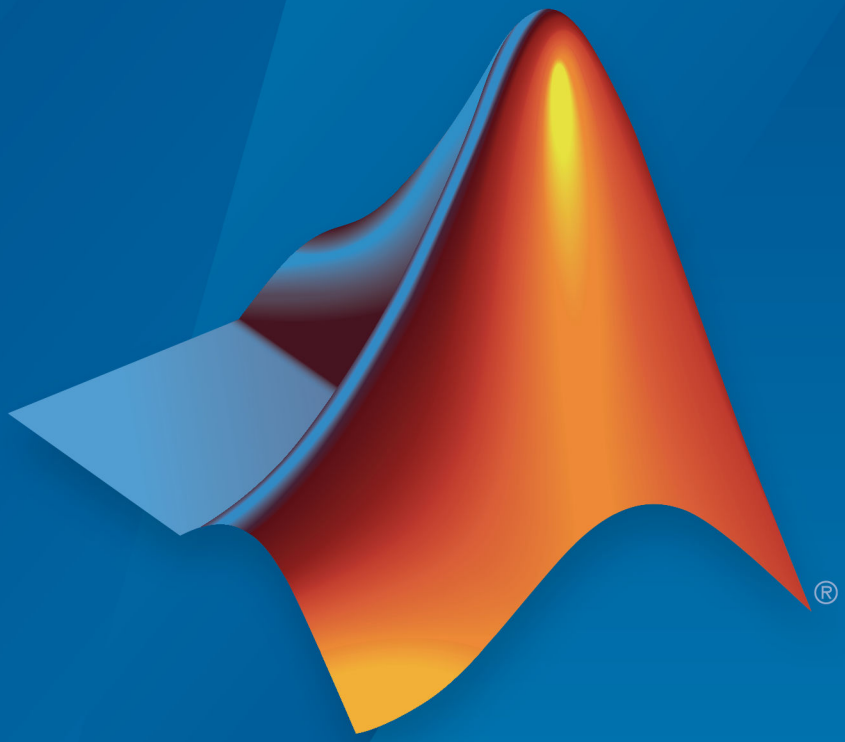
Simulink®

Graphical User Interface



MATLAB® & SIMULINK®

R2017b

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

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

Simulink® Graphical User Interface

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)

Contents

1 Configuration Parameters Dialog Box

2 Simulink Configuration Parameters: Advanced

3 Data Import/Export Parameters

4 Optimization Parameters

5 Optimization Parameters: Signals and Parameters

6 Optimization Parameters: Stateflow

7 Diagnostics Parameters: Compatibility

8 Diagnostics Parameters: Connectivity

9 Diagnostics Parameters: Data Validity

10 Diagnostics Parameters: Model Referencing

11 Diagnostics Parameters: Sample Time

13 Diagnostics Parameters: Stateflow

14 Diagnostics Parameters: Type Conversion

15 Model Referencing Parameters

16 Simulation Target Parameters

17 Solver Parameters

19 Signal Properties Dialog Box

20 Simulink Preferences Window

21 Simulink Mask Editor

23 Simulink Simulation Stepper

24 Simulink Variant Manager

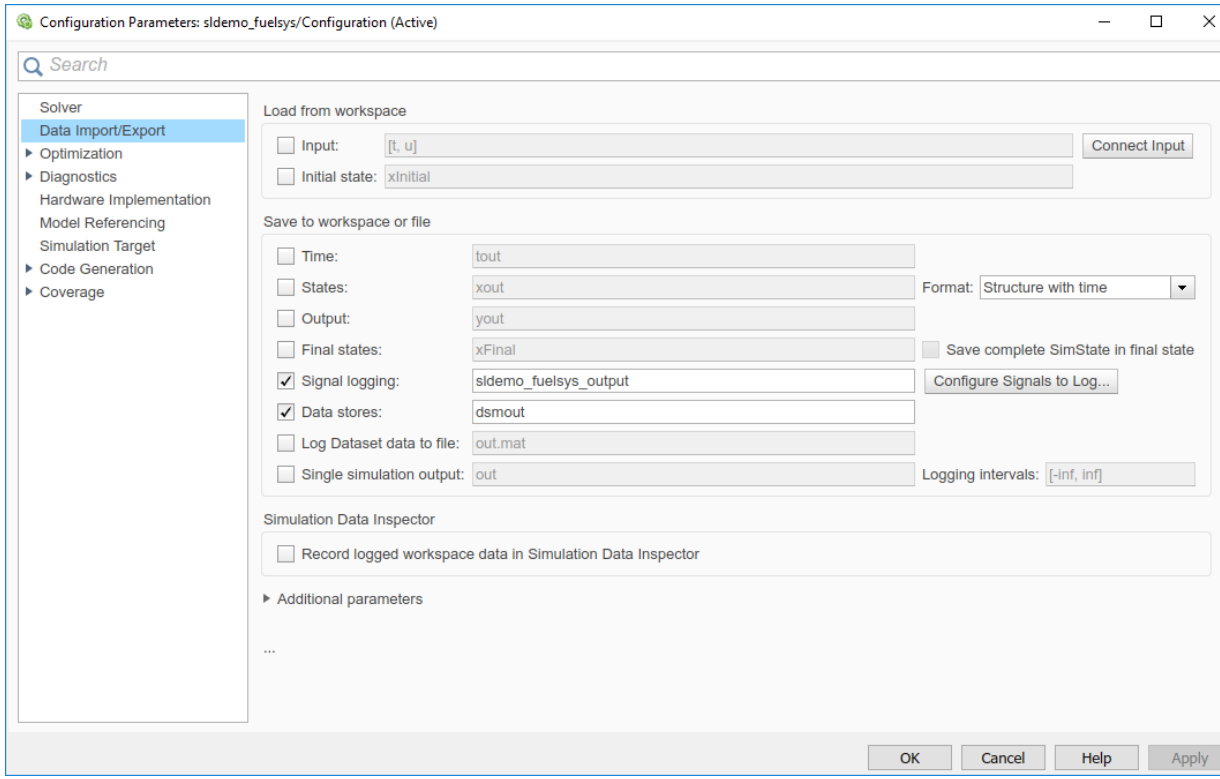
Configuration Parameters Dialog Box

Configuration Parameters Dialog Box Overview

The Configuration Parameters dialog box specifies the settings for the active configuration set of a model. The parameters in a configuration set determine the type of solver used, import and export settings, and other values that determine how the model runs. See Configuration Sets for more information.

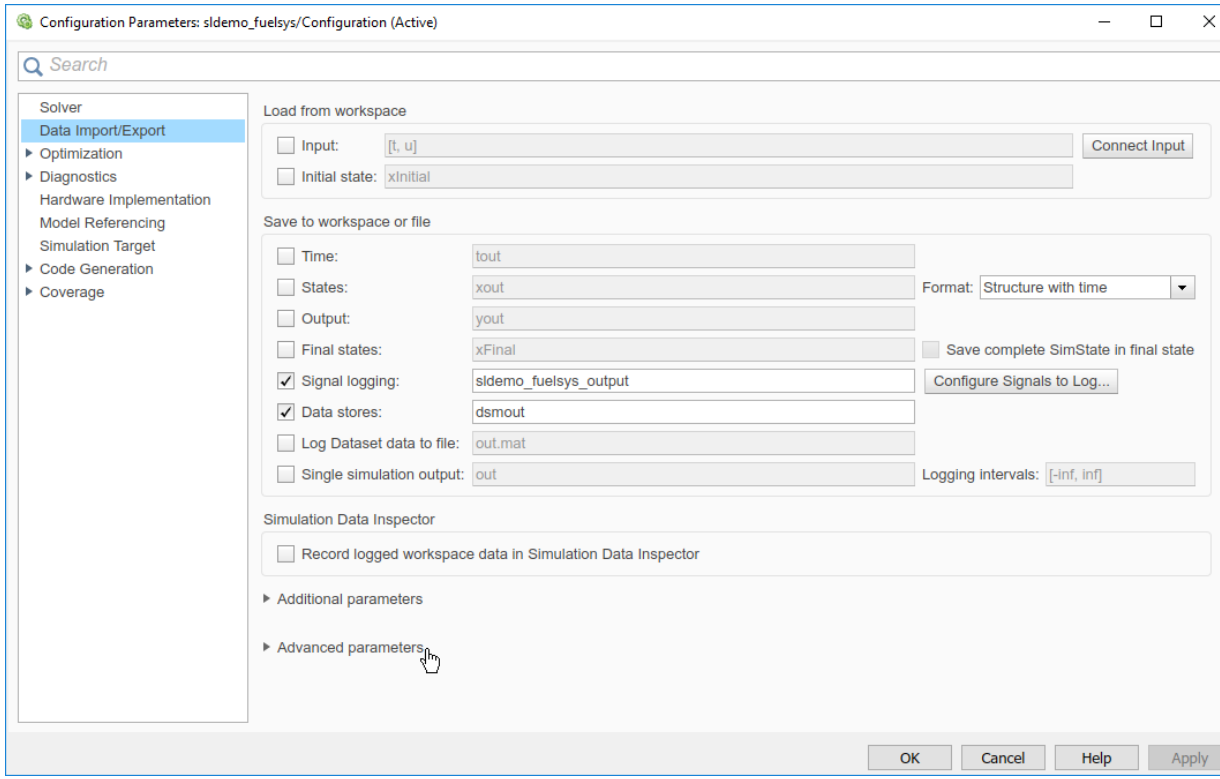
Note You can also use the Model Explorer to modify any configuration set. See "Search and Edit Using Model Explorer" for more information.

To open the dialog box, in the Simulink Editor, select **Simulation > Model Configuration Parameters**, or click the Model Configuration Parameters button  on the Simulink Editor toolbar.



To display the parameters for a specific category, click the category in the **Select** tree on the left side of the dialog box. The **Search** box at the top of the dialog box performs a keyword search of all parameters. The search tool supports regular expressions. Type `.` in the search box to see a list of all parameters.

You can access the **Advanced parameters** for each category by mousing over the ellipsis toward the bottom of the dialog box and clicking **Advanced parameters**.



Right-click a parameter name and select **What's This?** to see:

- A short parameter description.
- The parameter name that you can use in scripts.
- Parameter dependencies.

From the **What's This?** dialog box, click **Show more information** for complete parameter documentation.

Model Configuration Pane

In this section...

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.

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

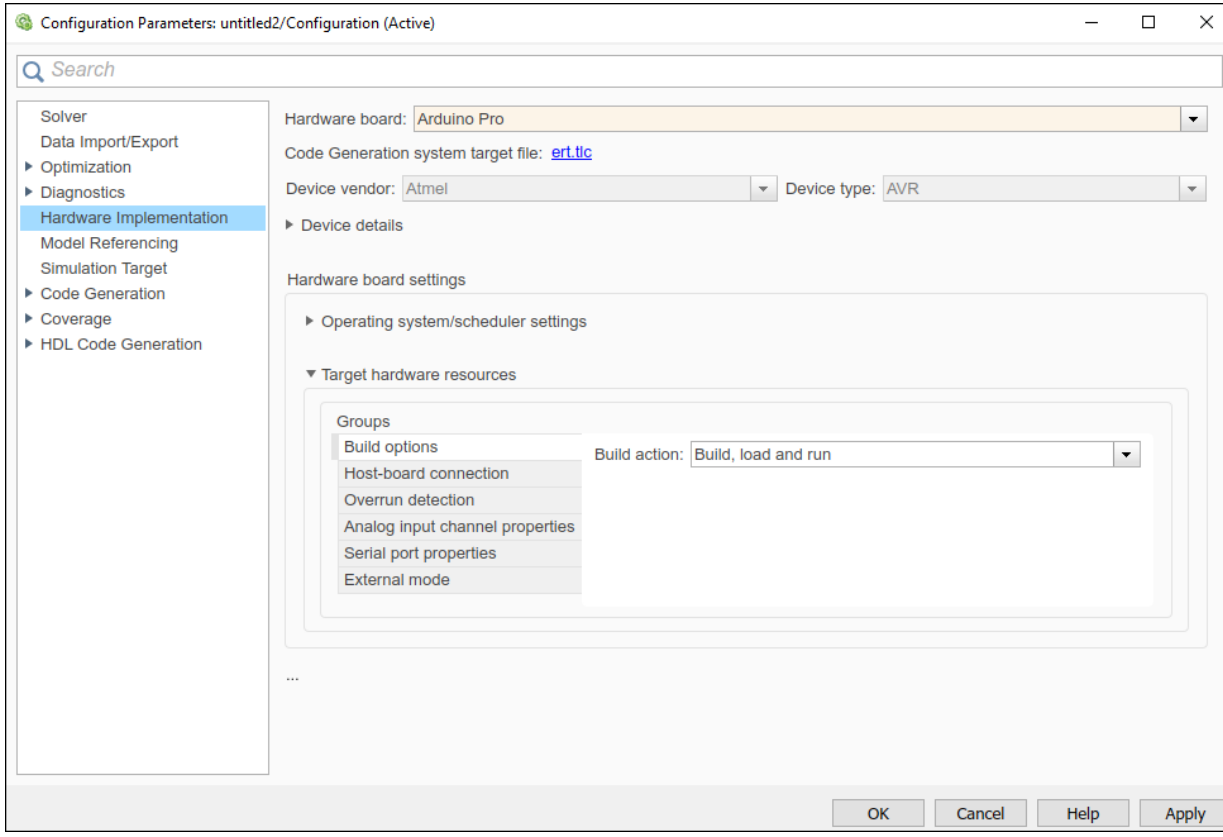
No Default

Enter text to describe your configuration set.

Configuration Parameters

No further help documentation is available for this parameter.

Hardware Implementation Pane



In this section...

In this section...

In this section...

In this section...

Hardware Implementation Overview

Specify hardware options to simulate and generate code for models of computer-based systems, such as embedded controllers.

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.

See Also

- Hardware Implementation Options (Simulink Coder)
- “Hardware Implementation Pane” on page 1-7

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

- "Device type" on page 1-14
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- "Hardware Implementation Pane" on page 1-7

Code Generation system target file

System target file that you select on the **Code Generation** pane.

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`
- `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 More Device Vendor and Device Type Values" (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 board" on page 1-10
- "Device type" on page 1-14
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- "Hardware Implementation Pane" on page 1-7

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`

Altera® options:

- `SoC (ARM CortexA)`

Analog Devices® options:

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

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`
- `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	✓	□
K5/K6/ Athlon	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
x86-32 (Windows 32)	8	16	32	32	64	32	32	32	32	Char	Float	Little Endian	Zero	✓	□
x86-64 (Linux 64)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
x86-64 (macOS)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□

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 (Windows 64)	8	16	32	32	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
ARM Compatible															
ARM 7/8/9/10	8	16	32	32	64	32	32	32	32	Long	Float	Little Endian	Zero	✓	□
ARM 11	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
ARM Cortex	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Altera															
SoC (ARM Cortex A)	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
Analog Devices															
ADSP-CM40x(ARM Cortex-M)	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□

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	✓	□
SHARC	32	32	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	□
TigerSHARC	32	32	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Atmel															
AVR	8	16	16	32	64	8	16	16	16	Char	None	Little Endian	Zero	✓	□
AVR (32-bit)	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
AVR (8-bit)	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
Freescale															
32-bit PowerPC	8	16	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	□

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				
68332	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	□
68HC08	8	16	16	32	64	8	8	16	8	Char	None	Big Endian	Zero	✓	□
68HC11	8	16	16	32	64	8	8	16	16	Char	None	Big Endian	Zero	✓	□
ColdFire	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	□
DSP563xx (16-bit mode)	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
DSP5685x	8	16	16	32	64	16	16	16	16	Char	Float	Little Endian	Zero	✓	□
HC(S)12	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	□

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	✓	□
MPC85xx	8	16	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	□
S08	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	□
S12x	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	□
StarCore	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
Infineon															

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				
C16x, XC16x	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
TriCore	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
Intel															
x86-32 (Windows 32)	8	16	32	32	64	32	32	32	32	Char	Float	Little Endian	Zero	✓	□
x86-64 (Linux 64)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
x86-64 (macOS)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
x86-64 (Windows 64)	8	16	32	32	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
Microchip															
PIC18	8	16	16	32	64	8	8	24	24	Char	None	Little Endian	Zero	✓	□

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				
dsPIC	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
NXP															
Cortex–M0/M0+	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Cortex–M3	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Cortex–M4	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Renesas															
M16C	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
M32C	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
R8C/Tiny	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□

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				
RH850	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
RL78	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
SH-2/3/4	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	□
V850	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
STMicroelectronics															
ST10/ Super10	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
Texas Instruments															
C2000	16	16	16	32	64	16	32	16	16	Int	None	Little Endian	Zero	✓	□
C5000	16	16	16	32	64	16	16	16	16	Int	None	Big Endian	Zero	✓	□

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	✓	□
MSP430	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
Stellaris Cortex—M3	8	16	32	32	6	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
TMS470	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
TMS570 Cortex—R4	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
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 More Device Vendor and Device Type Values" (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 board" on page 1-10
- "Device vendor" on page 1-12
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- "Hardware Implementation Pane" on page 1-7

Device details

Click the arrow to list parameters for:

- Data type bit specifications
- Largest atomic sizes for integer and floating-point values
- Byte ordering
- What signed integer division rounds to
- Whether signed integer as an arithmetic shift shifts right

- Whether there is support for the `long long` data type

Number of bits: char

Describe the character bit length for the hardware.

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 Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- “Hardware Implementation Pane” on page 1-7

Number of bits: short

Describe the data bit length for the hardware.

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
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)
- “Hardware Implementation Pane” on page 1-7

Number of bits: int

Describe the data integer bit length for the hardware.

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
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)
- “Hardware Implementation Pane” on page 1-7

Number of bits: long

Describe the data bit lengths for the hardware.

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
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)
- “Hardware Implementation Pane” on page 1-7

Number of bits: long long

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

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
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

- "Support long long" on page 1-51
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- "Hardware Implementation Pane" on page 1-7

Number of bits: float

Describe the bit length of floating-point data for the hardware (read only).

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)
- “Hardware Implementation Pane” on page 1-7

Number of bits: double

Describe the bit-length of `double` data for the hardware (read only).

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 Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- “Hardware Implementation Pane” on page 1-7

Number of bits: native

Describe the microprocessor native word size for the hardware.

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: `32`

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 Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- “Hardware Implementation Pane” on page 1-7

Number of bits: pointer

Describe the bit-length of pointer data for the hardware.

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 Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- “Hardware Implementation Pane” on page 1-7

Number of bits: size_t

Describe the bit-length of `size_t` data for the hardware.

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

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 Options (Simulink Coder)

- Specifying Production Hardware Characteristics (Simulink Coder)
- "Hardware Implementation Pane" on page 1-7

"Verification of Code Generation Assumptions" (Embedded Coder)

Number of bits: ptrdiff_t

Describe the bit-length of `ptrdiff_t` data for the hardware.

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

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

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 Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- “Hardware Implementation Pane” on page 1-7

“Verification of Code Generation Assumptions” (Embedded Coder)

Largest atomic size: integer

Specify the largest integer data type that can be atomically loaded and stored on the hardware.

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

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 Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- “Hardware Implementation Pane” on page 1-7

Largest atomic size: floating-point

Specify the largest floating-point data type that can be atomically loaded and stored on the hardware.

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
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 Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- “Hardware Implementation Pane” on page 1-7

Byte ordering

Describe the byte ordering for the hardware board.

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: `ProdEndianess`
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 Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- "Hardware Implementation Pane" on page 1-7

Signed integer division rounds to

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

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.

Application	Setting
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 Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- “Hardware Implementation Pane” on page 1-7

Shift right on a signed integer as arithmetic shift

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

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
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)

• “Hardware Implementation Pane” on page 1-7

Support long long

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

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

Application	Setting
Traceability	No impact
Efficiency	On (execution, ROM)
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

- “Number of bits: long long” on page 1-34
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- “Hardware Implementation Pane” on page 1-7

Device vendor

Select the manufacturer of the hardware board to use to implement the test system that this model represents.

Settings

Default: `Intel`

- `AMD`
- `ARM Compatible`
- `Altera`
- `Analog Devices`
- `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 `TargetHWDeviceType`. When specifying this parameter from 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 More Device Vendor and Device Type Values" (Simulink Coder).

