also

“Configure Your Model for Concurrent Execution”

Profile Report Pane

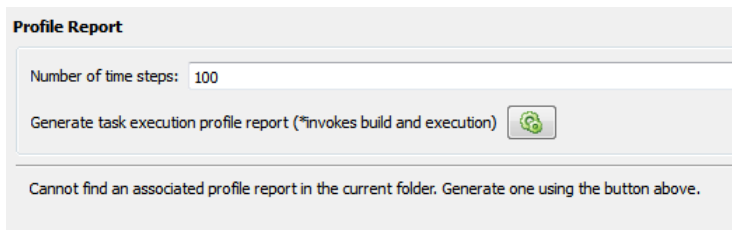
In this section...

“Profile Report Pane Overview” on page 19-24

“Number of time steps” on page 19-24

Profile Report Pane Overview

Generate and examine profile report for model.



The screenshot shows a software interface for generating a profile report. It features a title bar labeled "Profile Report". Below the title bar, there is a text input field labeled "Number of time steps:" with the value "100" entered. To the right of the input field is a button with a green circular icon containing a gear. Below the input field and button, there is a message that reads: "Cannot find an associated profile report in the current folder. Generate one using the button above."

See Also

“Configure Your Model for Concurrent Execution”

Number of time steps

Specify number of time steps to generate profile report.

Settings

Default: 100

- Enter the number of time steps to collect data.

Command-Line Information

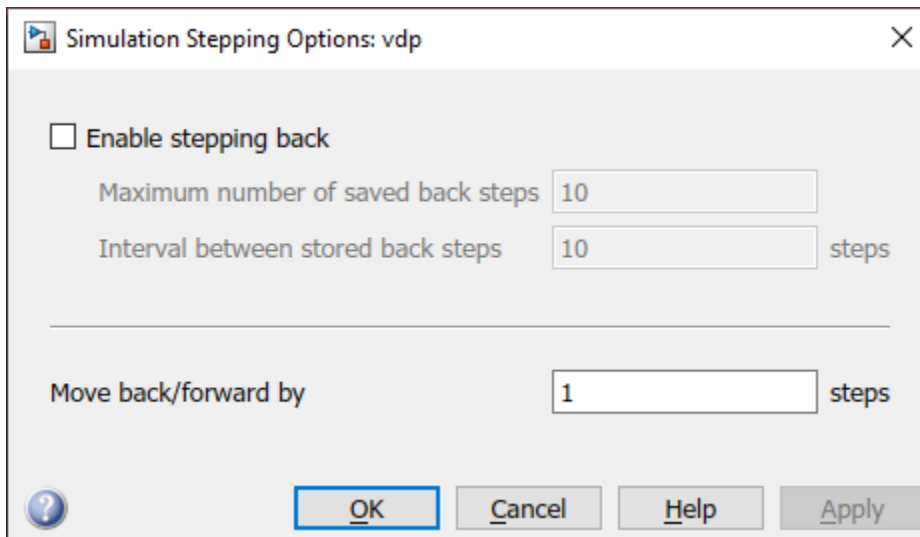
See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Simulink Simulation Stepper

Simulation Stepping Options



In this section...

“Simulation Stepping Options Overview” on page 20-2

“Enable stepping back” on page 20-3

“Maximum number of saved back steps” on page 20-3

“Interval between stored back steps” on page 20-4

“Move back/forward by” on page 20-4

Simulation Stepping Options Overview


Use the Simulation Stepping Options dialog box to manually step through a simulation.

Configuration

This pane appears when you select **Step Back > Configure Simulation Stepping** in the **Simulation** tab.

- 1 To step backwards through a simulation, select **Enable stepping back** and specify the total number and frequency of snapshots.
- 2 Specify the increment of steps by which the simulation steps either forward or backwards.

Tips

- To start the Simulation Stepping Options dialog box from the Simulink toolstrip, click .
- To pause simulation at a particular time, select **Pause simulation when time reaches** check box and enter the pause time.
- You can change the value while the simulation is running or paused.

See Also

- “How Simulation Stepper Helps With Model Analysis”

Enable stepping back

Enable stepping back.

Settings

Default: Off

On

Enable stepping back.

Off

Disable stepping back.

Tip

Simulation stepping (forward and back) is available only for Normal and Accelerator modes.

Dependencies

This parameter enables the **Maximum number of saved back steps** and **Interval between stored back steps** parameters.

See Also

“How Simulation Stepper Helps With Model Analysis”

Maximum number of saved back steps

Enter the maximum number of snapshots that the software can capture. A snapshot at a particular simulation time captures all the information required to continue a simulation from that point.

Settings

Default: 10

Minimum: 0

Dependencies

Enable stepping back enables this parameter and the **Interval between stored back steps** parameter.

See Also

- “How Simulation Stepper Helps With Model Analysis”
- “Simulation Snapshots”

Interval between stored back steps

Enter the number of major time steps to take between capturing simulation snapshots.

Settings

Default: 10

Minimum: 1

- “How Simulation Stepper Helps With Model Analysis”
- “Simulation Snapshots”

Tip

The number of steps to skip between snapshots. This parameter enables you to save snapshots of simulation state for stepping backward at periodic intervals, such as every three steps forward. This interval is independent of the number of steps taken in either the forward or backward direction. Because taking simulation snapshots affects simulation speed, saving snapshots less often can improve simulation speed.

Dependencies

Enable stepping back enables this parameter and the **Maximum number of saved back steps** parameter.

See Also

- “How Simulation Stepper Helps With Model Analysis”
- “Simulation Snapshots”

Move back/forward by

Enter the number of major time steps for a single call to step forward or back.

Settings

Default: 1

Minimum: 1

Tip

The maximum number of steps, or snapshots, to capture while simulating forward. The greater the number, the more memory the simulation occupies and the longer the simulation takes to run.

See Also

- “How Simulation Stepper Helps With Model Analysis”
- “Simulation Snapshots”

Variant Manager for Simulink

Variant Manager for Simulink

Note This functionality requires Variant Manager for Simulink.

In Model-Based Design for system development, you might have to use multiple design alternatives for components in the system. For example, in a model that represents an automobile, you might have several exhaust temperature sensors supplied by different vendors. Throughout the development life cycle, from requirements to deployment, you may need to switch between these design choices.

You might also model systems that represent a product line such as cars, aeroplanes, and communication systems. Product lines are created by adding variation points to a system. For example, vehicles in a product line of passenger cars can have multiple variation points such as fuel consumption, motor type, or engine size.