Dependencies

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

Command-Line Information

Parameter: `TargetHWDeviceType_Vendor`
Type: string
Value: any valid value (see tips)
Default: `'Intel'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact

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

- "Hardware board" on page 1-10
- "Device type" on page 1-54
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- "Hardware Implementation Pane" on page 1-7

Device type

Select the type of hardware to use to implement the test system.

Settings

Default: `x86-64 (Windows64)`

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`

Altera options:

- `SoC (ARM CortexA)`

Analog Devices options:

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

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`
- `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.
- Selecting a device type specifies the hardware device to define system constraints:
  - Default hardware properties appear in the dialog box display 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.

  This 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	✓	□
K5/K6/ Athlon	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
x86-32 (Windows 32)	8	16	32	32	64	32	32	32	32	Char	Float	Little Endian	Zero	✓	□
x86-64 (Linux 64)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
x86-64 (macOS)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
x86-64 (Windows 64)	8	16	32	32	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
ARM Compatible															
ARM 7/8/9/10	8	16	32	32	64	32	32	32	32	Long	Float	Little Endian	Zero	✓	□

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 11	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
ARM Cortex	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Altera															
SoC (ARM Cortex A)	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
Analog Devices															
ADSP-CM40x(ARM Cortex-M)	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Blackfin	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
SHARC	32	32	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	□

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				
TigerSHARC	32	32	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Atmel															
AVR	8	16	16	32	64	8	16	16	16	Char	None	Little Endian	Zero	✓	□
AVR (32-bit)	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
AVR (8-bit)	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
Freescale															
32-bit PowerPC	8	16	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	□
68332	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	□
68HC08	8	16	16	32	64	8	8	16	8	Char	None	Big Endian	Zero	✓	□

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	✓	□
ColdFire	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	□
DSP563xx (16-bit mode)	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
DSP5685x	8	16	16	32	64	16	16	16	16	Char	Float	Little Endian	Zero	✓	□
HC(S)12	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	□

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	✓	□
MPC85xx	8	16	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	□
S08	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	□
S12x	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	□
StarCore	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
Infineon															

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				
C16x, XC16x	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
TriCore	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
Intel															
x86-32 (Windows 32)	8	16	32	32	64	32	32	32	32	Char	Float	Little Endian	Zero	✓	□
x86-64 (Linux 64)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
x86-64 (macOS)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
x86-64 (Windows 64)	8	16	32	32	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
Microchip															
PIC18	8	16	16	32	64	8	8	24	24	Char	None	Little Endian	Zero	✓	□

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				
dsPIC	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
NXP															
Cortex–M0/M0+	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Cortex–M3	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Cortex–M4	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Renesas															
M16C	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
M32C	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
R8C/Tiny	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□

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				
RH850	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
RL78	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
SH-2/3/4	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	□
V850	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
STMicroelectronics															
ST10/ Super10	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
Texas Instruments															
C2000	16	16	16	32	64	16	32	16	16	Int	None	Little Endian	Zero	✓	□
C5000	16	16	16	32	64	16	16	16	16	Int	None	Big Endian	Zero	✓	□

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	✓	□
MSP430	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
Stellaris Cortex—M3	8	16	32	32	6	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
TMS470	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
TMS570 Cortex—R4	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
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 atth e 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 More Device Vendor and Device Type Values" (Simulink Coder).

Dependencies

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

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 the device type.

Command-Line Information

Parameter: `TargetHWDeviceType_Type`
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 impact

See Also

- "Hardware board" on page 1-10
- "Device vendor" on page 1-52
- Hardware Implementation Options (Simulink Coder)
- Specifying Production Hardware Characteristics (Simulink Coder)
- "Hardware Implementation Pane" on page 1-7

Advanced Parameters

These parameters appear under the **Advanced parameters**.

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 2-4	Select the manufacturer and type of the hardware to use to test the code generated from the model.
"Number of bits: char" on page 2-19	Describe the character bit length for the hardware that you use to test code.
"Number of bits: short" on page 2-21	Describe the data bit length for the hardware that you use to test code.

Parameter	Description
"Number of bits: int" on page 2-23	Describe the data integer bit length of the hardware that you use to test code.
"Number of bits: long" on page 2-25	Describe the data bit lengths for the hardware that you use to test code.
"Number of bits: long long" on page 2-27	Describe the length in bits of the C `long long` data type that the test hardware supports.
"Number of bits: float" on page 2-29	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 2-31	Describe the bit-length of `double` data for the hardware that you use to test code (read only).
"Number of bits: native" on page 2-33	Describe the microprocessor native word size for the hardware that you use to test code.
"Number of bits: pointer" on page 2-35	Describe the bit-length of pointer data for the hardware that you use to test code.
"Number of bits: size_t" on page 2-37	Describe the bit-length of `size_t` data for the hardware that you use to test code.
"Number of bits: ptrdiff_t" on page 2-39	Describe the bit-length of `ptrdiff_t` data for the hardware that you use to test code.
"Largest atomic size: integer" on page 2-41	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 2-43	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 2-45	Describe the byte ordering for the hardware that you use to test code.
"Signed integer division rounds to" on page 2-47	Describe how your compiler for the test hardware rounds the result of dividing two signed integers.

Parameter	Description
“Shift right on a signed integer as arithmetic shift” on page 2-50	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 2-52	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.

Connection type

Select the connection type to connect between the host computer and the LEGO®EV3 brick.

The connection types available are:

- USB - This is the default option to connect your host computer with the EV3 brick using a USB mini cable.
- Bluetooth - Use this option when you connect your host computer to the EV3 brick using Bluetooth. EV3 has in-built bluetooth. The computer to which you want to connect to needs a bluetooth module. If your computer does not have an in-built bluetooth module, then use an external bluetooth dongle.
- WiFi - Use this option when you connect your host computer to the EV3 brick using WiFi network connection.

- Ethernet - Use this option when you connect your host computer to the EV3 brick using Ethernet connection.

SPI0 CE0 Bus Speed (kHz)

Select a value from the list of bus speed in kilo hertz to obtain an SPI bus speed for channel 0.

SPI0 CE1 Bus Speed (kHz)

Select a value from the list of bus speed in kilo hertz to obtain an SPI bus speed for channel 1.

Run external mode in a background

Select this check box to force the External mode engine to run the generated code in a background task.

Make sure you do not select this option for a model that has very small time step or that may encounter overruns as this may result in Simulink becoming nonresponsive.

Enable External mode

Enable External mode to tune and monitor a model while it runs on your hardware board.

With External mode, changing a parameter value in the model on the host changes the corresponding value in the model running on the hardware. Similarly, scopes in the model display data from the model running on the hardware.

Enabling External mode adds a lightweight server to the model running on the hardware board. This server increases the processing burden upon the hardware board, which can result in an overrun condition. If you enable the **Enable overrun detection** check box, and the software reports an overrun, consider disabling External mode.

Enabling the **External mode** parameter makes the following communication-related parameters visible:

- **Set host COM port** LEGO MINDSTORMS® NXT hardware and Arduino® Mega 2560 hardware

- **TCP/IP port (1024-65535)** for BeagleBoard hardware

Enabling the **External mode** parameter disables the **Enable communication between two NXT bricks** parameter LEGO MINDSTORMS NXT hardware.

Settings

Default: Disabled

Disabled
: The model application does not support External mode.

Enabled
: The model application supports External mode.

Digital output to set on overrun

This parameter appears when the **Hardware board** parameter is set to an Arduino hardware and the **Enable overrun detection** check box is selected.

Select the digital output pin the Arduino hardware uses to signal a task overrun.

Do not use a pin that is assigned to another block within the model.

Settings

Default: `13`

Device ID

Enter the device ID of the LEGOEV3 brick. You can use the EV3 Brick Interface to get the Device ID from the **Brick Info** screen.

This parameter appears when you select `Bluetooth`, `WiFi`, or `Ethernet` option in **Connection type** parameter.

IP address

Enter the IP address of the LEGOEV3 brick.

You can use the EV3 Brick Interface to get the IP address from the **Brick Info**.

This parameter appears when you select `WiFi`or `Ethernet` option in **Connection type** parameter.

Base Rate Task Priority

This parameter sets the static priority of the base rate task. However, the changes that you make in this parameter do not result in any functionality differences on the LEGO MINDSTORMS EV3 brick.

Settings

Default: `40`

Detect task overruns

Detect when a task overrun occurs in a Simulink model running on the target hardware. Indicate when an overrun has occurred.

A task overrun occurs if the target hardware is still performing one instance of a task when the next instance of that task is scheduled to begin.

You can fix overruns by decreasing the frequency with which tasks are scheduled to run, and by reducing the number or complexity of the tasks defined by your model.

If those solutions do not fix the task overrun condition, and you are using External mode, consider disabling External mode.

Settings

Default: `None`

Device Address

Enter the IP address or host name of the hardware board.

When you use the Support Package Installer to update the firmware on the board, the Support Package Installer automatically gets the value of the IP address from the board and applies it to this parameter.

If you swap boards, or change the IP address of the board, get the value of the new IP address and enter it here.

Settings

Default: `192.168.0.101`

Username

Enter the root user name for Linux® running on the hardware board.

When you use the Support Package Installer to update the hardware board firmware, the Support Package Installer automatically applies the value you entered there to this parameter.

Settings

Default: `pi`

Password

Enter the root password for Linux running on the hardware board.

When you use the Support Package Installer to update the hardware board firmware, the Support Package Installer automatically applies the value you entered there to this parameter.

Settings

Default: `raspberry`

Build action

Specify whether you want only build or build, load, and run actions during code generation.

Settings

Default: `Build, load and run`

`Build`

Build the code during the build process.

`Build, load and run`

Build, load, and to run the generated code during the build process.

Build directory

Enter the build directory for Linux running on the hardware board.

When you use the Support Package Installer to update the hardware board firmware, the Support Package Installer automatically applies the value you entered there to this parameter.

Settings

Default: `/home/pi`

Set host COM port

Automatically detect or manually set the COM port your host computer uses to communicate with the hardware board.

This parameter appears when the **Hardware board** parameter is set to `LEGO MINDSTORMS NXT`, `Arduino Mega 2560`, or `Arduino Uno`.

Warning Do not connect Arduino Uno and Arduino Mega 2560 to a RS-232 serial interface, commonly found on computers and equipment. RS-232 interfaces can use voltages greater than 5 Volts, which can damage your Arduino hardware.

Settings

Default: `Automatically`

`Automatically`

Let the software determine which COM Port your host computer uses.

`Manually`

Select this option to display the **COM port number** parameter.

Analog input reference voltage

Set the reference voltage used to measure inputs to the ANALOG IN pins.

This parameter appears when the **Target hardware** parameter is set to `Arduino Mega 2560` or `Arduino Uno`.

Warning Only connect an external power source to AREF while this parameter is set to `External`. Connecting an external power source to AREF while this parameter is set to any other option exposes the internal voltage references to the external voltage. This voltage difference can damage your hardware.

Do not connect Arduino Uno and Arduino Mega 2560 to voltages greater than 5 Volts.

Do not connect Arduino Due to voltages greater than 3.3 Volts.

Voltages greater than the specified limits can damage your Arduino hardware.

Settings

Default: `Default`

`Default`

Use the default operating voltage of the board. For Arduino Uno and Arduino Mega 2560 the operating voltage is 5 Volts.

`Internal (1.1 V)`

Valid for Arduino Mega 2560 only: Use the internal 1.1 Volt reference.

`Internal (2.56 V)`

Valid for Arduino Mega 2560 only: Use the internal 2.56 Volt reference.

`External`

On the Arduino Uno, Arduino Nano and Arduino Mega 2560, use an external 0-5 volt power supply connected to the AREF pin. This voltage should match the voltage of the power supply connected to the Arduino hardware. If your application requires low-noise measurements, use this option with a filtered power supply.

Serial 0 baud rate, Serial 1 baud rate, Serial 2 baud rate, Serial 3 baud rate

Set the baud rate of the serial port on the Arduino hardware.

If you set **Set host COM port** to `Manually`, then set **Serial 0 baud rate** as described in the "Set the COM Port and Baud Rate Manually" topic.

For information on serial ports for different Arduino boards, see "Pin Mapping on Arduino Blocks" (Simulink Support Package for Arduino Hardware).

Settings

Default: `9600`

`300`, `1200`, `2400`, `4800`, `9600`, `14400`, `19200`, `28800`, `38400`, `57600`, `76800`, `115200`, `128000`, `500000`, `1000000`

SPI clock out frequency (in MHz)

Select a value from the list of master clock frequency to obtain an SPI clock frequency.

Settings

Default: `4000`

`8000`, `4000`, `2000`, `1000`, `500`, `250`, `125`

SPI mode

Select an SPI mode for data transmission.

Settings

Default: `Mode 0 - Clock Polarity 0, Clock Phase 0`

- `Mode 0 - Clock Polarity 0, Clock Phase 0`
- `Mode 1 - Clock Polarity 0, Clock Phase 1`
- `Mode 2 - Clock Polarity 1, Clock Phase 0`

- `Mode 3- Clock Polarity 1, Clock Phase 1`

Bit order

Select the bit order for transmissions.

`MSB first` to send the most significant bit first for transmission or select `LSB first` to send the least significant bit first for transmission.

Settings

Default: `MSB first`

- `MSB first` – Send the most significant bit first for transmission.
- `LSB first` – Send the least significant bit first for transmission.

Use static IP address and disable DHCP

Select this check box to disable the DHCP. The static IP address provided is used for setting up the ethernet connection

IP address (Ethernet shield)

Enter the IP address of the Arduino Ethernet shield.

MAC address

Enter the machine address of the Arduino Ethernet shield.

Use static IP address and disable DHCP

Select this check box to disable the DHCP. The static IP address provided is used for setting up the WiFi connection.

Connection type

Enter the type of connection that should be used to download the executable from the host machine to the LEGO MINDSTORMS EV3 brick.

Device ID

Enter the ID of the LEGO MINDSTORMS EV3 brick. You can find this information in the 'Brick Info' pane of the LEGO MINDSTORMS EV3 brick.

IP address

Enter the IP address of the LEGO MINDSTORMS EV3 brick..

IP address (WiFi shield)

Enter the IP address of the Arduino WiFi shield.

Service set identifier (SSID)

Enter the SSID of your network. An SSID is a unique ID consisting of 32 characters and is used for naming wireless networks. An SSID ensures that the data you send over the network reaches the correct destination.

WiFi encryption

The WiFi encryption of the network you connect to.

Settings

Default: `None`

`None`
: Network is not WiFi encrypted.

`WPA`
: Network uses WPA WiFi encryption.

`WEP`
: Network uses WEP WiFi encryption.

WEP key

Enter the WEP key of the network.

This parameter appears only when you select `WEP` for the **WiFi encryption** parameter.

WEP key index

Enter the WEP key index of the WEP key.

This parameter appears only when you select `WEP` for the **WiFi encryption** parameter.

WPA password

Enter the WPA password of the network.

This parameter appears only when you select `WPA` for the **WiFi encryption** parameter.

Connect to custom ThingSpeak server

Select this check box to connect to the custom ThingSpeak server. Otherwise, the ThingSpeak block connects to the default address of 184.106.153.149 through port 80.

Server IP address

Specify the IP address of the custom ThingSpeak server.

Port

Specify the port number through which the ThingSpeak block connects to the ThingSpeak server.

Communication interface

Use the 'serial' option to run your model in the External mode with serial communication.

Settings

Default: `Serial`

- `Serial`

- `TCP/IP`
- `WiFi`

Device

Select the device you are using. The list includes any devices that are connected to your computer and turned on.

To see a device that was recently connected and turned on, click **Refresh**.

Settings

Default: `No devices detected`

Package name

All Android applications have a full Java-language-style **package name**. The **package name** should be unique. To avoid conflicts with other developers, use Internet domain ownership, in the reverse order, as the basis for your package name. For example `com.mydomain.myappname`.

Settings

Default: `com.example`

Port

Set the value of the TCP/IP or WiFi port number, from 1024 to 65535. External mode uses this port for communications between the hardware board and host computer.

Settings

Default: `17725`

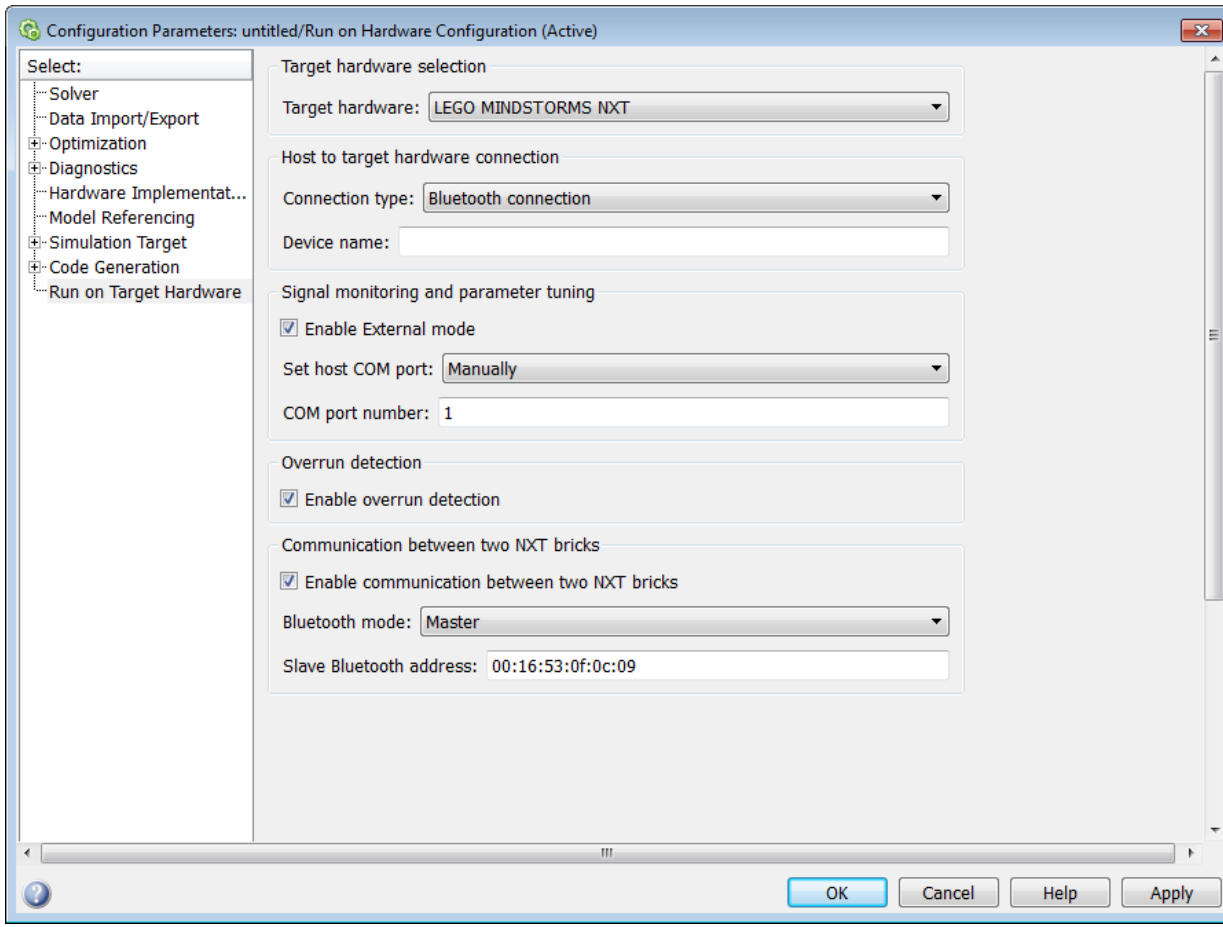
Verbose

Select this check box to view the External Mode execution progress and updates in the Diagnostic Viewer or in the MATLAB Command Window. This parameter appears when you select `TCP/IP` or `WiFi` for **Communication interface**.

IP Address

Enter the IP address of the LEGO MINDSTORMS EV3 brick.

Run on Target Hardware Pane



In this section...

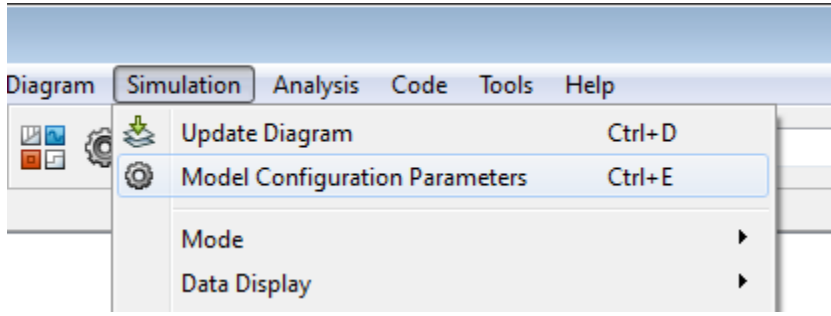
In this section...

Hardware Implementation Pane Overview

Specify the options for creating and running applications on target hardware.

Configuration

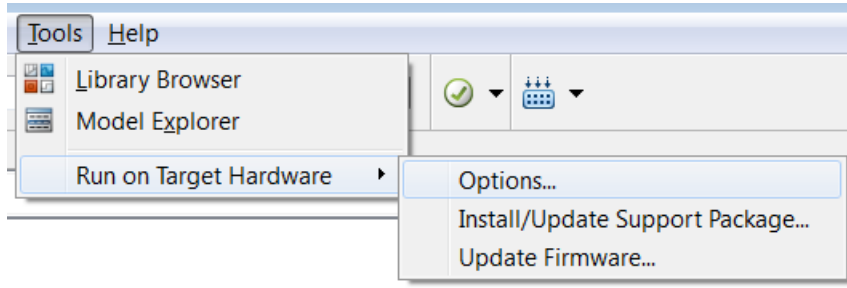
To configure a Simulink model to run on the target hardware, select **Simulation** > **Model Configuration Parameters**.



1. In the Configuration Parameters dialog box, click **Hardware Implementation**.
2. Select the **Hardware board** to match your target hardware.
3. (Optional) Review and set the other parameters.
4. Apply the changes.

Tip

After configuring a Simulink model, you can reopen the configuration parameters dialog box by selecting **Simulation** > **Model Configuration Parameters**.



Target hardware

Select the type of hardware upon which to run your model.

Changing this parameter updates the Configuration Parameters dialog so it only displays parameters that are relevant to your target hardware.

To install support for target hardware, start Support Package Installer by selecting `Get more`, or by entering `supportPackageInstaller` in the MATLAB Command Window.

After installing support for your target hardware, reopen the Configuration Parameters dialog and select your target hardware.

Settings

Default: `None`

`None`

This setting means your model has not been configured to run on target hardware. Choose your target hardware from the list of options.

`Get more...`

Select this option to start Support Package Installer and install support for additional hardware.

External mode transport layer

Select the transport layer the External mode uses to communicate between the Arduino hardware and the host computer:

- `serial` uses the standard serial USB connection.
- `tcpip` uses the Ethernet connection specified by the **Ethernet shield properties**.
- `wifi` uses the Wi-Fi connection specified by the **WiFi shield properties**.

Enable External mode

Enable External mode to tune and monitor a model while it runs on your hardware board.

With External mode, changing a parameter value in the model on the host changes the corresponding value in the model running on the hardware. Similarly, scopes in the model display data from the model running on the hardware.

Enabling External mode adds a lightweight server to the model running on the hardware board. This server increases the processing burden upon the hardware board, which can result in an overrun condition. If you enable the **Enable overrun detection** check box, and the software reports an overrun, consider disabling External mode.

Enabling the **External mode** parameter makes the following communication-related parameters visible:

- **Set host COM port** LEGO MINDSTORMS NXT hardware and Arduino Mega 2560 hardware
- **TCP/IP port (1024-65535)** for BeagleBoard hardware

Enabling the **External mode** parameter disables the **Enable communication between two NXT bricks** parameter LEGO MINDSTORMS NXT hardware.

Settings

Default: Disabled

Disabled

The model application does not support External mode.

Enabled

The model application supports External mode.

IP address

The IP address of the LEGO MINDSTORMS EV3 brick.

Base rate task priority

The value in this parameter defines the priority of the base rate task. However, the changes that you make in this parameter do not result in any functionality differences on the LEGO MINDSTORMS EV3 brick.

Connection type

Choose the connection Simulink uses to download your model from the host computer to the NXT hardware.

Set up a USB or Bluetooth® connection before running the model on the NXT hardware.

Note The NXT hardware always uses a Bluetooth connection for External mode communications. The **Connection type** parameter does not affect External mode communications.

Settings

Default: `USB connection`

`USB connection`

Use a USB connection to download a model to the NXT hardware.

`Bluetooth connection`

Use a Bluetooth connection to download a model to the NXT hardware.

Device name

This parameter appears when the **Hardware board** parameter is set to `LEGO MINDSTORMS NXT` and the **Connection type** parameter is set to `Bluetooth connection`.

While you are setting up a Bluetooth connection, get the name of the NXT hardware in Windows® **Devices and Printers** and assign it to the **Device name** parameter. For example, if the Windows device name is “myNXT”, enter `myNXT` for **Device name** parameter in the Configuration Parameters dialog.

TCP/IP port (1024-65535)

This parameter appears when the **Hardware board** setting supports External mode.

Set the value of the TCP/IP port number, from 1024 to 65535. External mode uses this IP port for communications between the target hardware (hardware board) and host computer.

Settings

Default: `17725`

Enable overrun detection

Detect when a task overruns occurs in a model running on the hardware board. Indicate when an overrun has occurred.

A task overrun occurs if the hardware board is still performing one instance of a task when the next instance of that task is scheduled to begin.

The "Detect and Fix Task Overruns" topics listed in the following "See Also" subtopic describe how your hardware board indicates that an overrun has occurred.

You can fix overruns by decreasing the frequency with which tasks are scheduled to run, and by reducing the number or complexity of the tasks defined by your model.

If those solutions do not fix the task overrun condition, and you are using External mode, consider disabling External mode.

Settings

Default: Disabled

Disabled

Do not detect overruns.

Enabled

Detect overruns and generate an error message when an overrun occurs.

Device

This parameter appears when the **Hardware board** parameter is set to your device type, and **Show advanced settings** has been clicked.

Select the device you are using. The list includes any devices that are connected to your computer and turned on.

To see a device that was recently connected and turned on, click **Refresh**. Refreshing the parameters updates **Device**, **Host name**, and **Package name** parameter fields.

Settings

Default: `None`

Package name

This parameter appears when the **Hardware board** parameter is set to one of the Samsung Galaxy Android™ devices, and **Show advanced settings** has been clicked.

Update this value with a unique name. Refer to the Android Developer instructions the `package` attribute in `<manifest>`. The package name uniquely identifies the application you are creating, and determines the path names your application uses. To avoid conflicts with apps created by other developers, use a domain name that you own as the beginning of the package name. Reverse the order of the elements, like this: `com.mydomain.myappname`.

Warning Do not use com.example to publish applications (make the app publicly available).

Settings

Default: `com.example`

Digital output to set on overrun

This parameter appears when the **Hardware board** parameter is set to an Arduino hardware and the **Enable overrun detection** check box is selected.

Select the digital output pin the Arduino hardware uses to signal a task overrun.

Do not use a pin that is assigned to another block within the model.

Settings

Default: `13`

Enable communication between two NXT bricks

This parameter appears when the **Hardware board** parameter is set to `LEGO MINDSTORMS NXT`.

You can enable direct Bluetooth communication between two NXT bricks. Enabling this parameter makes the **Bluetooth mode** parameter appear.

Enabling the **Enable communication between two NXT bricks** parameter disables External mode for LEGO MINDSTORMS NXT hardware.

Settings

Default: Disabled

Disabled

Disable communication between two NXT bricks.

Enabled

Enable direct Bluetooth communication between two NXT bricks.

Bluetooth mode

This parameter appears when the **Hardware board** parameter is set to `LEGO MINDSTORMS NXT`.

If you enable the **Enable communication between two NXT bricks** parameter, configure the Bluetooth device on one NXT brick to be a Bluetooth master or slave.

This parameter only applies to Bluetooth communications between two NXT bricks. It does not apply to Bluetooth communications between the host computer and the NXT brick.

Selecting `Master` makes the **Bluetooth slave address** parameter appear.

Settings

Default: `Master`

`Master`

The Bluetooth device on the NXT brick operates as a master. Selecting this option enables the **Slave Bluetooth address** parameter.

`Slave`

The Bluetooth device on the NXT brick operates as a slave.

Slave Bluetooth address

This parameter appears when the **Hardware board** parameter is set to `LEGO MINDSTORMS NXT` and the **Bluetooth mode** parameter is set to `Master`.

Enter the address of the slave Bluetooth device on other NXT brick.

Host name

This parameter appears when the **Hardware board** requires a network connection to load the model or application to the hardware.

When you use the Support Package Installer to update the firmware on the board, the Support Package Installer automatically gets the value of the IP address from the board and applies it to this parameter.

If you swap boards, or change the IP address of the board, get the value of the new IP address and enter it here.

User name

This parameter appears when the **Hardware board** parameter is set to `BeagleBoard` or `Raspberry Pi`.

Enter the root user name for Linux running on the BeagleBoard or Raspberry Pi™ hardware.

When you use the Support Package Installer to update the BeagleBoard or Raspberry Pi firmware, the Support Package Installer automatically applies the value you entered there to this parameter.

Settings

BeagleBoard Default: `ubuntu`

Raspberry Pi Default: `pi`

Password

This parameter appears when the **Hardware board** parameter is set to `BeagleBoard` or `Raspberry Pi`.

Enter the root password for Linux running on the BeagleBoard or Raspberry Pi hardware.

When you use the Support Package Installer to update the firmware on the BeagleBoard or Raspberry Pi hardware, the Support Package Installer automatically applies the value you entered there to this parameter.

Settings

BeagleBoard Default: `temppwd`

Raspberry Pi Default: `raspberry`

Build directory

This parameter appears when the **Hardware board** parameter is set to `BeagleBoard` or `Raspberry Pi`.

Enter the build directory for Linux running on the BeagleBoard or Raspberry Pi hardware.

When you use the Support Package Installer to update the firmware on the BeagleBoard or Raspberry Pi hardware, the Support Package Installer automatically applies the value you entered there to this parameter.

Settings

BeagleBoard Default: `/home/ubuntu`

Raspberry Pi Default: `/home/pi`

Set host COM port

This parameter appears when the **Hardware board** parameter is set to `LEGO MINDSTORMS NXT`, `Arduino Mega 2560`, or `Arduino Uno`.

Automatically detect or manually set the COM port your host computer uses to communicate with the hardware board.

Warning Do not connect Arduino Uno and Arduino Mega 2560 to a RS-232 serial interface, commonly found on computers and equipment. RS-232 interfaces can use voltages greater than 5 Volts, which can damage your Arduino hardware.

Settings

Default: `Automatically`

`Automatically`
: Let the software determine which COM Port your host computer uses.

`Manually`
: Select this option to display the **COM port number** parameter.

COM port number

This parameter appears when the **Hardware board** parameter is set to `LEGO MINDSTORMS NXT`, `Arduino Mega 2560`, or `Arduino Uno`, and the **Set host COM port** parameter is set to `Manually`.

Manually set the number of the COM Port the host computer uses to communicate with the hardware board, and then enter it here.

Warning Do not connect Arduino Uno and Arduino Mega 2560 to a RS-232 serial interface, commonly found on computers and equipment. RS-232 interfaces can use voltages greater than 5 Volts, which can damage your Arduino hardware.

Settings

Default: `0`

Analog input reference voltage

This parameter appears when the **Hardware board** parameter is set to `Arduino Mega 2560` or `Arduino Uno`.

Set the reference voltage used to measure inputs to the ANALOG IN pins.

Warning Only connect an external power source to AREF while this parameter is set to `External`. Connecting an external power source to AREF while this parameter is set to any other option exposes the internal voltage references to the external voltage. This voltage difference can damage your hardware.

Do not connect Arduino Uno and Arduino Mega 2560 to voltages greater than 5 Volts.

Do not connect Arduino Due to voltages greater than 3.3 Volts.

Voltages greater than the specified limits can damage your Arduino hardware.

Settings

Default: `Default`

`Default`

Use the default operating voltage of the board. For Arduino Uno and Arduino Mega 2560 the operating voltage is 5 Volts.

`Internal (1.1 V)`

Valid for Arduino Mega 2560 only: Use the internal 1.1 Volt reference.

`Internal (2.56 V)`

Valid for Arduino Mega 2560 only: Use the internal 2.56 Volt reference.

`External`

On the Arduino Uno, Arduino Nano and Arduino Mega 2560, use an external 0-5 volt power supply connected to the AREF pin. This voltage should match the voltage of the power supply connected to the Arduino hardware. If your application requires low-noise measurements, use this option with a filtered power supply.

Serial 0 baud rate, Serial 1 baud rate, Serial 2 baud rate, Serial 3 baud rate

Set the baud rate of the serial port on the Arduino hardware.

If you set **Set host COM port** to `Manually`, then set **Serial 0 baud rate** as described in the "Set the COM Port and Baud Rate Manually" topic.

For information on serial ports for different Arduino boards, see "Pin Mapping on Arduino Blocks" (Simulink Support Package for Arduino Hardware).

Settings

Default: `9600`

`300`, `1200`, `2400`, `4800`, `9600`, `14400`, `19200`, `28800`, `38400`, `57600`, `76800`, `115200`, `128000`, `500000`, `1000000`

IP address

Enter the IP address of the Arduino Ethernet shield.

MAC address

Enter the machine address of the Arduino Ethernet shield.

IP address

Enter the IP address of the Arduino WiFi shield.

Service set identifier (SSID)

Enter the SSID of your network. An SSID is a unique ID consisting of 32 characters and is used for naming wireless networks. An SSID ensures that the data you send over the network reaches the correct destination.

WiFi encryption

The WiFi encryption that is used in the network you connect to.

Settings

Default: `None`

`None`

Select this option when you connect to a network that is not WiFi encrypted.

`WPA`

Select this option when you connect to a network that uses WPA WiFi encryption.

`WEP`

Select this option when you connect to a network that uses WEP WiFi encryption.

WPA password

This parameter appears only when you select `WPA` option in the **WiFi encryption** parameter. Enter the WPA password of the network.

WEP key

This parameter appears only when you select `WEP` option in the **WiFi encryption** parameter. Enter the WEP key of the network.

WEP key index

This parameter appears only when you select `WEP` option in the **WiFi encryption** parameter. Enter the WEP key index of the WEP key.

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 1-7

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`
- `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

Altera options:

- SoC (ARM CortexA)

Analog Devices options:

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

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`
- `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	✓	□
K5/K6/ Athlon	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
x86-32 (Windows 32)	8	16	32	32	64	32	32	32	32	Char	Float	Little Endian	Zero	✓	□
x86-64 (Linux 64)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
x86-64 (macOS)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□

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 (Windows 64)	8	16	32	32	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
ARM Compatible															
ARM 7/8/9/10	8	16	32	32	64	32	32	32	32	Long	Float	Little Endian	Zero	✓	□
ARM 11	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
ARM Cortex	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Altera															
SoC (ARM Cortex A)	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
Analog Devices															
ADSP-CM40x(ARM Cortex-M)	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□

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	✓	□
SHARC	32	32	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	□
TigerSHARC	32	32	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Atmel															
AVR	8	16	16	32	64	8	16	16	16	Char	None	Little Endian	Zero	✓	□
AVR (32-bit)	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
AVR (8-bit)	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
Freescale															
32-bit PowerPC	8	16	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	□

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				
68332	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	□
68HC08	8	16	16	32	64	8	8	16	8	Char	None	Big Endian	Zero	✓	□
68HC11	8	16	16	32	64	8	8	16	16	Char	None	Big Endian	Zero	✓	□
ColdFire	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	□
DSP563xx (16-bit mode)	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
DSP5685x	8	16	16	32	64	16	16	16	16	Char	Float	Little Endian	Zero	✓	□
HC(S)12	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	□

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	✓	□
MPC85xx	8	16	32	32	64	32	32	32	32	Long	Double	Big Endian	Zero	✓	□
S08	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	□
S12x	8	16	16	32	64	16	16	16	16	Char	None	Big Endian	Zero	✓	□
StarCore	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
Infineon															

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				
C16x, XC16x	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
TriCore	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
Intel															
x86-32 (Windows 32)	8	16	32	32	64	32	32	32	32	Char	Float	Little Endian	Zero	✓	□
x86-64 (Linux 64)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
x86-64 (macOS)	8	16	32	64	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
x86-64 (Windows 64)	8	16	32	32	64	64	64	64	64	Char	Float	Little Endian	Zero	✓	□
Microchip															
PIC18	8	16	16	32	64	8	8	24	24	Char	None	Little Endian	Zero	✓	□

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				
dsPIC	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
NXP															
Cortex-M0/M0+	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Cortex-M3	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Cortex-M4	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
Renesas															
M16C	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
M32C	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
R8C/Tiny	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□

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				
RH850	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
RL78	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
SH-2/3/4	8	16	32	32	64	32	32	32	32	Char	None	Big Endian	Zero	✓	□
V850	8	16	32	32	64	32	32	32	32	Char	None	Little Endian	Zero	✓	□
STMicroelectronics															
ST10/ Super10	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
Texas Instruments															
C2000	16	16	16	32	64	16	32	16	16	Int	None	Little Endian	Zero	✓	□
C5000	16	16	16	32	64	16	16	16	16	Int	None	Big Endian	Zero	✓	□

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	✓	□
MSP430	8	16	16	32	64	16	16	16	16	Char	None	Little Endian	Zero	✓	□
Stellaris Cortex—M3	8	16	32	32	6	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
TMS470	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
TMS570 Cortex—R4	8	16	32	32	64	32	32	32	32	Long	Double	Little Endian	Zero	✓	□
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 More Device Vendor and Device Type Values" (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 1-10
- Specifying Test Hardware Characteristics (Simulink Coder)
- Hardware Implementation Options (Simulink Coder)
- "Hardware Implementation Pane" on page 1-7

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 1-7

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 1-7

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 1-7

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. (The value 64 is selected by default if you run MATLAB software on a 64-bit host computer and select the MATLAB host as the test hardware — that is, `TargetHWDeviceType` equals `'Generic->MATLAB Host Computer'`.)

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 1-7

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
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 1-7

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 1-7

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 1-7

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: 32

Minimum: 8

Maximum: 64

Enter a value between 8 and 64. (The value 64 is selected by default if you run MATLAB software on a 64-bit host computer and select the MATLAB host as the test hardware — that is, `TargetHWDeviceType` equals `'Generic->MATLAB Host Computer'`.)

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: `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 1-7

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: Device-specific value (see Dependencies)

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: device dependent

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact

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 1-7

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: Device-specific value (see Dependencies)

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: device dependent

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact

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 1-7
- “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: Device-specific value (see Dependencies)

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: device dependent

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact

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 1-7
- “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 2-52
- “Hardware Implementation Pane” on page 1-7

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: `None`

`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: `'None'`

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 1-7

Byte ordering

Description

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

Category: Hardware Implementation

Settings

Default: `Unspecified`

`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: `TargetEndianess`
Value: `'Unspecified' | 'LittleEndian' | 'BigEndian'`
Default: `'Unspecified'`

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 1-7

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: `Undefined`

`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: `'Undefined'`

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 1-7

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
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 1-7

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 2-27
- “Hardware Implementation Pane” on page 1-7

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.

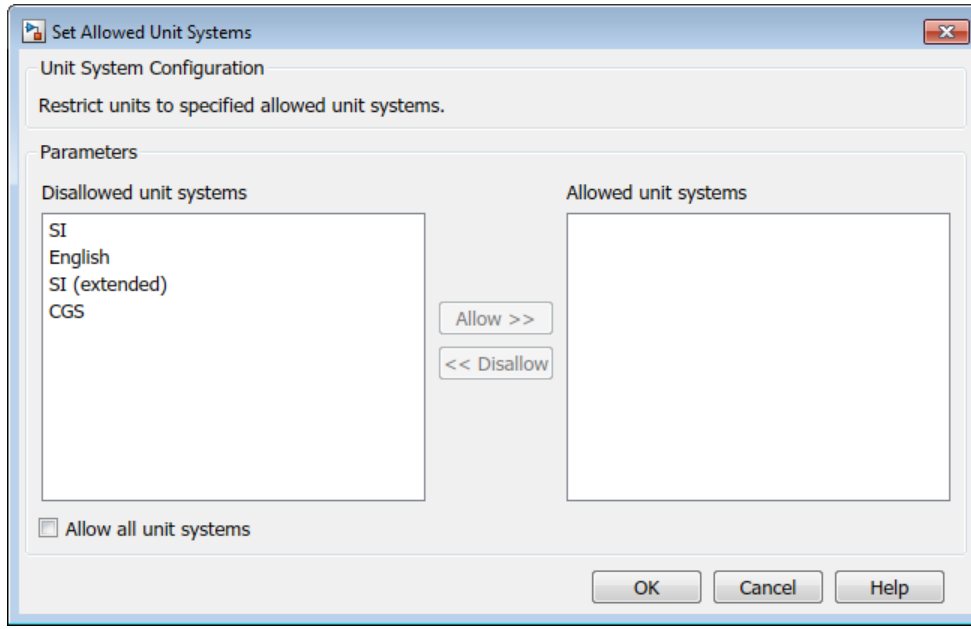
`SI (extended)`
: International System of Units (extended).