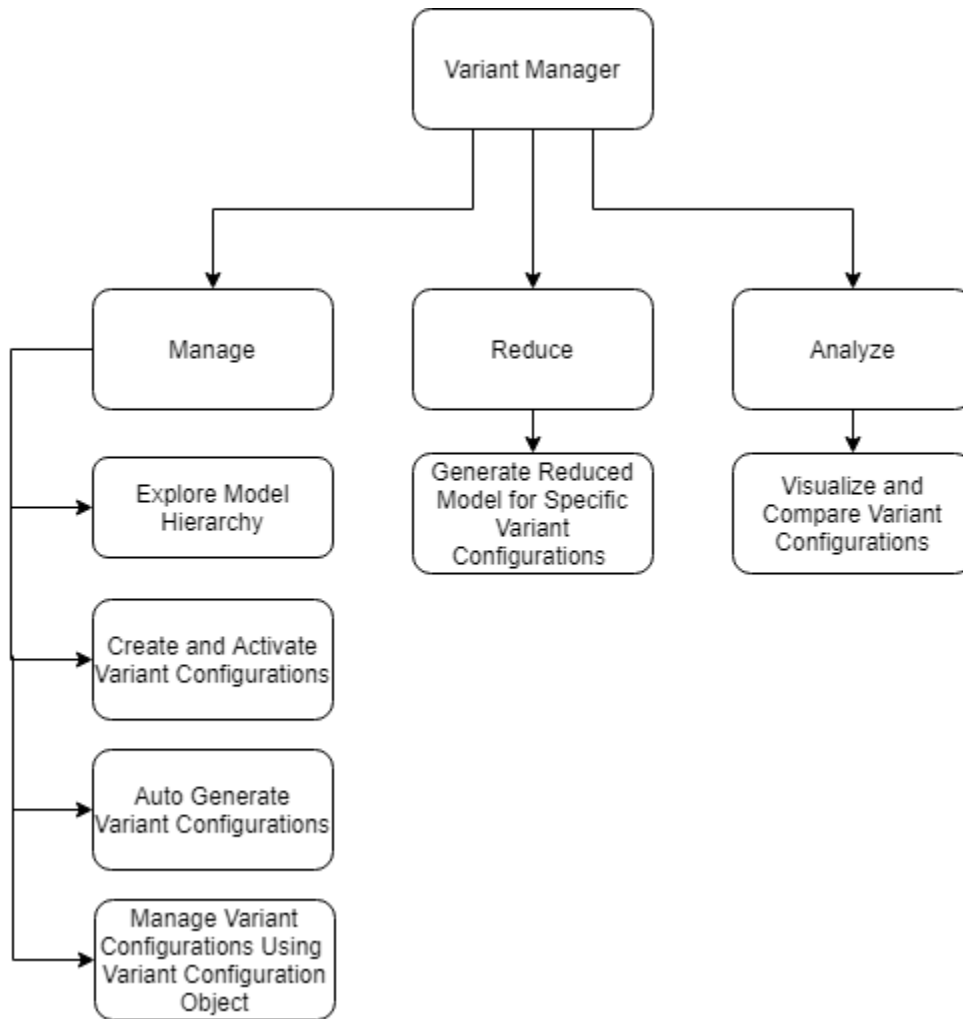
Instead of designing multiple models to represent all possible variants, you can use variant elements in Simulink to represent all the variations in a single model. For an introduction to variants in Simulink see, “What Are Variants and When to Use Them”.

Variant Manager

Variant Manager is a tool that allows you to visualize the model hierarchy and centrally manage the usage of variant elements such as variant blocks and variant transitions across the hierarchy.

The tool is available as a support package named Variant Manager for Simulink with these main capabilities:

- Variant Manager — Visualize the model hierarchy, manage the usage of variant elements across the hierarchy, and create and manage variant configurations.
- Variant Reducer — Generate a reduced model that contains only selected variant configurations.
- Variant Analyzer — Compare and contrast variant configurations to identify errors or inconsistencies.



Install Variant Manager for Simulink

To install the support package, use one of these methods:

- Open Variant Manager:
 - 1 In Simulink, on the **Modeling** tab, open the **Design** section and click **Variant Manager**. You can also use any of the alternate methods to open Variant Manager.
 - 2 In the Install Variant Manager for Simulink dialog box, click **Add** to install the Variant Manager for Simulink add-on.
- Use Add-On Explorer:
 - 1 In MATLAB, on the **Home** tab, in the **Environment** section, click **Add-Ons** and then select **Get Add-ons**.
 - 2 In the Add-On Explorer, find and click the Variant Manager for Simulink support package, and click **Install**.

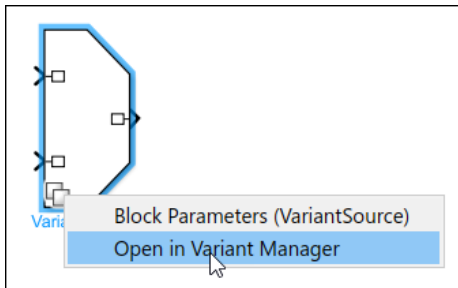
When you execute any Variant Manager related APIs from the MATLAB Command-Line, the APIs return an error with a hyperlink to launch the installer.

For information on the behavior changes in the support package, see “Compatibility Considerations When Using Variant Manager for Simulink Support Package”.

Open Variant Manager

Use any of these methods to open Variant Manager:

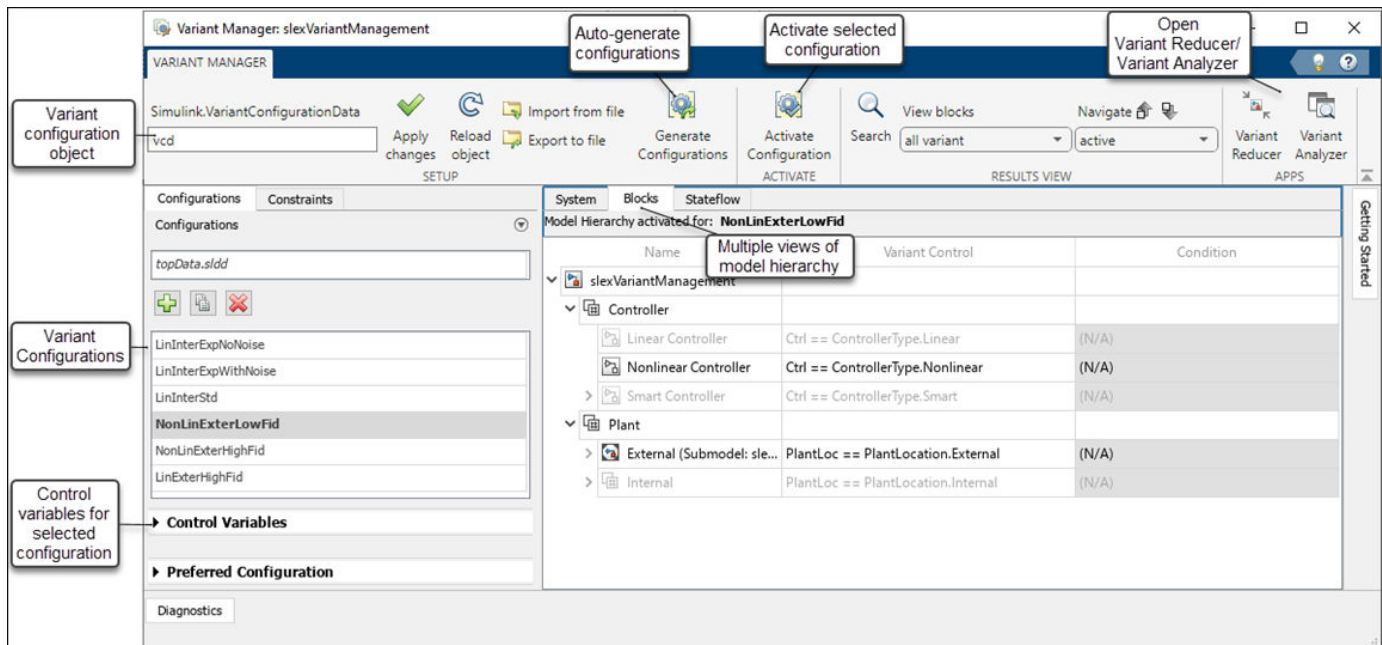
- Right-click the variant badge icon on any variant block and select **Open in Variant Manager**.