`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 12-2

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 12-2

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 12-2

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 fileds:

          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.

Command-Line Information

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”

Enable live streaming of selected signals to Simulation Data Inspector

Description

Specify whether to send signals marked for streaming  to the Simulation Data Inspector during simulation.

Category: Data Import/Export

Settings

Default: On

 On

Send signals marked for streaming to the Simulation Data Inspector during simulation. This setting turns on the streaming state on the **Simulation Data Inspector** button on the Simulink Editor toolbar. During simulation, the simulation data appears in the **Runs** pane in the Simulation Data Inspector. To view a streaming signal during simulation, open the Simulation Data Inspector, and select the signal check box in the **Runs** pane.

 Off

Do not send signals marked for streaming to the Simulation Data Inspector during simulation. This setting turns off the live streaming state on the **Simulation Data Inspector** button on the Simulink Editor toolbar.

Tip

To open the Simulation Data Inspector, on the Simulink Editor toolbar, click the **Simulation Data Inspector** button arrow and select `Simulation Data Inspector`.

Command-Line Information

Parameter: `VisualizeSimOutput`
Value: `'on' | 'off'`
Default: `'on'`

Recommended Settings

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

See Also

Related Examples

- “Iterate Model Design with the Simulation Data Inspector”
- “Inspect Simulation Data”
- “Organize Your Simulation Data Inspector Workspace”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Array bounds exceeded

Description

Enable live streaming of selected signals to Simulation Data Inspector

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 “Error Handling” 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, select **Analysis > Model Advisor**.
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 9-2

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 9-2

Check runtime output of execution context

Description

Specify whether to display a warning if Simulink software detects potential output differences from previous releases.

Category: Diagnostics

Settings

Default: Off

 On

Displays a warning if Simulink software detects potential output differences from previous releases.

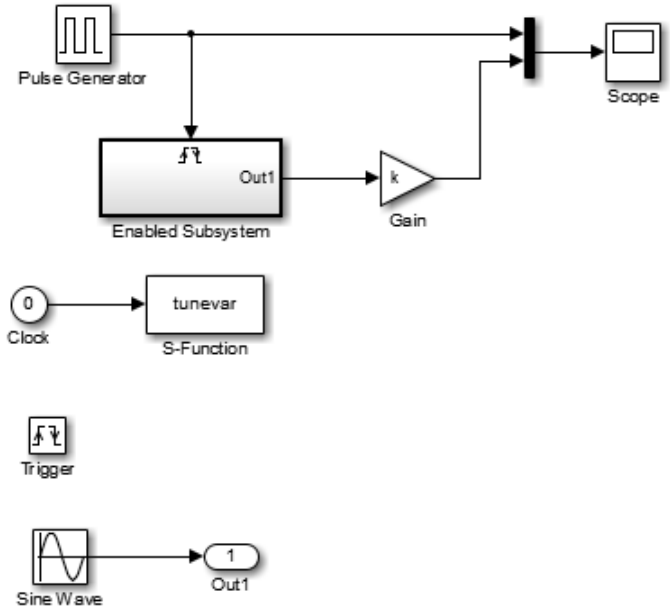
 Off

Does not display a warning.

Tips

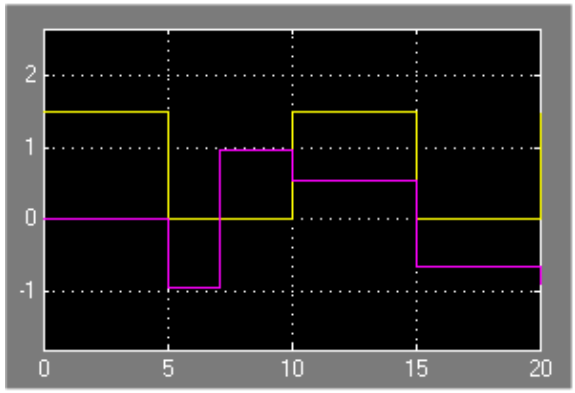
- This diagnostic is triggered if the model contains a block that meets the following conditions:
  - The block has a tunable parameter.
  - The block is connected to an output of a conditionally executed subsystem.
  - The block inherits its execution context from that subsystem.
  - The Outport to which it is connected has an undefined initial condition, i.e., the Outport block's **Initial output** parameter is set to [].
- Models with blocks that meet these criteria can produce results when the parameter is tuned in the current release that differ from results produced in Release 13 or earlier releases.

  Consider for example the following model.

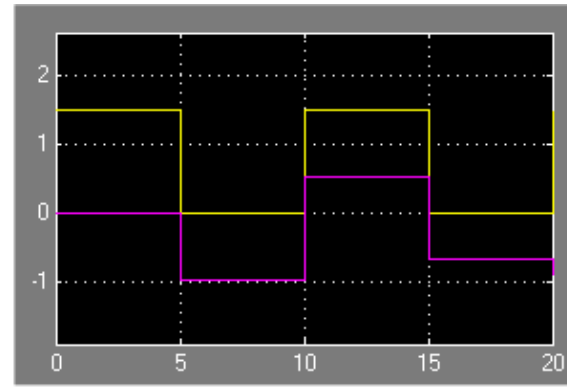


In this model, the `tunevar` S-function changes the value of the Gain block's `k` parameter and updates the diagram at simulation time 7 (i.e., it simulates tuning the parameter).

The following figure compares the superimposed output of the model's Pulse Generator block and its Gain block in Release 13 and the current release.



Release 13



Current Release

Note that the output of the Gain block changes at time `7` in Release 13 but does not change in the current release. This is because in Release 13, the Gain block belongs to the execution context of the root system and hence executes at every time step whereas in the current release, the Gain block belongs to the execution context of the triggered subsystem and hence executes only when the triggered subsystem executes, i.e., at times `5`, `10`, `15`, and `20`.

Dependency

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

Command-Line Information

Parameter: `CheckExecutionContextRuntimeOutputMsg`
Value: `'on' | 'off'`
Default: `'on'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact

Application	Setting
Efficiency	No impact
Safety precaution	On

See Also

Related Examples

- Diagnosing Simulation Errors
- “Underspecified initialization detection” on page 2-79
- “Model Configuration Parameters: Diagnostics” on page 12-2

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 Outport 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 Outport block with an undefined initial condition.

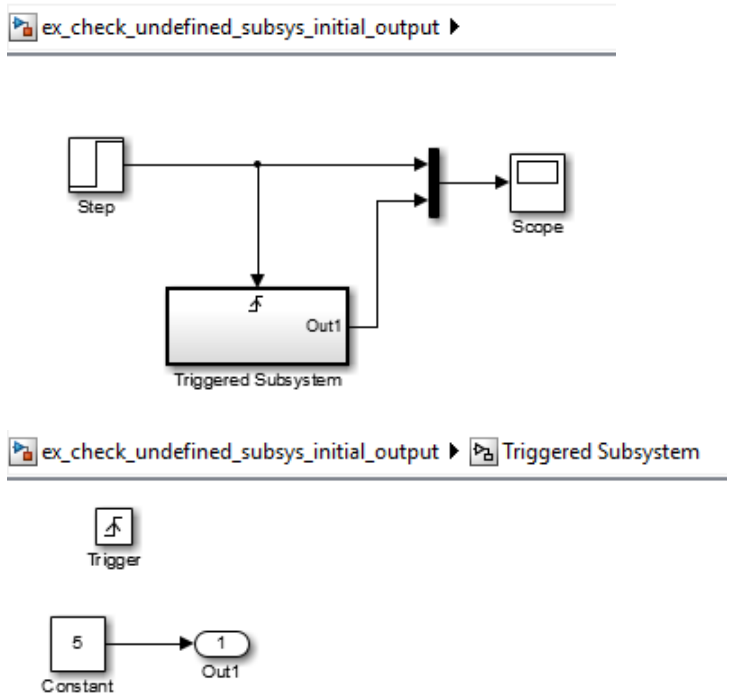
 Off

Does not display a warning.

Tips

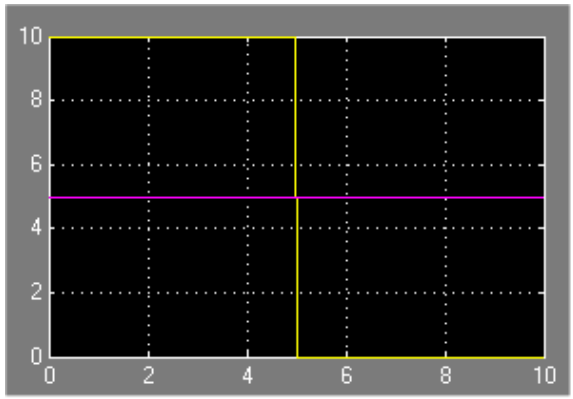
- This situation occurs when a block with a specified initial condition, such as a Constant, Initial Condition, or Delay block, drives an Outport block with an undefined initial condition (**Initial output** parameter is set to []).
- Models with such subsystems can produce initial results (i.e., before 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.

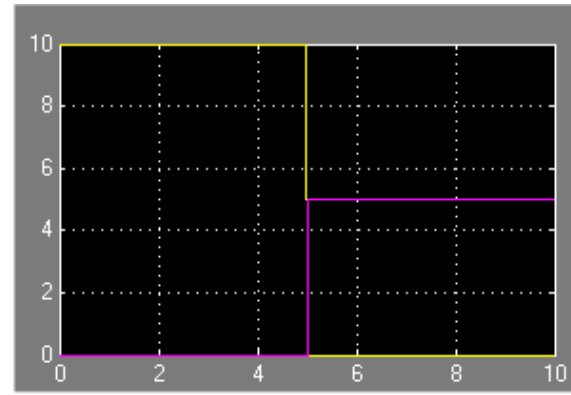


This model does not define the initial condition of the triggered subsystem's output port.

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

Application	Setting
Safety precaution	On

See Also

Related Examples

- Diagnosing Simulation Errors
- “Conditionally Executed Subsystems”
- “Underspecified initialization detection” on page 2-79
- “Model Configuration Parameters: Diagnostics” on page 12-2

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 software terminates the simulation and displays an error message.

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.
- If **Underspecified initialization detection** is set to `Simplified`, this parameter is disabled, and Simulink software automatically uses the strictest setting (`error`) for this diagnostic. Multiple driving blocks executing at the same time step always result in an error.

Dependency

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

Command-Line Information

Parameter: `MergeDetectMultiDrivingBlocksExec`
Value: `'none' | 'warning' | 'error'`
Default: `'error'`

Recommended Settings

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

See Also

Merge

Related Examples

- Diagnosing Simulation Errors
- “Check usage of Merge blocks”
- “Underspecified initialization detection” on page 2-79
- Data Validity Diagnostics on page 9-2

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 8-17 to `error` on the Connectivity Diagnostics pane.

Dependencies

Selecting `Classic` enables the following parameters:

- **Detect multiple driving blocks executing at the same time step**
- **Check undefined subsystem initial output**
- **Check runtime output of execution context**

Selecting `Simplified` disables these parameters, and automatically sets **Detect multiple driving blocks executing at the same time step** to `error`.

Command-Line Information

Parameter: `UnderspecifiedInitializationDetection`
Value: `'Classic' | 'Simplified'`
Default: `'Classic'`

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”
- “Simplified Initialization Mode”
- “Classic Initialization Mode”
- “Conditional Subsystem Output Values When Disabled”
- Diagnosing Simulation Errors
- Data Validity Diagnostics on page 9-2

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: `'warning'`

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 12-2

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

Application	Setting
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
- “Model Parameters”
- Solver Diagnostics on page 12-2

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 in formation 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

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

See Also

Related Examples

- “Identify parameterized library links”
- Solver Diagnostics on page 12-2

InitInArrayFormatMsg

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 3-7
- “State Information”
- “Dataset Conversion for Logged Data”
- “Model Configuration Parameters: Diagnostics” on page 12-2

Remove code from floating-point to integer conversions with saturation that maps NaN to zero

Description

Remove code that handles floating-point to integer conversion results for `NaN` values.

Category: Optimization

Settings

Default: On

On

Removes code when mapping from `NaN` to integer zero occurs. Select this check box if code efficiency is critical to your application and the following conditions are true for at least one block in the model:

- Computing outputs or parameters of a block involves converting floating-point data to integer or fixed-point data.
- The **Saturate on integer overflow** check box is selected in the Block Parameters dialog box.

Caution Execution of generated code might not produce the same results as simulation.

Off

Results for simulation and execution of generated code match when mapping from `NaN` to integer zero occurs. The generated code is larger than when you select this check box.

Tips

- Selecting this check box reduces the size and increases the speed of the generated code at the cost of producing results that do not match simulation in the case of `NaN` values.
- Selecting this check box affects code generation results only for `NaN` values and cannot cause code generation results to differ from simulation results for any other values.

Dependencies

- This parameter requires a Simulink Coder license.
- For ERT-based targets, this parameter is enabled when you select the **floating-point numbers** and **non-finite numbers** check boxes in the **Code Generation > Interface** pane.

Command-Line Information

Parameter: `EfficientMapNaN2IntZero`
Value: `'on' | 'off'`
Default: `'on'`

Recommended Settings

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

See Also

Related Examples

- “Remove Code That Maps NaN to Integer Zero” (Simulink Coder)
- “Optimization Pane: General” on page 4-2

Compiler optimization level

Description

Sets the degree of optimization used by the compiler when generating code for acceleration.

Category: Optimization

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

Application	Setting
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”
- “Optimization Pane: General” on page 4-2

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: Optimization

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”
- “Optimization Pane: General” on page 4-2

Implement logic signals as Boolean data (vs. double)

Description

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

Category: Optimization

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)

- “Optimization Pane: General” on page 4-2

Block reduction

Description

Reduce execution time by collapsing or removing groups of blocks.

Category: Optimization

Settings

Default: On

 **On**

Simulink software searches for and reduces the following 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

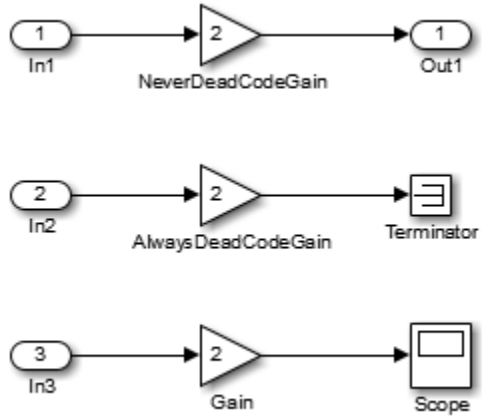
- When you select **Block reduction**, Simulink software collapses certain groups of blocks into a single, more efficient block, or removes them entirely. This results in faster execution during model simulation and in generated code.
- Block reduction does not change the appearance of the source model.
- 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.

- 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.

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 recommendation

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)
- “Optimization Pane: General” on page 4-2

Conditional input branch execution

Description

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

Category: Optimization

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 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.
- 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)
- “Conditional Subsystems”
- “Performance” (Simulink Coder)
- “Optimization Pane: General” on page 4-2

Use memset to initialize floats and doubles to 0.0

Description

Specify whether to generate code that explicitly initializes floating-point data to 0.0.

Category: Optimization

Settings

Default: On (GUI), `'off'` (command-line)

 On

Uses `memset` to clear internal storage for floating-point data to integer bit pattern 0 (all bits 0), regardless of type. If your compiler and target CPU both represent floating-point zero with the integer bit pattern `0`, use this parameter to gain execution and ROM efficiency.

 Off

Generates code to explicitly initialize storage for data of types `float` and `double` to 0.0. The resulting code is slightly less efficient than code generated when you select the option.

You should not select this option if you need to ensure that memory allocated for C MEX S-function wrappers is initialized to zero.

Dependency

This parameter requires a Simulink Coder license.

Command-Line Information

Parameter: `InitFltsAndDblsToZero`
Value: `'on' | 'off'`
Default: `'off'`

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	On (GUI), `'off'` (command-line) (execution, ROM), No impact (RAM)
Safety precaution	No impact

See Also

Related Examples

- "Optimize Generated Code Using memset Function" (Simulink Coder)
- "Optimization Pane: General" on page 4-2

Signal storage reuse

Description

Reuse signal memory.

Category: Optimization

Settings

Default: On

 **On**

Simulink software reuses memory buffers allocated to store block input and output signals, reducing the memory requirement of your real-time program.

 **Off**

Simulink software allocates a separate memory buffer for each block's outputs. This makes all block outputs global and unique, which in many cases significantly increases RAM and ROM usage.

Tips

- This option applies only to signals with storage class `Auto`.
- Signal storage reuse can occur only among signals that have the same data type.
- Clearing this option can substantially increase the amount of memory required to simulate large models.
- Clear this option if you need to:
  - Debug a C-MEX S-function
  - Use a Floating Scope or a Floating Scope block with the **Floating display** option selected to inspect signals in a model that you are debugging
- Simulink software opens an error dialog if **Signal storage reuse** is enabled and you attempt to use a Floating Scope or floating Display block to display a signal whose buffer has been reused.

Dependencies

This parameter enables:

- **“Enable local block outputs” on page 2-109**
- **“Reuse local block outputs” on page 2-111**
- **“Eliminate superfluous local variables (Expression folding)” on page 2-113**

If you have an Embedded Coder license, this parameter enables:

- **“Optimize global data access” on page 2-117**
- **“Perform inplace updates for Bus Assignment blocks” on page 2-142**
- **“Reuse global block outputs” on page 2-115**
- **“Optimize block operation order in the generated code” on page 2-146**
- **“Reuse buffers for Data Store Read and Data Store Write blocks” on page 2-144**

Command-Line Information

Parameter:`OptimizeBlockIOStorage`
Value: `'on' | 'off'`
Default: `'on'`

Recommended Settings

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

See Also

Related Examples

- “Minimize Computations and Storage for Intermediate Results at Block Outputs” (Simulink Coder)
- “Performance” (Simulink Coder)
- “Optimization Pane: Signals and Parameters” on page 5-2

Enable local block outputs

Description

Specify whether block signals are declared locally or globally.

Category: Optimization

Settings

Default: On

 **On**

Block signals are declared locally in functions.

 **Off**

Block signals are declared globally.

Tips

- If it is not possible to declare an output as a local variable, the generated code declares the output as a global variable.
- If you are constrained by limited stack space, you can turn **Enable local block outputs** off and still benefit from memory reuse.

Dependencies

- This parameter requires a Simulink Coder license.
- This parameter is enabled by **Signal storage reuse**.

Command-Line Information

Parameter: `LocalBlockOutputs`
Value: `'on' | 'off'`
Default: `'on'`

Recommended Settings

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

See Also

Related Examples

- “Enable and Reuse Local Block Outputs in Generated Code” (Simulink Coder)
- “Performance” (Simulink Coder)
- “Optimization Pane: Signals and Parameters” on page 5-2

Reuse local block outputs

Description

Specify whether Simulink Coder software reuses signal memory.

Category: Optimization

Settings

Default: On

On

- Simulink Coder software reuses signal memory whenever possible, reducing stack size where signals are being buffered in local variables.
- Selecting this parameter trades code traceability for code efficiency.

Off

Signals are stored in unique locations.

Dependencies

This parameter:

- Is enabled by **Signal storage reuse**.
- Requires a Simulink Coder license.

Command-Line Information

Parameter: `BufferReuse`
Value: `'on' | 'off'`
Default: `'on'`

Recommended Settings

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

See Also

Related Examples

- “Enable and Reuse Local Block Outputs in Generated Code” (Simulink Coder)
- “Performance” (Simulink Coder)
- “Optimization Pane: Signals and Parameters” on page 5-2

Eliminate superfluous local variables (Expression folding)

Description

Collapse block computations into single expressions.