- On the **Modeling** tab, open the **Design** section and click **Variant Manager**.
- Right-click a variant block and select **Variant > Open in Variant Manager**.
- Select a variant block, for example, a Variant Subsystem block, and then in the **Variant Subsystem** tab of the Simulink toolstrip select **Variant Manager**.
- Click **Open block in Variant Manager** available on the variant block’s **Block Parameter** dialog box.

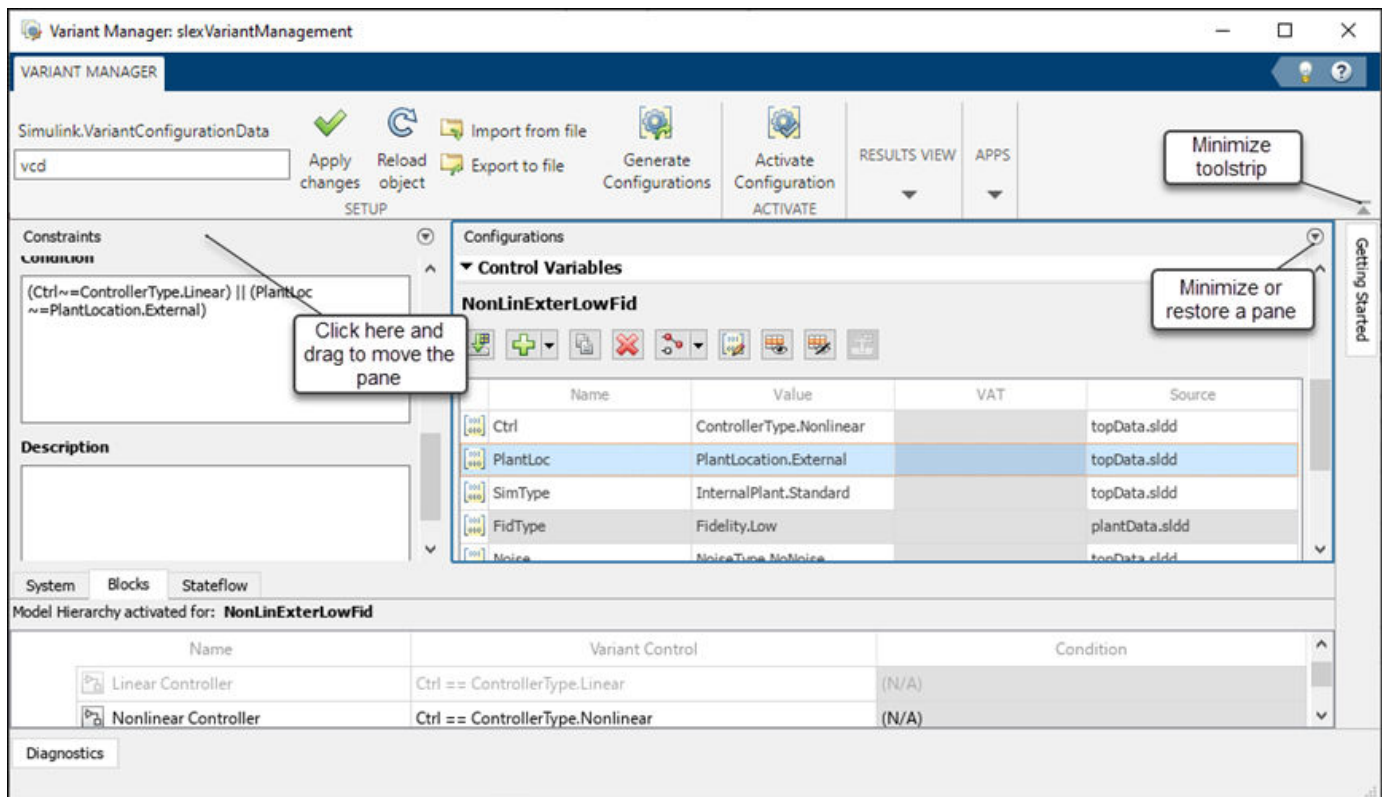
Explore Variant Manager Window

This image shows the default view of the Variant Manager window.



- You can change the layout of the window according to your preference. To move a pane, click at the top of the pane and drag.
- You can minimize unused panes. When you want to work on a minimized pane again, restore it to stop it from collapsing automatically.
- The **Getting Started** pane appears on the right side of the window by default and provides a quick overview of the common workflows.
- You can use the Help button  in the top right corner of the Variant Manager window to access the documentation.
- The **Diagnostics** pane appears at the bottom of the window by default and displays messages, errors, and warnings related to the actions performed from the Variant Manager.

This image shows a custom layout of the window.



Manage Variant Elements

Visualize Model Hierarchy

The model hierarchy table presents a tree view of the model where each node represents a block or a referenced component. You can expand the nodes and navigate the hierarchy.

To get different *views* of the model hierarchy, use these tabs:

- **System** — Displays all blocks
- **Blocks** — Displays variant blocks

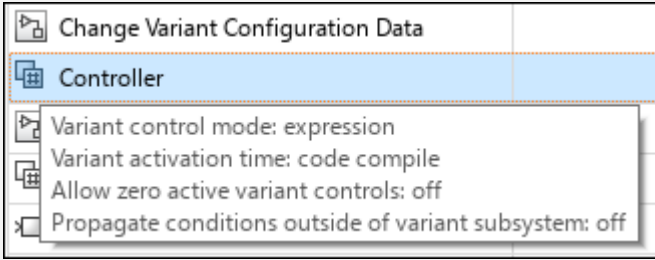
- **Stateflow** — Displays variant transitions used in Stateflow charts
- **Component Configurations** — Displays available variant configurations for referenced components

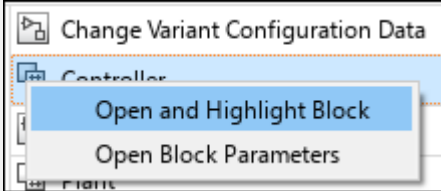

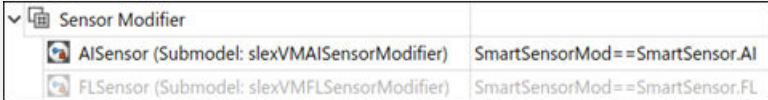
Note When you open the Variant Manager for a top-level model, variant elements inside referenced components such as Model blocks and libraries are not loaded. The referenced components are loaded and activated only when you explicitly activate the model or expand them in the model hierarchy.

The **Component Configurations** tab is not shown by default. To open the tab, click the **Show Component Configurations** button in the **Control Variables** section of a selected variant configuration.

Interact with Model Hierarchy

You can perform these actions from the model hierarchy.