Category: Optimization

Settings

Default: On

On

- Enables expression folding.
- Eliminates local variables, incorporating the information into the main code statement.
- Improves code readability and efficiency.

Off

Disables expression folding.

Dependencies

- This parameter requires a Simulink Coder license.
- This parameter is enabled by **Signal storage reuse**.

Command-Line Information

Parameter: `ExpressionFolding`
Value: `'on' | 'off'`
Default: `'on'`

Recommended Settings

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

See Also

Related Examples

- “Minimize Computations and Storage for Intermediate Results at Block Outputs” (Simulink Coder)
- “Performance” (Simulink Coder)
- “Optimization Pane: Signals and Parameters” on page 5-2

Reuse global block outputs

Description

Reuse global memory for block outputs.

Category: Optimization

Settings

Default: On

On

- Software reuses signal memory whenever possible, reducing global variable use.
- Selecting this parameter trades code traceability for code efficiency.

Off

Signals are stored in unique locations.

Dependencies

This parameter:

- Is enabled by **"Signal storage reuse" on page 2-106**.
- Requires an Embedded Coder license.
- Appears only for ERT-based targets.

Command-Line Information

Parameter: `GlobalBufferReuse`
Value: `'on' | 'off'`
Default: `'on'`

Recommended Settings

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

See Also

Related Examples

- “Reuse Global Block Outputs in the Generated Code” (Embedded Coder)
- “Performance” (Embedded Coder)
- “Optimization Pane: Signals and Parameters” on page 5-2

Optimize global data access

Description

Select global variable optimization.

Category: Optimization

Settings

Default: `None`

`None`

Use default optimizations.

`Use global to hold temporary results`

Maximize use of global variables.

`Minimize global data access`

Minimize use of global variables by using local variables to hold intermediate values.

Dependencies

- This parameter is enabled by **"Signal storage reuse" on page 2-106**.
- This parameter requires an Embedded Coder license.
- Appears only for ERT-based targets.

Command-Line Information

Parameter: `GlobalVariableUsage`
Value: `'None' | 'Use global to hold temporary results' | 'Minimize global data access'`
Default: `'None'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	`'Use global to hold temporary results'` (RAM), `'Minimize global data access'` (ROM)
Safety precaution	No impact

See Also

Related Examples

- “Optimize Global Variable Usage” (Embedded Coder)
- “Performance” (Embedded Coder)
- “Optimization Pane: Signals and Parameters” on page 5-2

Simplify array indexing

Description

Replace multiply operations in array indices when accessing arrays in a loop.

Category: Optimization

Settings

Default: Off

 On

In array indices, replace multiply operations with add operations when accessing arrays in a loop in the generated code. When the original signal is multidimensional, the Embedded Coder generates one-dimensional arrays, resulting in multiply operations in the array indices. Using this setting eliminates costly multiply operations when accessing arrays in a loop in the C/C++ program. This optimization (commonly referred to as strength reduction) is particularly useful if the C/C++ compiler on the target platform does not have similar functionality. No appearance of multiply operations in the C/C++ program does not imply that the C/C++ compiler does not generate multiply instructions.

 Off

Leave multiply operations in array indices when accessing arrays in a loop.

Dependencies

This parameter:

- Requires a Embedded Coder license to generate code.
- Appears only for ERT-based targets.

Command-Line Information

Parameter: `StrengthReduction`
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

- “Simplify Multiply Operations in Array Indexing” (Embedded Coder)
- “Performance” (Embedded Coder)
- “Optimization Pane: Signals and Parameters” on page 5-2

Ensure responsiveness

Description

Enables responsiveness checks in code generated for MATLAB Function blocks.

Category: Simulation Target

Settings

Default: On

 On

Enables periodic checks for Ctrl+C breaks in code generated for MATLAB Function blocks. Also allows graphics refreshing.

 Off

Disables periodic checks for Ctrl+C breaks in code generated for MATLAB Function blocks. 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 16-2

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.

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 16-2

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. 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 16-2

Dynamic memory allocation in MATLAB Function blocks

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 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 Function blocks** 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 16-2

Dynamic memory allocation threshold in MATLAB Function blocks

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 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 Function blocks 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 16-2

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

Application	Setting
Efficiency	Off
Safety precaution	No impact

See Also

Related Examples

- “Speed Up Simulation” (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 16-2

Ensure memory integrity

Description

Detects violations of memory integrity while building MATLAB Function blocks and stops simulation with a diagnostic.

Category: Simulation Target

Settings

Default: On

 On

Detect violations of memory integrity while building MATLAB Function blocks and stops simulation with a diagnostic message.

 Off

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

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.
- Only disable these checks if you are sure that all array bounds and dimension checking is unnecessary.

Command-Line Information

Parameter: `SimIntegrity`
Value: `'on' | 'off'`
Default: `'on'`

Recommended Settings

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

See Also

Related Examples

- “Control Run-Time Checks”
- “Model Configuration Parameters: Simulation Target” on page 16-2

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**, under **Simulation Target > Custom Code > Header file**.

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 16-2

Simulation target build mode

Description

Specifies how you build the simulation target for a model that contains MATLAB Function blocks, Stateflow charts, or Truth Table blocks.

Category: Simulation Target

Settings

Default: `Incremental build`

`Incremental build`

This option rebuilds only those portions of the target that you changed since the last build.

`Rebuild all (including libraries)`

This option rebuilds the target, including libraries, from scratch.

`Make without generating code`

This option invokes the make process without generating code.

`Clean all (delete generated code/executables)`

This option deletes both generated source code and executable files.

`Clean objects (delete executables only)`

This option deletes only executable files.

Tips

- The default `Incremental build` is a good choice for most models. This action takes place whenever you simulate your model.
- Set `Rebuild all (including libraries)` if you have changed your compiler or updated your object files since the last simulation. For example, use this option to rebuild the simulation target to include custom code changes.
- Set `Make without generating code` when you have custom source files that you must recompile in an incremental build mechanism that does not detect changes in custom code files.

Command-Line Information

Parameter: `SimBuildMode`
Value: `'sf_incremental_build' | 'sf_nonincremental_build' | 'sf_make' | 'sf_make_clean' | 'sf_make_clean_objects'`
Default: `'sf_incremental_build'`

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 16-2

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: Off

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 is available only for library models that contain MATLAB Function blocks, Stateflow charts, or Truth Table blocks. To access this parameter, in the MATLAB Function Block Editor, select **Tools** > **Open Simulation Target**.

Command-Line Information

Parameter: `SimUseLocalCustomCode`
Value: `'on' | 'off'`
Default: `'off'`

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
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 16-2

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 Dimension Variants for Array Sizes in Generated Code” (Embedded Coder)
- “Model Configuration Parameters: Diagnostics” on page 12-2

Perform inplace updates for Bus Assignment blocks

Description

Reuse the input and output variables of Bus Assignment blocks if possible.

Category: Optimization

Settings

Default: On

 On

Embedded Coder reuses the input and output variables of Bus Assignment blocks if possible. Reusing these variables reduces data copies, conserves RAM consumption and increases code execution speed.

 Off

Embedded Coder does not reuse the input and output variables of Bus Assignment blocks.

Dependency

- The parameter **Signal Storage Reuse** enables this parameter.
- This parameter requires an Embedded Coder license.
- This parameter appears only for ERT-based targets.

Command-Line Information

Parameter: `BusAssignmentInplaceUpdate`
Value: `'on' | 'off'`
Default: `'on'`

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	No impact
Efficiency	On
Safety precaution	No recommendation

See Also

Related Examples

- “Optimization Pane: Signals and Parameters” on page 5-2
- “Data copy reduction for Bus Assignment block” (Embedded Coder)

Reuse buffers for Data Store Read and Data Store Write blocks

Description

Remove temporary buffers for Data Store Read and Data Store Write blocks. Use the Data Store Memory block directly if possible.

Category: Optimization

Settings

Default: On

 On

Embedded Coder reads directly from the Data Store Memory block and writes directly to the Data Store Memory block, if possible. Using the Data Store Memory block directly eliminates data copies in the generated code, conserving RAM consumption and increasing code execution speed.

 Off

Embedded Coder inserts buffers in the generated code for Data Store Read and Data Store Write blocks.

Dependency

- The parameter **Signal Storage Reuse** enables this parameter.
- This parameter requires an Embedded Coder license.
- This parameter appears only for ERT-based targets.

Command-Line Information

Parameter: `OptimizeDataStoreBuffers`
Value: `'on' | 'off'`
Default: `'on'`

Recommended Settings

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

See Also

Related Examples

- “Data copy reduction for Data Store Read and Data Store Write blocks” (Embedded Coder)
- “Optimization Pane: Signals and Parameters” on page 5-2

Optimize block operation order in the generated code

Description

Reorder block operations in the generated code for improved code execution speed.

Category: Optimization

Settings

Default: Off

☑ Off

Embedded Coder does not reorder block operation order in the generated code to create additional instances of buffer reuse.

☐ Improved Code Execution Speed

Embedded Coder changes the block operation order in the generated code so that more instances of buffer reuse can occur. Reusing buffers conserves RAM and ROM consumption and improves code execution speed.

Dependency

- The parameter **Signal Storage Reuse** enables this parameter.
- This parameter requires an Embedded Coder license.
- This parameter appears only for ERT-based targets.

Command-Line Information

Parameter: `OptimizeBlockOrder`
Value: `'Speed' | 'off'`
Default: `'off'`

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
Traceability	No impact
Efficiency	Improved Code Execution Speed
Safety precaution	No recommendation

See Also

Related Examples

- “Remove Data Copies by Reordering Block Operations in the Generated Code” (Embedded Coder)
- “Improve Execution Efficiency by Reordering Block Operations in the Generated Code” (Embedded Coder)
- “Optimization Pane: Signals and Parameters” on page 5-2

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 17-2

Reuse buffers of different sizes and dimensions

Reduce memory consumption by reusing buffers to store data of different sizes and dimensions.

Settings

Default: Off

 On

The code generator tries to reuse the same buffers to store data of different sizes and dimensions. This optimization conserves RAM and ROM consumption.

 Off

The code generator reuses buffers only if they have the same size and shape as the data.

Dependencies

- This parameter appears only for ERT-based targets.
- This parameter requires an Embedded Coder license when generating code.
- This parameter is enabled by “Signal storage reuse” on page 2-106.

Tips

- If your model contains a reusable custom storage class to specify reuse on signals that have different sizes and shapes, you must select the **Reuse buffers of different sizes and dimensions** parameter or remove the specification. Otherwise, during simulation, the model errors out.
- The code generator does not replace a buffer with a lower priority buffer that has a smaller size.
- The code generator does not reuse buffers that have different sizes and symbolic dimensions.

Command-Line Information

Parameter: `DifferentSizesBufferReuse`

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

Recommended Settings

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

See Also

Related Examples

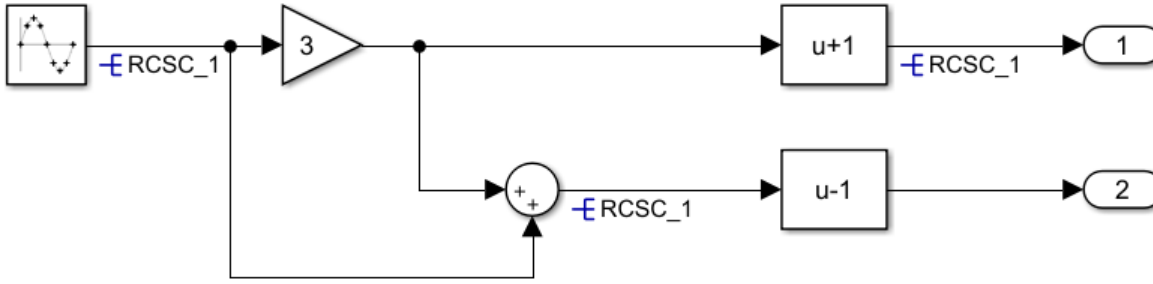
- “Optimization Pane: Signals and Parameters” on page 5-2
- “Optimization Pane: General” on page 4-2

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

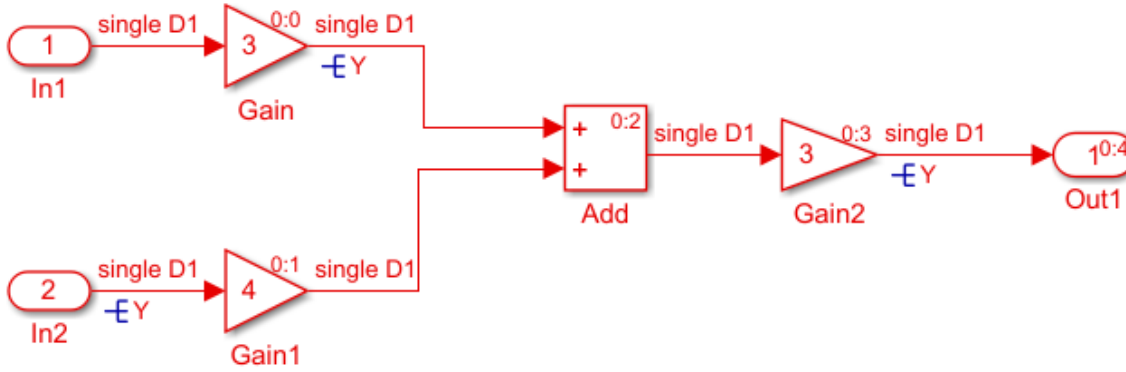
- “Specify Buffer Reuse by Using Simulink.Signal Objects” (Embedded Coder)
- Data Validity Diagnostics on page 9-2
- “Optimization Pane: General” on page 4-2

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. 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.

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

- “Specify Buffer Reuse by Using Simulink.Signal Objects” (Embedded Coder)
- Data Validity Diagnostics on page 9-2
- “Optimization Pane: General” on page 4-2

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 import 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

- "What is an Artificial Algebraic Loop?"
- "Model Configuration Parameters: Diagnostics" on page 12-2

3

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).

Parameter	Description
"Input" on page 3-5	Loads input data from a workspace before the simulation begins.
"Initial state" on page 3-7	Loads the model initial states from a workspace before simulation begins.
"Time" on page 3-10	Saves simulation time data to the specified variable during simulation.
"States" on page 3-12	Saves state data to the specified MATLAB variable during a simulation.
"Output" on page 3-14	Saves signal data to the specified MATLAB variable during simulation.
"Final states" on page 3-17	Saves the logged states of the model at the end of a simulation to the specified MATLAB variable.
"Format" on page 3-19	Select the data format for saving states, output, and final states data.
"Limit data points" on page 3-22	Limit the number of data points to export to the MATLAB workspace.
"Decimation" on page 3-24	Specify that Simulink software output only every `N` points, where `N` is the specified decimation factor.
"Save complete SimState in final state" on page 3-26	At the end of a simulation, Simulink saves the complete set of states of the model, including logged states, to the specified MATLAB variable.
"Signal logging" on page 3-28	Globally enable or disable signal logging to the workspace for this model.

Parameter	Description
“Data stores” on page 3-31	Globally enable or disable logging of Data Store Memory block variables for this model.
“Log Dataset data to file” on page 3-33	Log data to MAT-file.
“Output options” on page 3-36	Select options for generating additional output signal data for variable-step solvers.
“Refine factor” on page 3-38	Specify how many points to generate between time steps to refine the output.
“Output times” on page 3-40	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 3-42	Enable the single-output format of the `sim` command.
“Logging intervals” on page 3-44	Set intervals for logging
“Record logged workspace data in Simulation Data Inspector” on page 3-47	Specify whether to send logged states and Simscape™ 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 2-58	Format for logged `Dataset` leaf elements.

See Also

Related Examples

- Importing Data from a Workspace
- “Export Simulation Data”
- “Export Signal Data Using Signal Logging”

Data Import/Export Overview

The Data Import/Export pane allows 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.

Configuration

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.

Tips

- To open the **Data Import/Export** pane, in the Simulink Editor, select **Simulation** > **Model Configuration Parameters** > **Data Import/Export**.
- For more information importing and exporting data, see "Load Signal Data for Simulation" and "Save Runtime Data from Simulation".
- See the documentation of the `sim` command for some capabilities that are available only for programmatic simulation.

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 Import/Export" on page 3-2

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 imported from a workspace. The Simulink software resolves symbols used in this specification as described in "Symbol Resolution".

See "Load Data to Root-Level Input Ports" for information on how to use this field.

 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 Inport.
- 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".

Command-Line Information

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.

 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 a SimState.
- 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).

Command-Line Information

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
- “State Information”
- “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.
- The **Time, State, Output** area 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.

Command-Line Information

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

States

Description

Saves state data to the specified MATLAB variable during a simulation.

Category: Data Import/Export

Settings

Default: Off, `xout`

 On

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

Specify the name of the MATLAB variable used to store state data. See Importing and Exporting States for more information.

 Off

Simulink does not export state data during simulation.

Tips

- Simulink saves the states in a MATLAB workspace variable having the specified name.
- The saved data has the format that you specify with the **Format** parameter.
- If you select the **States** check box, Simulink logs fixed-point states only if you set the **Format** parameter to `Dataset`.
- `Dataset` format does not support:
  - Logging states information inside a function-call subsystem
  - Code generation
- Simulink creates empty variables for state logging (`xout`) if both of these conditions apply:
  - You enable **States**.

  - A model has no states.
- 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".
- To log **States** data to the Simulation Data Inspector, check the **Record logged workspace data in Simulation Data Inspector** parameter.

Command-Line Information

Parameter: `SaveState`
Value: `'on' | 'off'`
Default: `'off'`
Parameter: `StateSaveName`
Type: character vector
Value: any valid value
Default: `'xout'`

Recommended Settings

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

See Also

Related Examples

- "State Information"
- "Comparison of Signal Loading Techniques"
- "Model Configuration Parameters: Data Import/Export" on page 3-2

Output

Description

Saves root Outport data to the specified MATLAB variable during simulation.

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 **Time, State, Output** 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.

- 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".
- 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 invoke a `sim` command inside a function, the output logged by the function is in the function workspace. To be able 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.

Command-Line Information

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

- “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 SimState 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 3-26 and “Save and Restore Simulation State as SimState”.

- See “State Information” 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”.

Command-Line Information

Parameter: `SaveFinalState`
Value: `'on' | 'off'`
Default: `'off'`
Parameter: `FinalStateName`
Type: character vector
Value: any valid value
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

Select the data format for saving states, output, and final states data.

Category: Data Import/Export

Settings

Default: `Dataset`

`Dataset`

Simulink uses a `Simulink.SimulationData.Dataset` object to store the logged data as MATLAB `timeseries` objects.

`Array`

Simulink saves the data in a matrix. Each row corresponds to a simulation time step.

`Structure`

For logging output, Simulink saves the data in a structure that contains substructures for each port. Each port substructure contains signal data for the corresponding port. For logging states, the structure contains a substructure for each block with a state.

`Structure with time`

The format of the data is a structure that has two fields: 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

- The `Dataset` format for logged state and root outport data:
  - Uses MATLAB `timeseries` objects to store logged data. MATLAB `timeseries` objects allow you to work with logged data in MATLAB without a Simulink license.
  - Supports logging multiple data values for a given time step, important for Iterator subsystem and Stateflow signal logging.

    - Supports logging **Output** data to the Simulation Data Inspector.
- For states logging, `Dataset` format does not support:
    - Logging states information inside a function-call subsystem
    - Code generation
- You can use array format to save model outputs and states only if the outputs:
    - Are all scalars or all vectors (or all matrices for states)
    - Are all real or all complex
    - Have the same data type

  Use the `Dataset`, `Structure`, or `Structure with time` output formats if your model outputs and states do not meet these conditions.
- Rapid accelerator mode does not support `Dataset` format for states and final states.
- If you enable the **Save complete SimState in final state** parameter, the specified format does not apply to final states data.
- Simulink can read back simulation data saved to the workspace in the `Structure with time` output format. See "Load Data to Root-Level Input Ports" for more information.

Command-Line Information

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

Limit data points

Description

Limit the number of data points logged to the MATLAB workspace.

Category: Data Import/Export

Settings

Default: Off, `1000`

 On

Limits the number of data points logged to the MATLAB workspace to the specified number.

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

 Off

Does not limit the number of data points logged to the MATLAB workspace.

Tips

- Saving data to the MATLAB workspace can consume memory. Use this parameter to limit the number of samples saved to help avoid this problem.
- You can also apply a **Decimation** factor to reduce the number of saved samples.

Command-Line Information

Parameter: `LimitDataPoints`
Value: `'on' | 'off'`
Default: `'off'`
Parameter: `MaxDataPoints`
Type: character vector
Value: any valid value

Default: `'1000'`

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

Decimation

Description

Specify the decimation factor, `N`, such that Simulink outputs data every `N` points.

Category: Data Import/Export

Settings

Default: `1`

- With the default value, `1`, all data points are saved.
- The specified value must be a positive integer greater than zero.
- 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

- Saving data to the MATLAB workspace can consume memory. Use **Decimation** to limit the number of saved samples to consume less memory.
- You can also use the **Limit data points to last** parameter to help resolve memory consumption issues.

Command-Line Information

Parameter: `Decimation`
Type: character vector
Value: any valid value
Default: `'1'`

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

Save complete SimState in final state

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**.

Command-Line Information

Parameter: `SaveCompleteFinalSimState`

Value: `'on' | 'off'`
Default: `'off'`
Parameter: `FinalStateName`
Type: character vector
Value: any valid value
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
- “Limitations of SimState”
- “Model Configuration Parameters: Data Import/Export” on page 3-2

Signal logging

Description

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

Category: Data Import/Export

Settings

Default: On, `logsout`

 On

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

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

 Off

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

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.
- When signal logging is disabled, signals marked for logging still send data to the Simulation Data Inspector.
- 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 with 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.

Dependencies

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

Command-Line Information

Parameter: `SignalLogging`
Value: `'on' | 'off'`
Default: `'on'`
Parameter: `SignalLoggingName`
Type: character vector
Value: any valid value
Default: `'logsout'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation

Application	Setting
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.

Command-Line Information

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
Traceability	No impact
Efficiency	No recommendation
Safety precaution	No recommendation

See Also

Data Store Memory | `Simulink.SimulationData.DataStoreMemory`

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 data to MAT-file.

Category: Data Import/Export

Settings

Default: `'off'`

 On

Enables logging data that uses `Dataset` format to a MAT-file.

Use this feature when logging large amounts of data that can cause memory issues. For details, see “Log Data to Persistent Storage”.

Specify a character vector for the path to the file to use for logging signals and states. Do not use a file name from one locale in a different locale.

 Off

Disables logging data to a MAT-file.

Tips

- To use the **Log Dataset data to file** option:
  - Select one or more of these kinds of logging:
    - **States**
    - **Final states**
    - **Signal logging**
    - **Output**
    - **Data stores**
  - If you are logging states or output data, set the **Format** parameter to `Dataset`.

  - If you select the **Final states** parameter, clear the **Save complete SimState in final state** parameter.
- To control whether logged data is sent to visualization tools such as the Simulation Data Inspector, use the `VisualizeLoggedSignalsWhenLoggingToFile` model parameter. By default, the data is not streamed to visualization tools when logging to file is enabled. For details, see "Model Parameters".
- To access in the MAT-file the `Simulink.SimulationData.Dataset` data for a logging variable (for example, the `logsout` variable for signal logging data), you can create a `Simulink.SimulationData.DatasetRef` object. Using a `Simulink.SimulationData.DatasetRef` object to access signal logging and states data stored in the MAT-file loads the data into memory incrementally (signal by signal).
- You can load the MAT-file contents into memory without creating a `Simulink.SimulationData.DatasetRef` object. You can load either the whole file or a `Dataset` object in that file using the MATLAB `load` command. However, to load data signal by signal or load individual signals incrementally, use a `Simulink.SimulationData.DatasetRef` object.

Dependencies

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

Command-Line Information

Parameter: `LoggingToFile`
Value: `'on' | 'off'`
Default: `'off'`
Parameter: `LoggingFileName`
Value: any valid value
Default: `'out.mat'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No recommendation

Application	Setting
Safety precaution	No recommendation

See Also

`Simulink.SimulationData.Dataset` | `Simulink.SimulationData.DatasetRef` | `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 17-11).

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

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

Command-Line Information

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 3-38
- “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**.

Command-Line Information

Parameter: `Refine`
Type: character vector
Value: any valid value
Default: `'1'`

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

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 at the MATLAB command line.
- 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`.

Command-Line Information

Parameter: `OutputTimes`
Type: character vector
Value: any valid value
Default: `'[]'`

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

Single simulation output

Description

Enable the single-output format of the `sim` command.

Category: Data Import/Export

Settings

Default: off

When you enable this option:

- Simulink returns all simulation outputs within a single `Simulink.SimulationOutput` object, providing that you simulate by choosing **Simulation** > **Start** from the model window.
- You must specify the variable name of the single output object which will contain the simulation outputs. Use the text field next to the check box to specify this name.
- The `sim` command becomes compatible with the `parfor` command, in terms of transparency issues.
- The setting overrides the `Dataset` format for signal logging data.

Tips

- To use the **Logging interval**s parameter, you must select **Single simulation output**.
- If you select this option and you simulate by entering the `sim` command at the command line of the MATLAB command window, then the output variables will not be stored in the object `'out'`. Instead, they will be stored in their respective variable names.
- The method who of the `Simulink.SimulationOutput` object returns the list of variables that the object contains.
- Use the `get` method of the `Simulink.SimulationOutput` object to access the variables that the object contains.

Command-Line Information

Parameter: `ReturnWorkspaceOutputs`
Value: `'on' | 'off' |`
Default: `'off'`
Parameter: `ReturnWorkspaceOutputsName`
Type: character vector
Value: `Any valid value`
Default: `'Out'`

Recommended Settings

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

See Also

Related Examples

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

Logging intervals

Description

Set intervals for logging

Category: Data Import/Export

Settings

Default: `[-inf,inf]`

- Use a real double matrix with two columns.
- The matrix elements cannot be `NaN`.
- You can specify as many intervals as you want.
- Each row defines the start and end times for an interval.
- 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 data logged for:
  - Time
  - States
  - Output
  - Signal logging
  - The To Workspace block
  - The To File block

  The logging intervals do not apply to final state logged data, scopes or streaming data to the Simulation Data Inspector.
- PIL simulation mode does not support logging intervals. Simulink ignores specified logging intervals, without displaying a warning.

- SIL simulation mode supports logging intervals for data logged to a `Simulink.SimulationOutput` object. In SIL mode, Simulink ignores specified logging intervals, without displaying a warning, for:
  - Data logged to a To File block
  - MAT-file logging (enabled with the **MAT-file logging** configuration parameter)
- The interval times that meet either of these two conditions do not return logged data:
  - The time is before the simulation start time.
  - The time is after the simulation stop time.

  Interval times that meet these conditions do not cause a warning.
- All logged data, except for data logged to a To File block, is stored in the object you specify for the **Single simulation output** parameter. Data for the To File block reflects the specified intervals, but is stored in the file associated with the block.
- To prevent logging of To Workspace blocks, set **Logging intervals** to an empty matrix (`[]`).
- If you set **Decimation** to `2`, then the logged data is for alternating times in the intervals. In other words, data is for times 2, 4, and 8.
- If you set **Limit data points to last** to `4`, then the logged data is for the last four times in the intervals. In other words, data is for times 4, 7, 8, and 9.
- Simulation Stepper rollback reflects logging intervals. If you change the logging intervals of a simulation before rollback, logging:
  - Includes data starting with the first step after the rollback
  - Does not include data for time steps that are outside of the original logging intervals

Dependency

This parameter is enabled only if you select the **Single simulation output** parameter.

Command-Line Information

Parameter: `LoggingIntervals`
Type: real double matrix with two columns
Default: `[-inf,inf]`

Recommended Settings

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

See Also

Related Examples

- “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 logged states and Simscape data to the Simulation Data Inspector after simulation pauses or completes.

Category: Data Import/Export

Settings

Default: Off

 On

Record logged **States** and Simscape data to send to the Simulation Data Inspector when a simulation pauses or completes. This setting adds a recording icon to the **Simulation Data Inspector** button on the Simulink Editor toolbar. After a simulation is recorded, the logged simulation data appears in a run in the **Inspect** pane of the Simulation Data Inspector.

 Off

Do not record logged signals during simulation.

Tip

To record **States**, `Dataset` is the recommended **Format**. You can also set **Format** to `Structure with time` or `Array`. If **Format** is configured as `Array`, you must also log **Time** for **States** to record to the Simulation Data Inspector.

To open the Simulation Data Inspector, on the Simulink Editor toolbar, click the **Simulation Data Inspector** button arrow and select `Simulation Data Inspector`.

Command-Line Information

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

Related Examples

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

Optimization Parameters

Optimization Pane: General

The **Optimization > General** pane includes parameters for improving the simulation speed of your models and improving the performance of the generated code. Model configuration parameters to improve the generated code require Simulink Coder or Embedded Coder.

Parameter	Description
"Application lifespan (days)" on page 4-6	Specify how long (in days) an application that contains blocks depending on elapsed or absolute time should be able to execute before timer overflow.
"Use division for fixed-point net slope computation" on page 4-9	The Fixed-Point Designer™ software performs net slope computation using division to handle net slopes when simplicity and accuracy conditions are met.
"Use floating-point multiplication to handle net slope corrections" on page 4-11	The Fixed-Point Designer software uses floating-point multiplication to perform net slope correction for floating-point to fixed-point casts.
"Default for underspecified data type" on page 4-13	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.
"Optimize using the specified minimum and maximum values" on page 4-15	Optimize generated code using the specified minimum and maximum values for signals and parameters in the model.
"Remove root level I/O zero initialization" on page 4-18	Specify whether to generate initialization code for root-level inports and outports set to zero.
"Remove internal data zero initialization" on page 4-20	Specify whether to generate initialization code for internal work structures, such as block states and block outputs, to zero.
"Remove code from floating-point to integer conversions that wraps out-of-range values" on page 4-22	Remove wrapping code that handles out-of-range floating-point to integer conversion results.

Parameter	Description
"Remove code that protects against division arithmetic exceptions" on page 4-24	Specify whether to generate code that guards against division by zero and `INT_MIN/-1` operations for integers and fixed-point data.

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

Parameter	Description
"Remove code from floating-point to integer conversions with saturation that maps NaN to zero" on page 2-90	Remove code that handles floating-point to integer conversion results for `NaN` values.
"Compiler optimization level" on page 2-92	Sets the degree of optimization used by the compiler when generating code for acceleration.
"Verbose accelerator builds" on page 2-94	Select the amount of information displayed during code generation for Simulink Accelerator mode, referenced model Accelerator mode, and Rapid Accelerator mode.
"Implement logic signals as Boolean data (vs. double)" on page 2-96	Controls the output data type of blocks that generate logic signals.
"Block reduction" on page 2-99	Reduce execution time by collapsing or removing groups of blocks.
"Conditional input branch execution" on page 2-102	Improve model execution when the model contains Switch and Multiport Switch blocks.
"Use memset to initialize floats and doubles to 0.0" on page 2-104	Specify whether to generate code that explicitly initializes floating-point data to 0.0.
"Signal storage reuse" on page 2-106	Reuse signal memory.
Evaluated application lifespan	Evaluated version of the "Application lifespan (days)" on page 4-6 parameter. This parameter is read-only.

Parameter	Description
Disable incompatible optimizations	Specify whether to disable optimizations that are incompatible with Simulink Code Inspector.

Optimization Pane: General Tab Overview

Set up optimizations for a model's active configuration set. Optimizations are set for both simulation and code generation.

Tips

- To open the Optimization pane, in the Simulink Editor, select **Simulation** > **Model Configuration Parameters** > **Optimization**.
- Simulink Coder optimizations appear only when the Simulink Coder product is installed on your system. Selecting a GRT-based or ERT-based system target file changes the available options. ERT-based target optimizations require a Embedded Coder license when generating code. See the **Dependencies** sections below for licensing information for each parameter.

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

- “Optimization Pane: General” on page 4-2
- “Perform Acceleration”
- “Performance” (Simulink Coder)

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: Optimization

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) in the Simulink Coder documentation.
- 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 (10E-09 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)
- “Optimization Pane: General” on page 4-2
- “Performance” (Simulink Coder)

Use division for fixed-point net slope computation

Description

The Fixed-Point Designer software performs net slope computation using division to handle net slopes when simplicity and accuracy conditions are met.

Category: Optimization

Settings

Default: `Off`

`Off`

Performs net slope computation using integer multiplication followed by shifts.

`On`

Performs net slope computation using a rational approximation of the net slope. This results in an integer multiplication and/or division 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 slope computation is necessary. Normally, net slope computation uses an integer multiplication followed by shifts. Enabling this new optimization replaces the multiplication and shifts with an integer division or an integer multiplication and division under certain simplicity and accuracy conditions.
- Performing net slope 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.

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)
- “Optimization Pane: General” on page 4-2

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: Optimization

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)
Safety precaution	No recommendation

See Also

Related Examples

- “Floating-Point Multiplication to Handle a Net Slope Correction” (Simulink Coder)
- “Optimization Pane: General” on page 4-2

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: Optimization

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 9-12
- “Validate a Single-Precision Model”
- “Optimization Pane: General” on page 4-2

Optimize using the specified minimum and maximum values

Description

Optimize generated code using the specified minimum and maximum values for signals and parameters in the model.

Category: Optimization

Settings

Default: Off

 On

Optimizes the generated code using range information derived from the minimum and maximum specified values for signals and parameters in the model.

 Off

Ignores specified minimum and maximum values when generating code.

Tips

- To detect mismatches between model and generated code simulations that arise from the use of this parameter, before running normal, accelerator, software-in-the-loop (SIL), or processor-in-the-loop (PIL) (Embedded Coder) simulations, set **Diagnostics > Data Validity > Simulation range checking** to `Warning` or `Error`.
- Specify minimum and maximum values for signals and parameters in the model for:
  - Inport and Outport blocks.
  - Block outputs.
  - Block inputs, for example, for the MATLAB Function and Stateflow Chart blocks.
  - `Simulink.Signal` objects.
- This optimization does not take into account minimum and maximum values specified for:
  - Merge block inputs. To work around this, use a `Simulink.Signal` object on the Merge block output and specify the range on this object

- Bus elements.
- Conditionally-executed subsystem (such as a triggered subsystem) block outputs that are directly connected to an Outport block.

  Outport blocks in conditionally-executed subsystems can have an initial value specified for use only when the system is not triggered. In this case, the optimization cannot use the range of the block output because the range might not cover the initial value of the block.

- If you use the Polyspace® Code Prover™ software to verify code generated using this optimization, it might mark code that was previously green as orange. For example, if your model contains a division where the range of the denominator does not include zero, the generated code does not include protection against division by zero. Polyspace Code Prover might mark this code orange because it does not have information about the minimum and maximum values specified for the inputs to the division.

  The Polyspace Code Prover software does automatically capture some minimum and maximum values specified in the MATLAB workspace, for example, for `Simulink.Signal` and `Simulink.Parameter` objects. In this example, to provide range information to the Polyspace Code Prover software, use a `Simulink.Signal` object on the input of the division and specify a range that does not include zero.

  The Polyspace Code Prover software stores these values in a Data Range Specification (DRS) file. However, they do not capture all minimum and maximum values specified in your Simulink model. To provide additional min/max information to Polyspace Code Prover, you can manually define a DRS file. For more information, see the Polyspace Code Prover documentation.

- If you are using double-precision data types and the **Code Generation** > **Interface** > **Support non-finite numbers** configuration parameter is selected, this optimization does not occur.
- If your model contains multiple instances of a reusable subsystem and each instance uses input signals with different specified minimum and maximum values, this optimization might result in different generated code for each subsystem so code reuse does not occur. Without this optimization, the Simulink Coder software generates code once for the subsystem and shares this code among the multiple instances of the subsystem.
- The Model Advisor **Check safety-related optimization settings** check generates a warning if this option is selected. For many safety critical applications, it is not acceptable to remove dead code automatically because this might result in

requirements without traceable code. For more information, see Check safety-related optimization settings (Simulink Check).

- Enabling this optimization improves the ability of the Fixed-Point Designer software to eliminate unnecessary utility functions and saturation code from the generated code.

Dependencies

- This parameter appears for ERT-based targets only.
- This parameter requires a Embedded Coder license when generating code.

Command-Line Information

Parameter: `UseSpecifiedMinMax`
Value: `'on' | 'off'`
Default: `'off'`

Recommended Settings

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

See Also

Related Examples

- "Optimize Generated Code Using Minimum and Maximum Values" (Embedded Coder)
- "Optimize Generated Code Using Specified Minimum and Maximum Values" (Fixed-Point Designer)
- "Optimization Pane: General" on page 4-2

Remove root level I/O zero initialization

Description

Specify whether to generate initialization code for root-level inports and outports set to zero.

Category: Optimization

Settings

Default: Off (GUI), `'on'` (command-line)

 On

Does not generate initialization code for root-level inports and outports set to zero.

 Off

Generates initialization code for all root-level inports and outports. Use the default:

- To initialize memory allocated for C MEX S-function wrappers to zero.
- To initialize all internal and external data to zero.

Note Generated code never initializes data of `ImportedExtern` or `ImportedExternPointer` storage classes, regardless of configuration parameter settings.

Dependencies

- This parameter appears only for ERT-based targets.
- This parameter requires a Embedded Coder license when generating code.

Command-Line Information

Parameter: `ZeroExternalMemoryAtStartup`
Value: `'off' | 'on'`
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	On (GUI), `off` (command line) (execution, ROM), No impact (RAM)
Safety precaution	Off (GUI), `on` (command line)

See Also

Related Examples

- "Remove Initialization Code for Root-Level Inports and Outports Set to Zero" (Embedded Coder)
- "Optimization Pane: General" on page 4-2
- "Performance" (Simulink Coder)

Remove internal data zero initialization

Description

Specify whether to generate initialization code for internal work structures, such as block states and block outputs, to zero.

Category: Optimization

Settings

Default: Off (GUI), `'on'` (command-line)

 On

Does not generate code that initializes internal work structures to zero. An example of when you might select this parameter is to test the behavior of a design during warm boot—a restart without full system reinitialization.

Selecting this parameter does not guarantee that memory is in a known state each time the generated code begins execution. When you run a model or generated S-function multiple times, each run can produce a different answer, even when calling the model initialization function in an attempt to reset memory.

If want to get the same answer on every run from a generated S-function, enter the command `clear` *`SFcnNam`* or `clear mex` in the MATLAB Command Window before each run.

 Off

Generates code that initializes internal work structures to zero. You should use the default:

- To ensure that memory allocated for C MEX S-function wrappers is initialized to zero
- For safety critical applications that require that all internal and external data be initialized to zero

Dependencies

- This parameter appears only for ERT-based targets.
- This parameter requires a Embedded Coder license when generating code.

Command-Line Information

Parameter: `ZeroInternalMemoryAtStartup`
Value: `'off' | 'on'`
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	On (GUI), `off` (command line), (execution, ROM), No impact (RAM)
Safety precaution	Off (GUI), `on` (command line)

See Also

Related Examples

- "Remove Zero Initialization Code for Internal Data" (Embedded Coder)
- "Optimization Pane: General" on page 4-2
- "Performance" (Simulink Coder)

Remove code from floating-point to integer conversions that wraps out-of-range values

Description

Remove wrapping code that handles out-of-range floating-point to integer conversion results.

Category: Optimization

Settings

Default: Off

On

Removes code when out-of-range conversions occur. Select this check box if code efficiency is critical to your application and the following conditions are true for at least one block in the model:

- Computing the outputs or parameters of a block involves converting floating-point data to integer or fixed-point data.
- The **Saturate on integer overflow** check box is cleared in the Block Parameters dialog box.

Caution Execution of generated code might not produce the same results as simulation.

Off

Results for simulation and execution of generated code match when out-of-range conversions occur. The generated code is larger than when you select this check box.

Tips

- Selecting this check box reduces the size and increases the speed of the generated code at the cost of potentially producing results that do not match simulation in the case of out-of-range values.

- Selecting this check box affects code generation results only for out-of-range values and cannot cause code generation results to differ from simulation results for in-range values.

Dependency

This parameter requires a Simulink Coder license.

Command-Line Information

Parameter: `EfficientFloat2IntCast`
Value: `'on' | 'off'`
Default: `'off'`

Recommended Settings

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

See Also

Related Examples

- "Remove Code From Floating-Point to Integer Conversions That Wraps Out-of-Range Values" (Simulink Coder)
- "Optimization Pane: General" on page 4-2

Remove code that protects against division arithmetic exceptions

Description

Specify whether to generate code that guards against division by zero and `INT_MIN/-1` operations for integers and fixed-point data.