Action	Model Hierarchy Interaction
View and edit the variant condition expression for each variant choice	<p>The Variant Control column in the table is similar to the Variant Control field in the block parameter dialog box of variant blocks. You can edit this field for variant elements in the hierarchy.</p> <p>For variant elements, the field shows a list of context-specific keywords that are allowed as the variant control for a variant block. For example, for a Variant Subsystem with Variant control mode set to expression, the list shows default in addition to the variant control expression. For sim-codegen switching mode, the list shows sim or codegen values. For a variant Simulink Function block, the list shows inherit.</p>
Search	Use the Search button in the toolbar to search for any element in the hierarchy.
See block parameter values	<p>Point to any variant block to see a tooltip with the block parameter values.</p> 

Action	Model Hierarchy Interaction
See block specific context menu	<p>Right-click a block to find these options:</p> <ul style="list-style-type: none"> • Open and Highlight Block: Opens the selected block in the model and highlights it. This provides traceability to the model. • Open Model: Opens a referenced model. • Open Block Parameters: Opens the block parameter dialog box for the selected block. You can modify the parameter values. • Open Parent Block Parameters: Opens the block parameter dialog box for the parent block of the selected block. You can modify the parameter values. • Set as Label Mode Active Choice: Sets the selected choice of Variant Subsystem, Variant Sink, or Variant Source blocks as active choice. This option is available only if the Variant Control mode block parameter is set to <code>label</code>. 
Filter variant blocks by their Variant control mode	Use the View blocks list in the toolbar.
Navigate the model hierarchy based on filters	<p>Use the Navigate buttons in the toolbar to step through the model hierarchy based on these filters:</p> <ul style="list-style-type: none"> • variable usage — Selects the previous/next rows in the hierarchy that uses the selected variant control variable. <p>To enable the Navigate buttons, select the required variant configuration from the Configurations tab. In the Control Variables section for that configuration, select the control variable from the table. Click the Show usage of selected control variables  button.</p> <ul style="list-style-type: none"> • active — Selects the previous/next rows in the hierarchy that has active variant choices. • invalid — Selects the previous/next rows in the hierarchy that has invalid variant choices.
Identify active variant choices	<p>The inactive choice appears greyed out.</p> 
Identify rows with errors	They are highlighted in red.

Action	Model Hierarchy Interaction
Identify type of block by the block icon	For the list of block icons, see “Model Hierarchy Table” on page 21-11.

Create and Activate Variant Configurations

A variant configuration represents a combination of variant choices across the model hierarchy. From Variant Manager, you can:

- Create a named variant configuration.
- Create a temporary configuration in the global workspace.
- Add, import, export, and edit control variables in a configuration.
- Select referenced model configurations (component configurations).
- Add constraints applicable for all configurations.
- Validate and activate a configuration on a model.
- Set a preferred variant configuration for a model.

For an overview of variant configurations, see “Variant Configurations”.

For detailed steps to create a variant configuration, see “Create and Activate Variant Configurations”.

Auto-generate Variant Configurations

Creating all possible variant configurations for a model manually can be time consuming. You must activate them individually to check if they are valid and if they satisfy necessary constraints. Instead, you can automatically generate variant configurations for a model using Variant Manager, which enables you to:

- Consider all possible combinations of variant control variables while creating configurations.
- Specify the value range that must be considered for each control variable, to generate only the required subset of configurations.
- Specify preconditions to restrict the configurations to generate, and optionally export the preconditions as constraints.
- Automatically validate generated configurations to identify invalid cases.
- Generate valid, valid and unique, or all configurations.
- Export the configurations to a variant configuration data object. You can export valid configurations for which the model compiles successfully or all configurations including invalid ones.

For detailed steps to generate variant configurations, see “Generate Variant Configurations Automatically”.

Manage Variant Configurations

You can use a variant configuration data object of type `Simulink.VariantConfigurationData` to manage and reuse variant configurations for a model. The object stores all the variant configurations and constraints created for a model. If the model is not associated with a variant configuration data object, Variant Manager helps you to setup a new variant configuration data object.

From the **Manage** tab in Variant Manager, you can:

- Specify a name for the variant configuration data object for the model.
- Apply the changes made to the variant configuration data object from Variant Manager to the global workspace (base workspace or data dictionary) used by the model.
- Export the variant configuration data object to a MAT-file or MATLAB script file.
- Import a variant configuration data object from a MAT-file or MATLAB script file into Variant Manager.
- Reload the object from the global workspace used by the model. This allows you to revert the changes that are not yet exported to the global workspace.

When you export the variant control variables in a variant configuration to the global workspace or when you activate a variant configuration, the corresponding variant control variables are pushed to the global workspace. Reloading the variant configuration object from Variant Manager does not revert these changes.

The `Simulink.VariantConfigurationData` class has methods that enable you to add or remove variant configurations, constraints, and control variables.

For an example that shows how to save and reload a variant configuration data object from Variant Manager, see “Save and Reuse Variant Configurations Using Variant Configuration Data Object”.

Reduce a Variant Model

You can use Variant Reducer to generate simplified, stand alone models that contains only the specified set of variant configurations from the parent model. For example, to generate a model that maps to a specific product from a product line (single configuration reduction), or that which corresponds to a product line from a product line family (multi-configuration reduction).

To open Variant Reducer, in the Variant Manager toolstrip, in the **APPS** section, click **Variant Reducer**.

Variant Reducer performs these high-level operations during the reduction process:

- Removes inactive model components based on the variant configurations that you choose to retain in the reduced model.
- Removes or modifies model components such as blocks, variant parameter objects, masks, model references, subsystem references, libraries, dependent files, and variables in the input model to create the reduced model.
- Packages the reduced model and related artifacts into a user-specified output folder.
- Generates a detailed summary of the reduction process that helps you to analyze these changes.

See, “Reduce Variant Models Using Variant Reducer”.

Analyze Variant Configurations

You can use Variant Analyzer to analyze and compare the variant configurations for a model. To open Variant Analyzer, in the Variant Manager toolstrip, in the **APPS** section, click **Variant Analyzer**.




You can analyze the named variant configurations created for a model or perform an analysis after setting values for the variant control variables. The variant analysis report generated by the app helps you to:

- Compare different variant configurations for a model to understand the common and differing model elements used between them.
- Check if all variant choices have been activated at least once and whether the model is covered completely for simulation and code generation.
- Verify if the active, implemented model is different between different variant configurations.
- Find the dependent model artifacts such as referenced models and libraries used by a particular variant configuration.

See, “Analyze Variant Configurations in Models Containing Variant Blocks”.





Icons in Variant Manager

Configurations




Button	Description
	Add a variant configuration.
	Delete a variant configuration.
	Copy a variant configuration.







Control Variables

This table lists the icons used to represent different types of control variables.


Control Variable Icon	Type of Control Variable
	Normal MATLAB variable
	Simulink.Parameter or AUTOSAR.Parameter
	Simulink.VariantControl with value as normal MATLAB variable
	Simulink.VariantControl with value as Simulink.Parameter

Control Variables Section










Button	Description
	Import control variables from the entire model reference hierarchy Note Control variables from blocks in Label mode are not imported, as they are not variant control variables.
	Add a control variable.
	Create a copy of a control variable.





















Button	Description
	Delete a control variable.
	Change the data type of a control variable.
	Edit Simulink.Parameter or AUTOSAR.Parameter control variables. This option gets activated when the selected control variable is one of these types. Note To specify Simulink.Parameter control variable as an expression, set the Value property of the parameter object by using an equals sign (=) followed by a mathematical expression. For example, = A + B.
	Show usage of selected control variables.
	Hide usage of selected control variables.
	Export control variables to global workspace.