For more information on division arithmetic exceptions, see “Division Arithmetic Exceptions in Generated Code” (Embedded Coder).

Category: Optimization

Settings

Default: On

 On

Does not generate code that guards against division by zero and `INT_MIN/-1` operations for integers and fixed-point data. To retain bit-true agreement between simulation results and results from generated code, ensure that your model never produces division by zero or `INT_MIN/-1` operations, where the quotient cannot be represented in the data type.

 Off

Generates code that guards against division by zero and `INT_MIN/-1` operations for integers and fixed-point data.

Dependencies

- This parameter appears only for ERT-based targets.
- This parameter requires a Embedded Coder license when generating code.

Command-Line Information

Parameter: `NoFixptDivByZeroProtection`
Value: `'on' | 'off'`
Default: `'on'`

Recommended Settings

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

See Also

Related Examples

- “Remove Code That Guards Against Division Exceptions for Integers and Fixed-Point Data” (Embedded Coder)
- “Division Arithmetic Exceptions in Generated Code” (Embedded Coder)
- “Optimization Pane: General” on page 4-2
- “Performance” (Simulink Coder)

5

Optimization Parameters: Signals and Parameters

Optimization Pane: Signals and Parameters

The **Optimization > Signals and Parameters** pane includes parameters for improving the simulation speed of your models and improving the performance of the generated code. Model configuration parameters to improve the generated code require Simulink Coder or Embedded Coder.

Parameter	Description
"Default parameter behavior" on page 5-5	Transform numeric block parameters into constant inlined values in the generated code.
"Inline invariant signals" on page 5-8	Transform symbolic names of invariant signals into constant values.
"Use memcpy for vector assignment" on page 5-10	Optimize code generated for vector assignment by replacing `for` loops with `memcpy`.
"Memcpy threshold (bytes)" on page 5-12	Specify the minimum array size in bytes for which `memcpy` and `memset` function calls should replace `for` loops for vector assignments in the generated code.
"Pack Boolean data into bitfields" on page 5-14	Specify whether Boolean signals are stored as one–bit bitfields or as a Boolean data type.
"Bitfield declarator type specifier" on page 5-16	Specify the bitfield type when selecting configuration parameter "Pack Boolean data into bitfields" on page 5-14.
"Loop unrolling threshold" on page 5-18	Specify the minimum signal or parameter width for which a `for` loop is generated.
"Maximum stack size (bytes)" on page 5-20	Specify the maximum stack size in bytes for your model.
"Pass reusable subsystem outputs as" on page 5-22	Specify how a reusable subsystem passes outputs.
"Parameter structure" on page 5-24	Control how parameter data is generated for reusable subsystems.

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

Parameter	Description
“Enable local block outputs” on page 2-109	Specify whether block signals are declared locally or globally.
“Reuse local block outputs” on page 2-111	Specify whether Simulink Coder software reuses signal memory.
“Eliminate superfluous local variables (Expression folding)” on page 2-113	Collapse block computations into single expressions.
“Reuse global block outputs” on page 2-115	Reuse global memory for block outputs.
“Optimize global data access” on page 2-117	Select global variable optimization.
“Simplify array indexing” on page 2-119	Replace multiply operations in array indices when accessing arrays in a loop.
“Perform inplace updates for Bus Assignment blocks” on page 2-142	Reuse the input and output variables of Bus Assignment blocks if possible.
“Reuse buffers for Data Store Read and Data Store Write blocks” on page 2-144	Remove temporary buffers for Data Store Read and Data Store Write blocks. Use the Data Store Memory block directly if possible.
“Optimize block operation order in the generated code” on page 2-146	Reorder block operations in the generated code for improved code execution speed.
“Reuse buffers of different sizes and dimensions” on page 5-29	Reduce memory consumption by reusing buffers to store data of different sizes and dimensions.
Buffer for reusable subsystems	No further documentation is available for this parameter.
Maximum number of arguments for subsystem outputs	Set maximum number of subsystem outputs to pass individually.

See Also

Related Examples

- “Performance” (Simulink Coder)

Optimization Pane: Signals and Parameters Tab Overview

Set up optimizations for a model's active configuration set.

Tips

- To open the Optimization: Signals and Parameters pane, in the Simulink Editor, select **Simulation > Model Configuration Parameters > Optimization > Signals and Parameters**.
- Simulink Coder optimizations appear only when the Simulink Coder product is installed on your system. Selecting a GRT-based or ERT-based system target file changes the available options. ERT-based target optimizations require a Embedded Coder license when generating code. See the **Dependencies** sections below for licensing information for each parameter.

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

- “Optimization Pane: Signals and Parameters” on page 5-2
- “Performance” (Simulink Coder)

Default parameter behavior

Description

Transform numeric block parameters into constant inlined values in the generated code.

Category: Optimization

Settings

Default: `Tunable` for GRT targets | `Inlined` for ERT targets

`Inlined`

Set **Default parameter behavior** to `Inlined` to reduce global RAM usage and increase efficiency of the generated code. The code does not allocate memory to represent numeric block parameters such as the **Gain** parameter of a Gain block. Instead, the code inlines the literal numeric values of these block parameters.

`Tunable`

Set **Default parameter behavior** to `Tunable` to enable tunability of numeric block parameters in the generated code. The code represents numeric block parameters and variables that use the storage class `Auto`, including numeric MATLAB variables, as tunable fields of a global parameters structure.

Tips

- Whether you set **Default parameter behavior** to `Inlined` or to `Tunable`, create parameter data objects to preserve tunability for block parameters. For more information, see "Block Parameter Representation in the Generated Code" (Simulink Coder).
- When you switch from a system target file that is not ERT-based to one that is ERT-based, **Default parameter behavior** sets to `Inlined` by default. However, you can change the setting of **Default parameter behavior** later.
- When a top model uses referenced models or if a model is referenced by another model:
  - All referenced models must set **Default parameter behavior** to `Inlined` if the top model has **Default parameter behavior** set to `Inlined`.

  - The top model can specify **Default parameter behavior** as `Tunable` or `Inlined`.
- If your model contains an Environment Controller block, you can suppress code generation for the branch connected to the Sim port if you set **Default parameter behavior** to `Inlined` and the branch does not contain external signals.

Dependencies

When you set **Default parameter behavior** to `Inlined`, you enable these configuration parameters:

- **"Parameter structure" on page 5-24**
- **"Inline invariant signals" on page 5-8**

Command-Line Information

Parameter: `DefaultParameterBehavior`
Type: character vector
Value: `'Inlined'` | `'Tunable'`
Default: `'Tunable'` for GRT targets | `'Inlined'` for ERT targets

Recommended Settings

Application	Setting
Debugging	`Tunable` during development `Inlined` for production code generation
Traceability	`Inlined`
Efficiency	`Inlined`
Safety precaution	No impact

See Also

Related Examples

- "Optimization Pane: Signals and Parameters" on page 5-2

- “Inline Numeric Values of Block Parameters” (Simulink Coder)
- “Block Parameter Representation in the Generated Code” (Simulink Coder)

Inline invariant signals

Description

Transform symbolic names of invariant signals into constant values.

Category: Optimization

Settings

Default: Off

On

Simulink Coder software uses the numerical values of model parameters, instead of their symbolic names, in generated code. An invariant signal is not inline if it is nonscalar, complex, or the block inport the signal is attached to takes the address of the signal.

Off

Uses symbolic names of model parameters in generated code.

Dependencies

- This parameter requires a Simulink Coder license.
- This parameter is enabled when you set **Default parameter behavior** to `Inlined`.

Command-Line Information

Parameter: `InlineInvariantSignals`
Value: `'on' | 'off'`
Default: `'off'`

Recommended Settings

Application	Setting
Debugging	Off

Application	Setting
Traceability	Off
Efficiency	On
Safety precaution	No impact

See Also

Related Examples

- “Optimization Pane: Signals and Parameters” on page 5-2
- “Inline Invariant Signals” (Simulink Coder)
- “Performance” (Simulink Coder)

Use memcpy for vector assignment

Description

Optimize code generated for vector assignment by replacing `for` loops with `memcpy`.

Category: Optimization

Settings

Default: On

On

Enables use of `memcpy` for vector assignment based on the associated threshold parameter **Memcpy threshold (bytes)**. `memcpy` is used in the generated code if the number of array elements times the number of bytes per element is greater than or equal to the specified value for **Memcpy threshold (bytes)**. One byte equals the width of a character in this context.

Off

Disables use of `memcpy` for vector assignment.

Dependencies

- This parameter requires a Simulink Coder license.
- When selected, this parameter enables the associated parameter **Memcpy threshold (bytes)**.

Command-Line Information

Parameter: `EnableMemcpy`
Value: `'on' | 'off'`
Default: `'on'`

Recommended Settings

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

See Also

Related Examples

- “Optimization Pane: Signals and Parameters” on page 5-2
- “Use memcpy Function to Optimize Generated Code for Vector Assignments” (Simulink Coder)
- “Performance” (Simulink Coder)

Memcpy threshold (bytes)

Description

Specify the minimum array size in bytes for which `memcpy` and `memset` function calls should replace `for` loops for vector assignments in the generated code.

Category: Optimization

Settings

Default: `64`

Dependencies

- This parameter requires a Simulink Coder license.
- For the `memcpy` optimization, this parameter is enabled when you select **Use memcpy for vector assignment**.

Command-Line Information

Parameter: `MemcpyThreshold`
Type: integer
Value: any valid quantity of bytes
Default: `64`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	Accept default or determine target-specific optimal value
Safety precaution	No impact

See Also

Related Examples

- “Optimization Pane: Signals and Parameters” on page 5-2
- “Use memcpy Function to Optimize Generated Code for Vector Assignments” (Simulink Coder)
- “Performance” (Simulink Coder)

Pack Boolean data into bitfields

Description

Specify whether Boolean signals are stored as one–bit bitfields or as a Boolean data type.

Category: Optimization

Note You cannot use this optimization when you generate code for a target that specifies an explicit structure alignment.

Settings

Default: Off

 On

Stores Boolean signals into one–bit bitfields in global block I/O structures or DWork vectors. This will reduce RAM, but might cause more executable code.

 Off

Stores Boolean signals as a Boolean data type in global block I/O structures or DWork vectors.

Dependencies

This parameter:

- Requires a Embedded Coder license.
- Appears only for ERT-based targets.

Command-Line Information

Parameter: `BooleansAsBitfields`
Value: `'on' | 'off'`
Default: `'off'`

Recommended Settings

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

See Also

Related Examples

- “Optimization Pane: Signals and Parameters” on page 5-2
- “Optimize Generated Code By Packing Boolean Data Into Bitfields” (Embedded Coder)
- “Bitfield declarator type specifier” on page 5-16
- “Performance” (Embedded Coder)

Bitfield declarator type specifier

Description

Specify the bitfield type when selecting configuration parameter "Pack Boolean data into bitfields" on page 5-14.

Category: Optimization

Note The optimization benefit is dependent upon your choice of target.

Settings

Default: `uint_T`

uint_T

The type specified for a bitfield declaration is an `unsigned int`.

uchar_T

The type specified for a bitfield declaration is an `unsigned char`.

Tip

The "Pack Boolean data into bitfields" on page 5-14 configuration parameter default setting uses unsigned integers. This might cause an increase in RAM if the bitfields are small and distributed. In this case, `uchar_T` might use less RAM depending on your target.

Dependency

Pack Boolean data into bitfields enables this parameter.

Command-Line Information

Parameter: `BitfieldContainerType`
Value: `uint_T` | `uchar_T`

Default: `uint_T`

Recommended Settings

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

See Also

Related Examples

- “Optimization Pane: Signals and Parameters” on page 5-2
- “Performance” (Embedded Coder)

Loop unrolling threshold

Description

Specify the minimum signal or parameter width for which a `for` loop is generated.

Category: Optimization

Settings

Default: `5`

Specify the array size at which the code generator begins to use a `for` loop instead of separate assignment statements to assign values to the elements of a signal or parameter array.

When there are perfectly nested loops, the code generator uses a for loop if the product of the loop counts for all loops in the perfect loop nest is greater than or equal to the threshold.

Dependency

This parameter requires a Simulink Coder license.

Command-Line Information

Parameter: `RollThreshold`
Type: character vector
Value: any valid value
Default: `'5'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	>0

Application	Setting
Safety precaution	No impact

See Also

Related Examples

- “Optimization Pane: Signals and Parameters” on page 5-2
- “Configure Loop Unrolling Threshold” (Simulink Coder)
- “Performance” (Simulink Coder)

Maximum stack size (bytes)

Description

Specify the maximum stack size in bytes for your model.

Category: Optimization

Settings

Default:`Inherit from target`

`Inherit from target`

The Simulink Coder software assigns the maximum stack size to the smaller value of the following:

- The default value (200,000 bytes) set by the Simulink Coder software
- Value of the TLC variable `MaxStackSize` in the system target file

`<Specify a value>`

Specify a positive integer value. Simulink Coder software assigns the maximum stack size to the specified value.

Note If you specify a maximum stack size for a model, the estimated required stack size of a referenced model must be less than the specified maximum stack size of the parent model.

Tips

- If you specify the maximum stack size to be zero, then the generated code implements all variables as global data.
- If you specify the maximum stack to be `inf`, then the generated code contains the least number of global variables.
- If your model contains a variable that is larger than 4096 bytes, the code generator implements it in global memory by default. You can increase the size of variables that the code generator places in local memory by changing the value of the TLC variable

`MaxStackVariableSize`. You can change this value by typing the following command in MATLAB Command Window:
`set_param(modelName,'TLCOptions','-aMaxStackVariableSize=N')`

Command-Line Information

Parameter: `MaxStackSize`
Type: `int`
Value: Any valid value
Default: `Inherit from target`

Recommended Settings

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

See Also

Related Examples

- "Optimization Pane: Signals and Parameters" on page 5-2
- "Customize Stack Space Allocation" (Simulink Coder)
- "Performance" (Simulink Coder)

Pass reusable subsystem outputs as

Description

Specify how a reusable subsystem passes outputs.

Category: Optimization

Settings

Default: `Individual arguments`

`Individual arguments`

Passes each reusable subsystem output argument as an address of a local, instead of as a pointer to an area of global memory containing all output arguments. This option reduces global memory usage and eliminates copying local variables back to global block I/O structures. When the signals are allocated as local variables, there may be an increase in stack size. If the stack size increases beyond a level that you want, use the default setting. By default, the maximum number of output arguments passed individually is 12. To increase the number of arguments, increase the value of the **Maximum number of arguments for subsystem outputs** parameter.

`Structure reference`

Passes reusable subsystem outputs as a pointer to a structure stored in global memory.

Note The default option is used for reusable subsystems that have signals with variable dimensions.

Dependencies

This parameter:

- Requires a Embedded Coder license.
- Appears only for ERT-based targets.

Command-Line Information

Parameter: `PassReuseOutputArgsAs`
Value: `'Structure reference' | 'Individual arguments'`
Default: `'Individual arguments'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	`Individual arguments` (execution, RAM),`Structure reference` (ROM)
Safety precaution	No impact

See Also

Related Examples

- “Optimization Pane: Signals and Parameters” on page 5-2
- “Generate Reusable Code from Library Subsystems Shared Across Models” (Simulink Coder)
- “Optimize Generated Code By Passing Reusable Subsystem Outputs as Individual Arguments” (Embedded Coder)
- “Performance” (Embedded Coder)

Parameter structure

Description

Control how parameter data is generated for reusable subsystems.

Category: Optimization

Settings

Default: `Hierarchical`

`Hierarchical`

Generates a separate header file, defining an independent parameter structure, for each subsystem that meets the following conditions:

- The subsystem **Code generation function packaging** parameter is set to `Reusable function`.
- The subsystem does not violate any code reuse limitations (Simulink Coder).
- The subsystem does not access parameters other than its own (such as parameters of the root-level model).

Each subsystem parameter structure is referenced as a substructure of the root-level parameter data structure, creating a structure hierarchy.

`NonHierarchical`

Generates a single, flat parameter data structure. Subsystem parameters are defined as fields within the structure. A nonhierarchical data structure can reduce compiler padding between word boundaries, producing more efficient compiled code.

Dependencies

- This parameter appears only for ERT-based targets.
- This parameter requires a Embedded Coder license when generating code.
- This parameter is enabled when you set **Default parameter behavior** to `Inlined`.

Command-Line Information

Parameter: `InlinedParameterPlacement`
Value: `'Hierarchical' | 'NonHierarchical'`
Default: `'Hierarchical'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	`Hierarchical`
Efficiency	`NonHierarchical`
Safety precaution	No impact

See Also

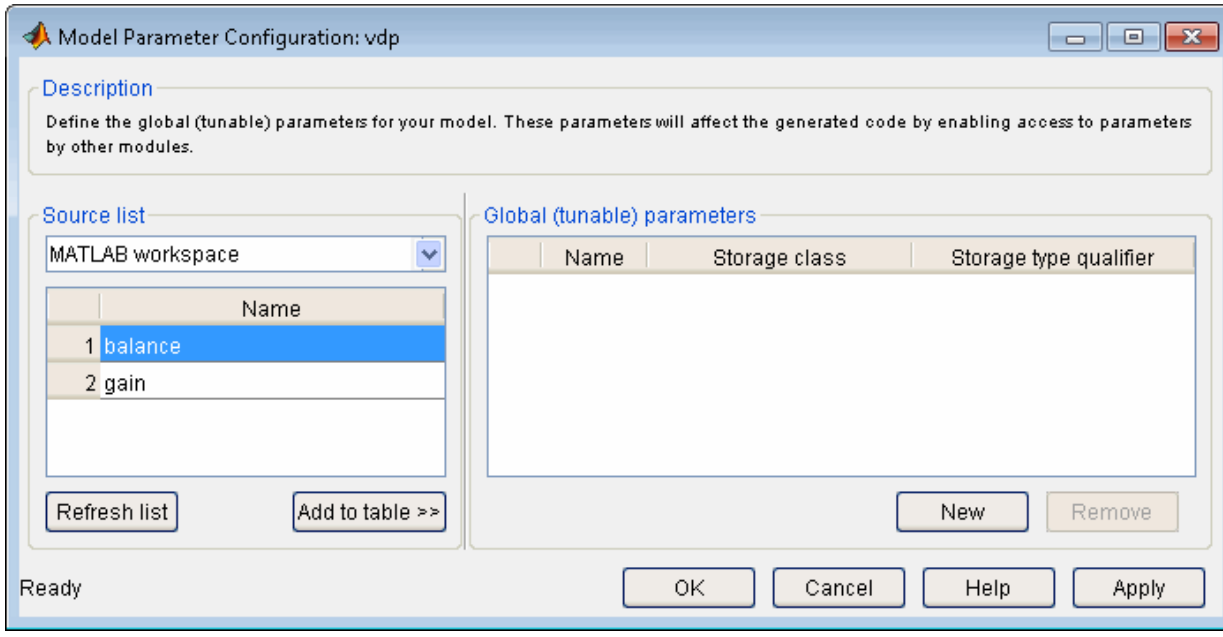
Related Examples

- “Optimization Pane: Signals and Parameters” on page 5-2
- “Flat Structures for Reusable Subsystem Parameters” (Embedded Coder)
- “Performance” (Embedded Coder)

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" on page 5-5.

To declare tunable parameters, use `Simulink.Parameter` objects instead of the **Model Parameter Configuration** dialog box. See "Block Parameter Representation 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 “Storage class” on page 19-9.

Storage type qualifier

Used for code generation. For more information, see “Type qualifier” on page 19-9.

See Also

Related Examples

- “Optimization Pane: Signals and Parameters” on page 5-2
- “Block Parameter Representation in the Generated Code” (Simulink Coder)

Reuse buffers of different sizes and dimensions

Reduce memory consumption by reusing buffers to store data of different sizes and dimensions.

Settings

Default: Off

 On

The code generator tries to reuse the same buffers to store data of different sizes and dimensions. This optimization conserves RAM and ROM consumption.

 Off

The code generator reuses buffers only if they have the same size and shape as the data.

Dependencies

- This parameter appears only for ERT-based targets.
- When generating code, this parameter requires an Embedded Coder license.
- This parameter is enabled by “Signal storage reuse” on page 2-106.