Component Configurations Tab

Icon	Purpose
	This icon next to a referenced model in the Component Configurations view indicates that the referenced component has its own predefined variant configurations.

Model Hierarchy Table

Icon	Element
	Model Block
	Inline Variant Blocks (Variant Source and Variant Sink)
	Variant Subsystem block
	Subsystem block
	Variant Model block
	Subsystem Reference block
	Simulink Function block
	Trigger port block
	Stateflow chart block

Icon	Element
	Variant Sink output port
	Variant Source input port
	Variant Subsystem block with Propagate conditions outside of variant subsystem option selected.
	Variant Subsystem block with Variant activation time set to update diagram.
	Variant Subsystem block with Variant activation time set to update diagram analyze all choices.
	Variant Subsystem block with Variant activation time set to code compile.
	Variant Subsystem block with Variant activation time set to startup.
	Variant Subsystem block with Allow zero active variant controls selected and Variant activation time set to update diagram.
	Variant Subsystem block with Allow zero active variant controls selected and Variant activation time set to update diagram analyze all choices.
	Variant Subsystem block with Allow zero active variant controls selected and Variant activation time set to code compile.
	Variant Subsystem block with Variant control mode set to label and an active variant choice selected from the Label mode active choice option.
	Variant Subsystem block with Propagate conditions outside of variant subsystem and Variant activation time set to update diagram.
	Variant Subsystem block with Propagate conditions outside of variant subsystem and Variant activation time set to update diagram analyze all choices.
	Variant Subsystem block with Propagate conditions outside of variant subsystem and Variant activation time set to code compile.
	Variant Subsystem block with Propagate conditions outside of variant subsystem and Variant activation time set to startup.
	Variant Subsystem block with Propagate conditions outside of variant subsystem option selected. Also, Variant control mode is set to label and an active variant choice is selected from the Label mode active choice option.
	Inline Variants Block (Variant Source and Variant Sink) with Allow zero active variant control option selected.
	Inline Variants Block (Variant Source and Variant Sink) with Variant control mode set to label and an active variant choice selected from the Label mode active choice option.
	Inline Variants Block (Variant Source and Variant Sink) with Variant activation time set to update diagram.
	Inline Variants Block (Variant Source and Variant Sink) with Variant activation time set to update diagram analyze all choices.

Icon	Element
	Inline Variants Block (Variant Source and Variant Sink) with Variant activation time set to code compile.
	Inline Variants Block (Variant Source and Variant Sink) with Variant activation time set to startup.
	Inline Variants Block (Variant Source and Variant Sink) with Allow zero active variant control and Variant activation time set to update diagram.
	Inline Variants Block (Variant Source and Variant Sink) with Allow zero active variant control and Variant activation time set to update diagram analyze all choices.
	Inline Variants Block (Variant Source and Variant Sink) with Allow zero active variant control and Variant activation time set to code compile.
	Inline Variants Block (Variant Source and Variant Sink) with Allow zero active variant control and Variant activation time set to startup.
	Initialize Function block
	Event Listener block of Initialize Function block
	Reset Function block
	Event Listener block of Reset Function block
	Terminate Function block
	Event Listener block of Terminate Function block
	Stateflow chart with Generate preprocessor conditionals option selected.
	Stateflow transition with Treat as Variant Transition option selected.

Note For variant blocks with **Variant activation time** set to `Inherit` from `Simulink.VariantControl`, the variant manager activation process updates the variant badge for the block in the model hierarchy to indicate the activation time that is computed from the corresponding `Simulink.VariantControl` variables.

Access the Variant Manager Functionality Programmatically

The `Simulink.VariantManager` class provides a set of methods to access Variant Manager functionality from the MATLAB Command Line.

The `Simulink.VariantConfigurationData` class has methods to add or remove variant configurations, constraints, and control variables programmatically.

The `Simulink.VariantConfigurationAnalysis` class has methods to analyze or compare variant configurations programmatically.

Limitations

- Variant Manager reports errors and warnings related to variant elements only.
- Variant Manager does not support variant controls defined in `InitFcn` callbacks and mask workspaces.
- The model hierarchy table does not show protected referenced models.
- Variant Manager constraints are not validated post compilation, for example, at startup variant activation time.
- Variant configurations can be created only for variant blocks that have **Variant control mode** set to `expression`.
- Variant Manager does not support workflows such as activation, viewing, or importing of control variables from variations inside a protected model. These variations are present when variant control variables of variant blocks with startup activation time are specified as `TunableParameters` while creating the protected model.

See Also

`Simulink.VariantConfigurationData` | `Simulink.VariantManager` |
`Simulink.VariantConfigurationAnalysis`

More About

- “What Are Variants and When to Use Them”
- “V-Model for System Development with Simulink Variants”
- “Compatibility Considerations When Using Variant Manager for Simulink Support Package”
- “Variant Configurations”
- “Create and Activate Variant Configurations”
- “Generate Variant Configurations Automatically”

Math and Data Types

Math and Data Types Pane

The **Math and Data Types** pane includes parameters for specifying data types and net slope calculations.

Parameter	Description
“Simulation behavior for denormal numbers” on page 22-3	Specify the desired behavior for denormal results from arithmetic operations. Requires a Fixed-Point Designer license.
“Use algorithms optimized for row-major array layout” on page 22-16	Enable algorithms for row-major format code generation and corresponding row-major algorithms for simulation.
“Default for underspecified data type” on page 22-4	Specify the default data type to use for inherited data types if Simulink software could not infer the data type of a signal during data type propagation.
“Use division for fixed-point net slope computation” on page 22-6	The Fixed-Point Designer software performs net slope computation using division to handle net slopes when simplicity and accuracy conditions are met.
“Gain parameters inherit a built-in integer type that is lossless” on page 22-8	The data type of the gain parameter is a built-in integer when certain conditions are met.
“Use floating-point multiplication to handle net slope corrections” on page 22-10	The Fixed-Point Designer software uses floating-point multiplication to perform net slope correction for floating-point to fixed-point casts.
“Inherit floating-point output type smaller than single precision” on page 22-12	Specify the desired inherited output data type behavior when block inputs are floating-point data types smaller than single precision.

These configuration parameters are in the **Advanced parameters** section.

Parameter	Description
“Application lifespan (days)” on page 22-14	Specify how long (in days) an application that contains blocks depending on elapsed or absolute time should be able to execute before timer overflow.
“Implement logic signals as Boolean data (vs. double)” on page 2-84	Controls the output data type of blocks that generate logic signals.

Simulation behavior for denormal numbers

Description

Specify the desired behavior for denormal results from arithmetic operations. Denormal numbers are any non-zero numbers whose magnitude is smaller than the smallest normalized floating-point number (see `realmin`). Some hardware targets flush denormal results from arithmetic operations to zero. You can emulate this behavior by setting the parameter to `Flush To Zero (FTZ)`.

Settings

Default: Gradual Underflow

Gradual Underflow

Denormal numbers are used in the results from arithmetic operations. This setting is supported by all simulation modes.

Flush To Zero (FTZ)

Emulates flush-to-zero behavior for denormal numbers. This setting flushes denormal results from arithmetic operations to zero. This setting is not supported during normal mode simulation.

Tips

In model reference systems, you can simulate a top-level model using gradual underflow with any simulation mode. Models referenced by the top-level model can simulate the flush-to-zero behavior only if the instance of the referenced model uses an accelerated simulation mode and has the **Simulation behavior for denormal numbers** parameter set to `Flush to zero (FTZ)`.

Command-Line Information

Parameter: `DenormalBehavior`

Value: `'GradualUnderflow' | 'FlushToZero'`

Default: `'GradualUnderflow'`

Dependency

This parameter requires a Fixed-Point Designer license.

See Also

More About

- “Exceptional Arithmetic” (Fixed-Point Designer)

Default for underspecified data type

Description

Specify the default data type to use for inherited data types if Simulink software could not infer the data type of a signal during data type propagation.

Category: Math and Data Types

Settings

Default: double

double

Sets the data type for underspecified data types during data type propagation to double. Simulink uses `double` as the data type for inherited data types.

single

Sets the data type for underspecified data types during data type propagation to single. Simulink uses `single` as the data type for inherited data types.

Tips

- This setting affects both simulation and code generation.
- For embedded designs that target single-precision processors, set this parameter to `single` to avoid the introduction of double data types.
- Use the Model Advisor Identify questionable operations for strict single-precision design check to identify the double-precision usage in your model.

Command-Line Information

Parameter: DefaultUnderspecifiedDataType

Value: 'double' | 'single'

Default: 'double'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	single (when target hardware supports efficient single computations) double (otherwise)
Safety precaution	No impact

See Also

Related Examples

- “Underspecified data types” on page 6-10
- “Validate a Single-Precision Model”
- “Model Configuration Parameters: Code Generation Optimization” (Simulink Coder)

Use division for fixed-point net slope computation

Description

The Fixed-Point Designer software performs net scaling computation using division to handle net scaling when simplicity and accuracy conditions are met.

Category: Math and Data Types

Settings

Default: Off

Off

Performs net scaling computation using integer multiplication followed by shifts.

On

Performs net scaling computation using a rational approximation of the net scaling. This results in integer division, multiplication, and addition when simplicity and accuracy conditions are met.

Use division for reciprocals of integers only

Performs net slope computation using division when the net slope can be represented by the reciprocal of an integer and simplicity and accuracy conditions are met.

Tips

- This optimization affects both simulation and code generation.
- When a change of fixed-point slope is not a power of two, net scaling computation is necessary. Normally, net scaling computation uses an integer multiplication followed by shifts. Enabling this optimization replaces the multiplication and shifts with an integer division, multiplication, and addition under certain simplicity and accuracy conditions.
- Performing net scaling computation using division is not always more efficient than using multiplication followed by shifts. Ensure that the target hardware supports efficient division.
- To ensure that this optimization occurs:
 - Set the word length of the block so that the software can perform division using the long data type of the target hardware. That setting avoids use of multiword operations.
 - Set the **Signed integer division rounds to** configuration parameter on the Hardware Implementation pane to `Zero` or `Floor`. The optimization does not occur if you set this parameter to `Undefined`.
 - Set the **Integer rounding mode** parameter of the block to `Simplest` or to the value of the **Signed integer division rounds to** configuration parameter setting on the Hardware Implementation pane.
- The following table summarizes how this parameter effects different fixed-point operations.

Operation	Use division for fixed-point net slope computation On	Use division for fixed-point net slope computation Off
Multiplication	Fixed-point multiplication operations with non-power-of-2 slopes and/or non-zero bias have improved representation.	Fixed-point multiplication operations follow legacy behavior.
Division	Fixed-point division operations with non-power-of-2 slopes and/or non-zero bias have improved representation.	
Cast	Fixed-point cast operations with non-power-of-2 slopes and/or non-zero bias have improved representation.	Fixed-point cast operations follow legacy behavior.

Dependency

This parameter requires a Fixed-Point Designer license.

Command-Line Information

Parameter: UseDivisionForNetSlopeComputation

Value: 'off' | 'on' | 'UseDivisionForReciprocalsOfIntegersOnly'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	On (when target hardware supports efficient division) Off (otherwise)
Safety precaution	No impact

See Also

Related Examples

- “Net Slope Computation” (Fixed-Point Designer)
- “Model Configuration Parameters: Code Generation Optimization” (Simulink Coder)

Gain parameters inherit a built-in integer type that is lossless

Description

The data type of the gain parameter is a built-in integer when the following conditions are met.

- The input type is a built-in integer.
- The **Parameter data type** is set to `Inherit:Inherit` via internal rule.
- The value of the gain parameter can be represented without losing any precision by a built-in integer.
- The **Parameter minimum** and **Parameter maximum** values in the **Parameter Attributes** tab of the Gain block parameters can be represented without losing any precision by a built-in integer.

Settings

Default: Off

On

For Gain blocks with the **Parameter data type** set to `Inherit:Inherit` via internal rule, the parameter data type compiles to a built-in integer type whenever possible.

Off

For Gain blocks with the **Parameter data type** set to `Inherit:Inherit` via internal rule, the parameter data type is the type which maximizes precision.

Tips

- This optimization affects both simulation and code generation.
- This optimization affects only Gain blocks in the model.
- For more efficient code, specify the **Parameter minimum** and **Parameter maximum** values in the **Parameter Attributes** tab of the Gain block parameters.

Dependencies

- In cases where the data type of the gain parameter compiles to a fixed-point data type (non-zero scaling), the software checks out a Fixed-Point Designer license.

Command-Line Information

Parameter: GainParamInheritBuiltInType

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	On (when target hardware supports efficient multiplication) Off (otherwise)
Safety precaution	No recommendation

Use floating-point multiplication to handle net slope corrections

Description

The Fixed-Point Designer software uses floating-point multiplication to perform net slope correction for floating-point to fixed-point casts.

Category: Math and Data Types

Settings

Default: Off

On

Use floating-point multiplication to perform net slope correction for floating-point to fixed-point casts.

Off

Use division to perform net slope correction for floating-point to fixed-point casts.

Tips

- This optimization affects both simulation and code generation.
- When converting from floating point to fixed point, if the net slope is not a power of two, slope correction using division improves precision. For some processors, use of multiplication improves code efficiency.

Dependencies

- This parameter requires a Fixed-Point Designer license.

Command-Line Information

Parameter: UseFloatMulNetSlope

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	On (when target hardware supports efficient multiplication) Off (otherwise)

Application	Setting
Safety precaution	No recommendation

See Also

Related Examples

- “Floating-Point Multiplication to Handle a Net Slope Correction” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Optimization” (Simulink Coder)

Inherit floating-point output type smaller than single precision

Description

Specify the desired inherited output data type behavior when block inputs are floating-point data types smaller than single precision.

Data types are smaller than single precision when the number of bits needed to encode the data type is less than the 32 bits needed to encode the single precision data type. For example, half and int16 are smaller than single precision.

This parameter affects only these blocks:

- Abs
- Add
- Difference
- Divide
- Dot Product
- Gain
- Math Function
- MinMax
- Product
- Product of Elements
- Sqrt
- Subtract
- Sum
- Sum of Elements

Settings

Default: Off

On

Inherit a floating-point output data type smaller than single precision when block inputs are floating-point data types smaller than single precision. In cases of overflow, the output data type is set to single precision.

Off

Use an internal rule to determine the output data type of the block. The internal rule chooses a data type that optimizes numerical accuracy, performance, and generated code size, while taking into account the properties of the embedded target hardware. It is not always possible for the software to optimize efficiency and numerical accuracy at the same time.

Tips

- This parameter affects blocks whose output data type is set to `Inherit: Inherit` via internal rule, `Inherit: Keep MSB`, `Inherit: Keep LSB`, or `Inherit: Match scaling` when the input is a floating-point data type smaller than single precision.
- This parameter affects both simulation and code generation.

Dependencies

- This parameter requires a Fixed-Point Designer license.

Command-Line Information

Parameter: `InheritOutputTypeSmallerThanSingle`

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	On (when targeting HDL code generation) Off (otherwise)
Safety precaution	No impact

See Also

“The Half-Precision Data Type in Simulink” (Fixed-Point Designer)

Application lifespan (days)

Description

Specify how long (in days) an application that contains blocks depending on elapsed or absolute time should be able to execute before timer overflow.

Category: Math and Data Types

Settings

Default: `auto`

Min: Must be greater than zero

Max: `inf`

Enter a positive (nonzero) scalar value (for example, `0.5`) or `inf`.

If you use Embedded Coder and select an ERT target for your model, the underlying value for `auto` is 1. If you are generating production code, you should set the value of this parameter based on your model.

If you use Simulink Coder and select a GRT target for your model, the underlying value for `auto` is `inf`.

This parameter is ignored when you are operating your model in external mode, have **MAT-file logging** enabled, or have a continuous sample time because a 64 bit timer is required in these cases.

Tips

- Specifying a lifespan, along with the simulation step size, determines the data type used by blocks to store absolute time values.
- For simulation, setting this parameter to a value greater than the simulation time will ensure time does not overflow.
- Simulink software evaluates this parameter first against the model workspace. If this does not resolve the parameter, Simulink software then evaluates it against the base workspace.
- The Application lifespan also determines the word size used by timers in the generated code, which can lower RAM usage. For more information, see “Control Memory Allocation for Time Counters” (Simulink Coder).
- Application lifespan, when combined with the step size of each task, determines the data type used for integer absolute time for each task, as follows:
 - If your model does not require absolute time, this option affects neither simulation nor the generated code.
 - If your model requires absolute time, this option optimizes the word size used for storing integer absolute time in generated code. This ensures that timers do not overflow within the lifespan you specify. If you set **Application lifespan** to `inf`, two `uint32` words are used.
 - If your model contains fixed-point blocks that require absolute time, this option affects both simulation and generated code.

For example, using 64 bits to store timing data enables models with a step size of 0.001 microsecond (1E-9 seconds) to run for more than 500 years, which would rarely be required. To run a model with a step size of one millisecond (0.001 seconds) for one day would require a 32-bit timer (but it could continue running for 49 days).

- A timer will allocate 64 bits of memory if you specify a value of `inf`.
- To minimize the amount of RAM used by time counters, specify a lifespan no longer than necessary.
- For code generation, must be the same for parent and referenced models. For simulation, the setting can be different for the parent and referenced models.
- Optimize the size of counters used to compute absolute and elapsed time.

Command-Line Information

Parameter: LifeSpan

Type: character vector

Value: positive (nonzero) scalar value or 'inf'

Default: 'auto'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Finite value
Safety precaution	inf

See Also

Related Examples

- “Optimize Memory Usage for Time Counters” (Simulink Coder)
- “Time-Based Scheduling and Code Generation” (Simulink Coder)
- “Timers in Asynchronous Tasks” (Simulink Coder)
- “Model Configuration Parameters: Code Generation Optimization” (Simulink Coder)
- “Performance” (Simulink Coder)

Use algorithms optimized for row-major array layout

Description

For certain blocks, enable optimized algorithms for row-major format code generation and corresponding row-major algorithms for model simulation.

Category: Math and Data Types

Settings

Default: Off

When **Array layout** (Simulink Coder) is set to `Row-major`, the code generator uses algorithms to maintain consistency of numeric results between the simulation and the generated code. Sometimes, the generated code for these algorithms might be inefficient. You can enable the **Use algorithms optimized for row-major array layout** configuration parameter to enable efficient algorithms that are optimized for certain blocks. The **Use algorithms optimized for row-major array layout** parameter affects the simulation and the generated code.

This parameter affects only these blocks:

- Sum of Elements
- Product of Elements
- n-D Lookup Table
- Interpolation Using Prelookup
- Direct Lookup Table (n-D)

For these blocks, the column-major and row-major algorithms might differ in the order of output calculations, possibly resulting in slightly different numeric values.

On

- When **Array layout** is set to `Row-major`, this parameter enables the use of efficient algorithms that traverse the data in row-major order. The generated code is efficient.
- When **Array layout** is set to `Column-major`, this parameter enables the use of algorithms that traverse the data in row-major order. The generated code is inefficient.

Off

- When **Array layout** is set to `Row-major`, the code generator uses algorithms that traverse the data in column-major order. The generated code is inefficient.
- When **Array layout** is set to `Column-major`, the code generator uses algorithms that traverse the data in column-major order. The generated code is efficient.

Tips

When **Array layout** is set to Row-major, the row-major algorithm operates on table data that is contiguous in memory. This table data leads to faster cache access, making these algorithms cache-friendly.

This table summarizes the relationship between array layout and cache-friendly algorithms. It is a best practice to use the algorithm that is optimized for the specified array layout to achieve good performance. For example, select **Use algorithms optimized for row-major array layout** when the **Array layout** is set to Row-major for code generation.

ArrayLayout	UseRowMajorAlgorithm	Algorithm Applied
Column-major	'off'	Efficient column-major algorithm Recommended
Row-major	'off'	Inefficient column-major algorithm Not recommended
Column-major	'on'	Inefficient row-major algorithm Not recommended
Row-major	'on'	Efficient row-major algorithm Recommended

Command-Line Information

Parameter: UseRowMajorAlgorithm

Type: character vector

Value: 'on' | 'off'

Default: 'off'

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Math and Data Types Pane” on page 22-2
- “Code Generation of Matrices and Arrays” (Simulink Coder)

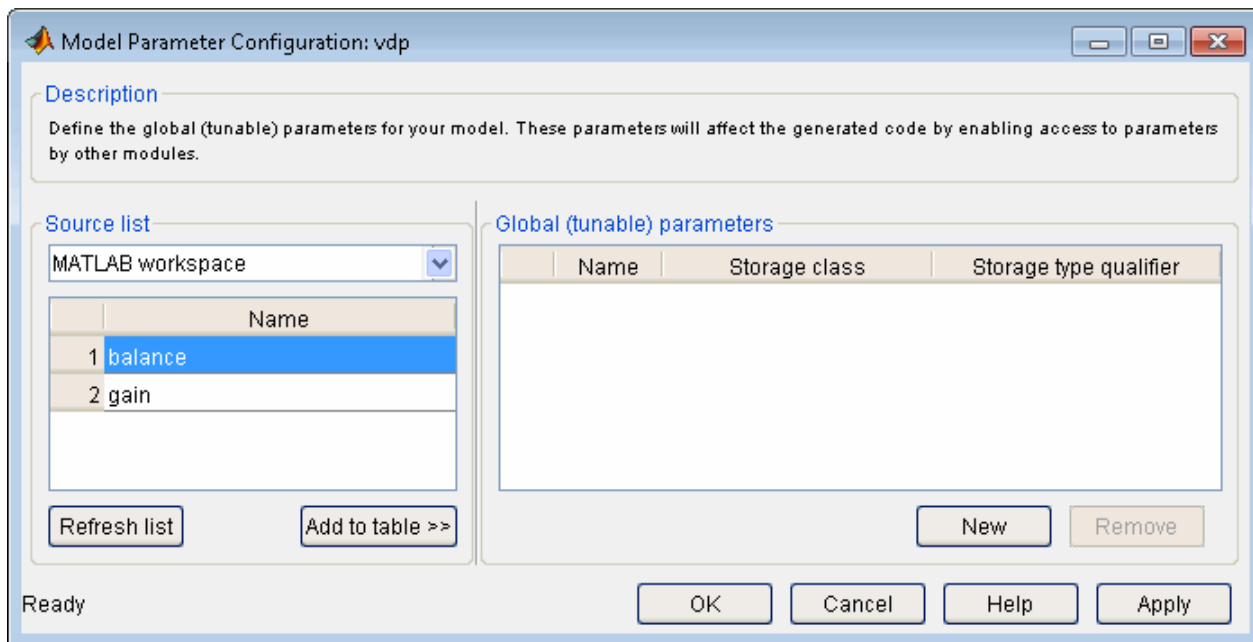
- “Row-Major Algorithms for Row-Major Array Layout” (Simulink Coder)

Model Parameter Configuration Dialog Box

Model Parameter Configuration Dialog Box

The **Model Parameter Configuration** dialog box allows you to declare specific tunable parameters when you set **Default parameter behavior** to **Inlined**. The parameters that you select appear in the generated code as tunable parameters. For more information about **Default parameter behavior**, see [Default parameter behavior \(Simulink Coder\)](#).

To declare tunable parameters, use `Simulink.Parameter` objects instead of the **Model Parameter Configuration** dialog box. See [“Create Tunable Calibration Parameter in the Generated Code” \(Simulink Coder\)](#).



Note Simulink Coder software ignores the settings of this dialog box if a model contains references to other models. However, you can still generate code that uses tunable parameters with model references, using `Simulink.Parameter` objects. See [“Create Tunable Calibration Parameter in the Generated Code” \(Simulink Coder\)](#).

The dialog box has the following controls.

Source list

Displays a list of workspace variables. The options are:

- **MATLAB workspace** — Lists all variables in the MATLAB workspace that have numeric values.
- **Referenced workspace variables** — Lists only those variables referenced by the model.

Refresh list

Updates the source list. Click this button if you have added a variable to the workspace since the last time the list was displayed.

Add to table

Adds the variables selected in the source list to the adjacent table of tunable parameters.

New

Defines a new parameter and adds it to the list of tunable parameters. Use this button to create tunable parameters that are not yet defined in the MATLAB workspace.

Note This option does not create the corresponding variable in the MATLAB workspace. You must create the variable yourself.

Storage class

Used for code generation. For more information, see “Choose Storage Class for Controlling Data Representation in Generated Code” (Simulink Coder).

Storage type qualifier

Used for code generation. For variables with a storage class other than `Auto`, you can add a qualifier (such as `const` or `volatile`) to the generated storage declaration. To do so, you can select a predefined qualifier from the list, or add qualifiers not in the list by typing them in.

See Also

Related Examples

- “Model Configuration Parameters: Code Generation Optimization” (Simulink Coder)
- “How Generated Code Stores Internal Signal, State, and Parameter Data” (Simulink Coder)

Model Advisor Parameters

- “Model Configuration Parameters: Model Advisor” on page 24-2
- “Model Advisor configuration file” on page 24-3
- “Show Model Advisor edit-time checks” on page 24-5

Model Configuration Parameters: Model Advisor

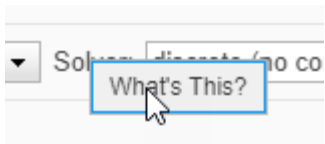
Model Advisor Pane Overview

The **Model Advisor** pane includes parameters for selecting a model-specific Model Advisor configuration file and enabling or disabling edit-time checking.

Parameter	Description
"Model Advisor configuration file" on page 24-3	Specifies a Model Advisor configuration file for the model.
"Show Model Advisor edit-time checks" on page 24-5	Enables Model Advisor edit-time checks for the model.

To get help on an option

- 1 Right-click the option text label.
- 2 From the context menu, select **What's This**.



Model Advisor configuration file

Description

Specify the Model Advisor configuration file for the model.

Category: Model Advisor

Settings

Default: ""

You can specify the Model Advisor configuration file by using one of these approaches:

- Use the Model Advisor. In the Model Advisor, in the **Configure** section, click **Load > Associate Configuration to Model**. When you click **Associate Configuration to Model**, the Model Advisor opens the Configuration Parameters for your model and sets the configuration parameter `ModelAdvisorConfigurationFile` to the full file path of the current configuration file. Click **OK** to accept the current configuration file as the configuration file for your model.
- Enter the name of your Model Advisor configuration file in this field.
- Specify a configuration at the command line by using the function `ModelAdvisor.setModelConfiguration`.

Tips

- The Model Advisor can use a configuration file to determine which Model Advisor and edit-time checks to run.
- To create your own custom Model Advisor configuration files, see “Use the Model Advisor Configuration Editor to Customize the Model Advisor” (Simulink Check).

Command-Line Information

Parameter: `ModelAdvisorConfigurationFile`

Type: string

Value: full file path for Model Advisor configuration file (JSON) saved with the Model Advisor Configuration Editor in R2022a release or later

Default: ""

Note The parameter `ModelAdvisorConfigurationFile` does not exist by default.

- Use the function `ModelAdvisor.setModelConfiguration` to set the model configuration.
 - Use the function `ModelAdvisor.getModelConfiguration` to get the model configuration.
-

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact

Show Model Advisor edit-time checks

Description

Enable Model Advisor edit-time checking for the model.

Category: Model Advisor

Settings

Default: Off

On

The Model Advisor shows edit-time checks on the Simulink canvas while you edit your model.

If you edit the model when edit-time checking is enabled, the Model Advisor checks out a Simulink Check™ license.

Off

The Model Advisor does not show edit-time checks on the Simulink canvas while you edit your model.

You can enable edit-time checking for your model by using one of these approaches:

- Navigate to the Model Advisor Configuration Parameters from the Simulink toolstrip for your model. In the **Debug** tab, click **Diagnostics > Edit-Time Checks**. Select the check box for **Edit-Time Checks**.
- In the **Modeling** tab, click **Model Advisor > Edit-Time Checks**. Select the check box for **Edit-Time Checks**.
- Enable edit-time checking at the command line by using the function `edittime.setAdvisorChecking`.

Tip

For more information on edit-time checking, see “Check Model Compliance Using Edit-Time Checking” (Simulink Check).

Command-Line Information

Parameter: ShowAdvisorChecksEditTime

Type: string

Value: "On" | "Off"

Default: "Off"

Note The parameter ShowAdvisorChecksEditTime does not exist by default.

Use the function `edittime.setAdvisorChecking` to enable or disable edit-time checking for the model.

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	No impact