Tips

- If your model contains a reusable custom storage class to specify reuse on signals that have different sizes and shapes, you must select the **Reuse buffers of different sizes and dimensions** parameter or remove the specification. Otherwise, during simulation, the model produces an error.
- The code generator does not replace a buffer with a lower priority buffer that has a smaller size.
- The code generator does not reuse buffers that have different sizes and symbolic dimensions.

Command-Line Information

Parameter: `DifferentSizesBufferReuse`

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

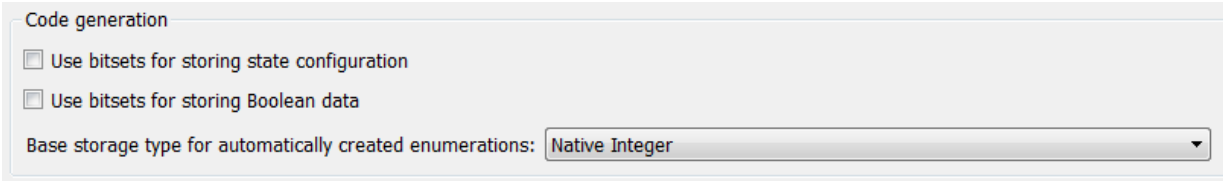
Recommended Settings

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

Optimization Parameters: Stateflow

Optimization Pane: Stateflow

When Simulink Coder is installed on your system, the **Optimization** > **Stateflow** pane includes the following parameters:



- "Use bitsets for storing state configuration" on page 6-4
- "Use bitsets for storing Boolean data" on page 6-6
- "Base storage type for automatically created enumerations" on page 6-8

See Also

Related Examples

- "Optimization Pane: General" on page 4-2
- "Optimize Generated Code" (Stateflow)

Optimization Pane: Stateflow Tab Overview

Set up optimizations for a model's active configuration set.

Tips

- To open the Optimization: Stateflow pane, in the Simulink Editor, select **Simulation > Model Configuration Parameters > Optimization > Stateflow**.
- Simulink Coder optimizations appear only when the Simulink Coder product is installed on your system.

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

- "Optimize Generated Code" (Stateflow)
- "Optimization Pane: Stateflow" on page 6-2

Use bitsets for storing state configuration

Description

Use bitsets to reduce the amount of memory required to store state configuration variables.

Category: Optimization

Settings

Default: Off

 On

Stores state configuration variables in bitsets. Potentially reduces the amount of memory required to store the variables. Potentially requires more instructions to access state configuration, which can result in less optimal code.

 Off

Stores state configuration variables in unsigned bytes. Potentially increases the amount of memory required to store the variables. Potentially requires fewer instructions to access state configuration, which can result in more optimal code.

Tips

- Selecting this check box can significantly reduce the amount of memory required to store the variables. However, it can increase the amount of memory required to store target code if the target processor does not include instructions for manipulating bitsets.
- Select this check box for Stateflow charts that have a large number of sibling states at a given level of the hierarchy.
- Clear this check box for Stateflow charts with a small number of sibling states at a given level of the hierarchy.

Dependency

This parameter requires a Simulink Coder license.

Command-Line Information

Parameter: `StateBitsets`
Value: `'on' | 'off'`
Default: `'off'`

Recommended Settings

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

See Also

Related Examples

- “Optimize Generated Code” (Stateflow)
- “Optimization Pane: Stateflow” on page 6-2

Use bitsets for storing Boolean data

Description

Use bitsets to reduce the amount of memory required to store Boolean data.

Category: Optimization

Settings

Default: Off

 On

Stores Boolean data in bitsets. Potentially reduces the amount of memory required to store the data. Potentially requires more instructions to access the data, which can result in less optimal code.

 Off

Stores Boolean data in unsigned bytes. Potentially increases the amount of memory required to store the data. Potentially requires fewer instructions to access the data, which can result in more optimal code.

Tips

- Select this check box for Stateflow charts that reference Boolean data infrequently.
- Clear this check box for Stateflow charts that reference Boolean data frequently.

Dependency

This parameter requires a Simulink Coder license.

Command-Line Information

Parameter: `DataBitsets`
Value: `'on' | 'off'`
Default: `'off'`

Recommended Settings

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

See Also

Related Examples

- “Optimize Generated Code” (Stateflow)
- “Optimization Pane: Stateflow” on page 6-2

Base storage type for automatically created enumerations

Description

Set the storage type and size for enumerations created with active state output.

Category: Optimization

Settings

Default: `'Native Integer'`

`'Native Integer'`
: Default target integer type

`int32`
: 32 bit signed integer type

`int16`
: 16 bit signed integer type

`int8`
: 8 bit signed integer type

`uint16`
: 16 bit unsigned integer type

`uint8`
: 8 bit unsigned integer type

Tips

- The default `'Native Integer'` is recommended for most models.
- If you need a smaller memory footprint for the generated enumerations, set the storage type to a smaller size. The size must be large enough to hold the number of states in the chart.

Dependency

This parameter requires a Simulink Coder license.

Command-Line Information

Parameter: `ActiveStateOutputEnumStorageType`
Value: `'Native Integer' | 'int32' | 'int16' | 'int8' | 'uint16' | 'uint8'`
Default: `'Native Integer'`

See Also

Related Examples

- “Optimize Generated Code” (Stateflow)
- “Optimization Pane: Stateflow” on page 6-2

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 7-4	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 7-6	Select the diagnostic action to take when Simulink software encounters a block whose behavior depends on the frame status of a signal.
"SimState object from earlier release" on page 7-8	Use this check to report that the SimState was generated by an earlier version of Simulink.

See Also

Related Examples

- Diagnosing Simulation Errors
- Solver Diagnostics on page 12-2
- Sample Time Diagnostics on page 11-2
- Data Validity Diagnostics on page 9-2
- Type Conversion Diagnostics on page 14-2
- Connectivity Diagnostics on page 8-2
- Compatibility Diagnostics on page 7-2
- "Model Configuration Parameters: Model Referencing Diagnostics" on page 10-2

Compatibility Diagnostics Overview

Specify the diagnostic actions that Simulink software should take when it detects an incompatibility between the current version of Simulink software and the model.

Configuration

Set the parameters displayed.

Tips

- To open the Compatibility pane, in the Simulink Editor, select **Simulation** > **Model Configuration Parameters** > **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 7-2

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 7-2

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 7-2

SimState object from earlier release

Description

Use this check to report that the SimState was generated by an earlier version of Simulink.

Category: Diagnostics

Settings

Default: `error`

`warning`

Simulink will restore as much of this SimState as possible.

`error`

When Simulink detects that the SimState was generated by an earlier version of Simulink, it does not attempt to load the object.

Command-Line Information

Parameter: `SimStateOlderReleaseMsg`
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

- “Save and Restore Simulation State as SimState”
- “Model Configuration Parameters: Compatibility Diagnostics” on page 7-2

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 8-5	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 8-7	Select the diagnostic action to take when the model contains a block with an unconnected input.
"Unconnected block output ports" on page 8-9	Select the diagnostic action to take when the model contains a block with an unconnected output.
"Unconnected line" on page 8-11	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 Outport block" on page 8-13	Select the diagnostic action to take while generating a simulation target for a referenced model if any of the model's root Outport blocks is connected to a bus but does not specify a bus object (see `Simulink.Bus`).
"Element name mismatch" on page 8-15	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 8-17	Select the diagnostic action to take when Simulink software detects a virtual bus signal that is used as a mux signal.

Parameter	Description
“Non-bus signals treated as bus signals” on page 8-19	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 8-21	Repair broken selections in the Bus Selector and Bus Assignment block parameter dialogs due to upstream bus hierarchy changes.
“Invalid function-call connection” on page 8-23	Select the diagnostic action to take if Simulink software detects incorrect use of a function-call subsystem.
“Context-dependent inputs” on page 8-25	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 12-2
- Sample Time Diagnostics on page 11-2
- Data Validity Diagnostics on page 9-2
- Type Conversion Diagnostics on page 14-2
- Compatibility Diagnostics on page 7-2
- Model Referencing Diagnostics on page 10-2

Connectivity Diagnostics Overview

Configuration

Set the parameters displayed.

Tips

- To open the **Connectivity** pane, in the Simulink Editor, select **Simulation** > **Model Configuration Parameters** > **Diagnostics** > **Connectivity**.
- 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: Connectivity Diagnostics” on page 8-2

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 8-2

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: `warning`

`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: `'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 8-2

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: `warning`

`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: `'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 8-2

Unconnected 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: `warning`

`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: `'warning'`

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 8-2

Unspecified bus object at root Outport block

Description

Select the diagnostic action to take while generating a simulation target for a referenced model if any of the model's root Outport blocks is connected to a bus but does not specify a bus object (see `Simulink.Bus`).

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.

Command-Line Information

Parameter: `RootOutportRequireBusObject`
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
- Outport
- “Model Configuration Parameters: Connectivity Diagnostics” on page 8-2

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.
- In a Bus Creator block, you can enforce strong data typing. See the tips section for the Bus Creator `Override bus signal names from inputs` 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 8-2

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 siganal 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 2-79
- "Check virtual bus inputs to blocks"
- "Model Configuration Parameters: Connectivity Diagnostics" on page 8-2

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 performing 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

Application	Setting
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 8-2

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

- “Nest Buses”
- Diagnosing Simulation Errors
- “Bus-Capable Blocks”
- “Model Configuration Parameters: Connectivity Diagnostics” on page 8-2
- “Model Configuration Parameters: Connectivity Diagnostics” on page 8-2

Invalid function-call connection

Description

Select the diagnostic action to take if Simulink detects incorrect use of a function-call subsystem.

Category: Diagnostics

Settings

Default: `error`

`warning`

Simulink displays a warning.

Note This option will be depreciated in future releases.

`error`

Simulink terminates the simulation and displays an error message.

Tips

- See the "Function-call subsystems" examples in the Simulink Subsystem Semantics library for examples of invalid uses of function-call subsystems.
- Setting this parameter to `warning` can lead to invalid simulation results.
- Setting this parameter to `warning` may cause Simulink to insert extra delay operations.

Command-Line Information

Parameter: `InvalidFcnCallConnMsg`
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

- Diagnosing Simulation Errors
- Subsystem Semantics
- “Model Configuration Parameters: Connectivity Diagnostics” on page 8-2

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 the "Function-call systems" examples in the Simulink "Subsystem Semantics" library).
- 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 15-26
- Subsystem Semantics
- Subsystem
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Connectivity Diagnostics” on page 8-2

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 9-7	Select how Simulink software resolves signals and states to `Simulink.Signal` objects.
"Division by singular matrix" on page 9-10	Select the diagnostic action to take if the Product block detects a singular matrix while inverting one of its inputs in matrix multiplication mode.
"Underspecified data types" on page 9-12	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 9-15	Select the diagnostic action to take when signals exceed specified minimum or maximum values.
"Wrap on overflow" on page 9-17	Select the diagnostic action to take if the value of a signal overflows the signal data type and wraps around.
"Saturate on overflow" on page 9-19	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.

Parameter	Description
“Inf or NaN block output” on page 9-21	Select the diagnostic action to take if the value of a block output is `Inf` or `NaN` at the current time step.
“"rt" prefix for identifiers” on page 9-23	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`.
“Detect downcast” on page 9-25	Select the diagnostic action to take when a parameter downcast occurs during simulation.
“Detect overflow” on page 9-27	Select the diagnostic action to take if a parameter overflow occurs during simulation.
“Detect underflow” on page 9-29	Select the diagnostic action to take when a parameter underflow occurs during simulation.
“Detect precision loss” on page 9-31	Select the diagnostic action to take when parameter precision loss occurs during simulation.
“Detect loss of tunability” on page 9-33	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 9-35	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 9-37	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.

Parameter	Description
"Detect write after write" on page 9-39	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 9-41	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 9-43	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 2-65	Enable live streaming of selected signals to Simulation Data Inspector
"Model Verification block enabling" on page 2-67	Enable model verification blocks in the current model either globally or locally.
"Detect multiple driving blocks executing at the same time step" on page 2-77	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.
"Underspecified initialization detection" on page 2-79	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 2-152	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.

Parameter	Description
"Detect non-reused custom storage classes" on page 2-154	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 12-2
- Sample Time Diagnostics on page 11-2
- Type Conversion Diagnostics on page 14-2
- Connectivity Diagnostics on page 8-2
- Compatibility Diagnostics on page 7-2
- Model Referencing Diagnostics on page 10-2

Data Validity Diagnostics Overview

Specify what diagnostic action Simulink software should take, if any, when it detects a condition that could compromise the integrity of data defined by the model, as well as the Data Validity parameters that pertain to code generation, and are used to debug a model.

Configuration

Set the parameters displayed.

Tips

- To open the **Data Validity** pane, in the Simulink Editor, select **Simulation** > **Model Configuration Parameters** > **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 9-2

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 (see Signal Properties Dialog Box on page 19-2) to specify explicit resolution for signals.
- 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
`'None'`	`None`
`'UseLocalSettings'`	`Explicit only`
`'TryResolveAll'`	`Explicit and implicit`
`'TryResolveAllWithWarning'`	`Explicit and warn implicit`

Recommended Settings

Application	Setting
Debugging	`Explicit only` or `None`
Traceability	`Explicit only` or `None`
Efficiency	`Explicit only` or `None`

Application	Setting
Safety precaution	`Explicit only` or `None`

See Also

`Simulink.Signal`

Related Examples

- Diagnosing Simulation Errors
- Signal Properties Dialog Box on page 19-2
- Discrete-Time Integrator
- "Model Configuration Parameters: Data Validity Diagnostics" on page 9-2

Division by singular matrix

Description

Select the diagnostic action to take if the Product 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, select **Analysis > Model Advisor**.
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
- “Model Configuration Parameters: Data Validity Diagnostics” on page 9-2

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 a

data type to inherit. However, the model uses heuristic rules to determine the most appropriate type to use, `uint8`.

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 4-13
- Diagnosing Simulation Errors
- “Use single Data Type as Default for Underspecified Types” (Embedded Coder)
- “Model Configuration Parameters: Data Validity Diagnostics” on page 9-2

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 “Signal Ranges”.
- When **Simulation range checking** is enabled, Simulink performs signal range checking at every time step during a simulation. Setting this diagnostic to `warning` or `error` can cause a decrease in simulation performance.
- 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

- “Signal Ranges”
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Data Validity Diagnostics” on page 9-2

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 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 9-21 for more information.)
- 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, select **Analysis > Model Advisor**.

2 Select **By Task**.

3 Run the **Check diagnostic settings ignored during accelerated model reference simulation** check.

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 9-2

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 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 9-21 for more information.)

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 9-2

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 9-17 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, select **Analysis > Model Advisor**.

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 9-2

"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 9-2

Detect downcast

Description

Select the diagnostic action to take when a parameter downcast 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 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.

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 9-2

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 9-2

Detect underflow

Description

Select the diagnostic action to take when a parameter underflow 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

- 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 value to the data type causes the parameter's modeled value to become zero, and therefore to differ from its ideal value.

Command-Line Information

Parameter: `ParameterUnderflowMsg`
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: Data Validity Diagnostics” on page 9-2

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 9-2

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

- 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
- “Preservation of Expressions” (Simulink Coder)
- “Model Configuration Parameters: Data Validity Diagnostics” on page 9-2

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, select **Analysis > Model Advisor**.
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 9-2

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, select **Analysis > Model Advisor**.
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 9-2

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, select **Analysis > Model Advisor**.
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 9-2

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 9-2

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

Application	Setting
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 9-2

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 10-5	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 10-7	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 10-9	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 10-14	Select the diagnostic action to take if this model contains To Workspace blocks or Scope, blocks with data logging enabled.

See Also

Related Examples

- "Model Referencing"
- Diagnosing Simulation Errors
- Solver Diagnostics on page 12-2
- Sample Time Diagnostics on page 11-2
- Data Validity Diagnostics on page 9-2
- Type Conversion Diagnostics on page 14-2

- Connectivity Diagnostics on page 8-2
- Compatibility Diagnostics on page 7-2

Model Referencing Diagnostics Overview

Specify the diagnostic actions that Simulink software should take when it detects an incompatibility relating to a model reference hierarchy.

Configuration

Set the parameters displayed.

Tips

- To open the Diagnostics: Model Referencing pane, in the Simulink Editor, select **Simulation > Model Configuration Parameters > Diagnostics > Model Referencing**.
- 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: Model Referencing Diagnostics” on page 10-2

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 referenced model's current version.

Category: Diagnostics

Settings

Default: `none`

`none`

Simulink software takes no action.

`warning`

Simulink software displays a warning and refreshes the Model block.

`error`

Simulink software displays an error message and does not refresh Model block.

Tip

If you have enabled display of referenced model version numbers on Model blocks for this model (see “Display Version Numbers”), Simulink software displays a version mismatch on the Model block icon, 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
Safety precaution	No recommendation

See Also

Related Examples

- “Model Referencing”
- Diagnosing Simulation Errors
- “Display Version Numbers”
- “Model Configuration Parameters: Model Referencing Diagnostics” on page 10-2

Port and parameter mismatch

Description

Select the diagnostic action to take if Simulink software detects a port or parameter mismatch during model loading or updating.

Category: Diagnostics

Settings

Default: `none`

`none`

Simulink software takes no action.

`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.

Tips

- Port mismatches occur when there is a mismatch between the I/O ports of a Model block and the root-level I/O ports of the model it references.
- Parameter mismatches occur when there is a mismatch between the parameter arguments recognized by the Model block and 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's Simulink Editor, select **Display > Blocks > Block I/O Mismatch for Referenced Models**.

Command-Line Information

Parameter: `ModelReferenceIOMismatchMessage`
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 Referencing”
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Model Referencing Diagnostics” on page 10-2

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`

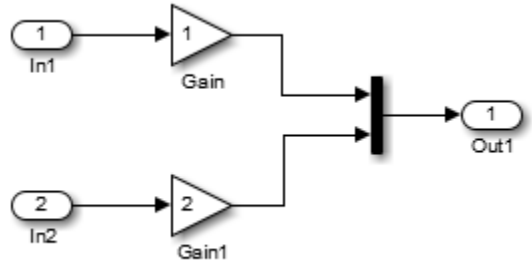
Simulink software warns you that a connection constraint has been violated and attempts to satisfy the constraint by inserting hidden blocks.

`error`

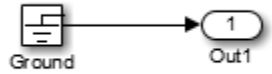
Simulink software terminates the simulation or code generation and displays an error message.

Tips

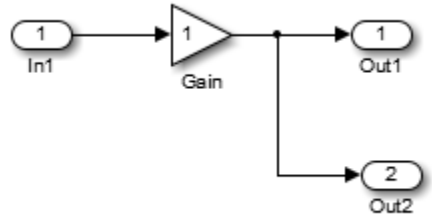
- In some cases (such as function-call feedback loops), automatically inserted hidden blocks may introduce delays and thus may change simulation results.
- Auto-inserting hidden blocks to eliminate root I/O problems stops at subsystem boundaries. Therefore, you may need to manually modify models with 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:



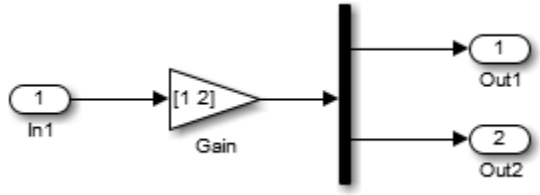
- A root Output port is connected to a Ground block:



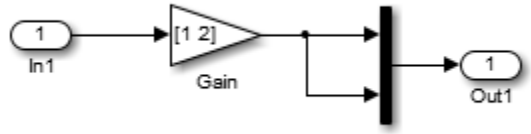
- Two root Outport blocks are connected to the same block port:



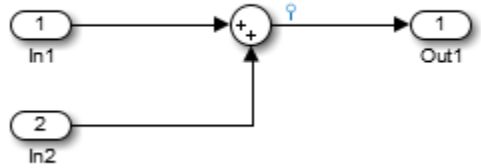
- An Outport block is connected to some elements of a block output and not others:



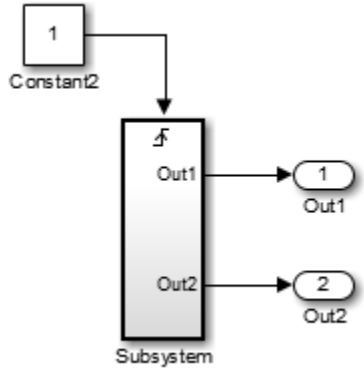
- An Outport block is connected more than once to the same element:



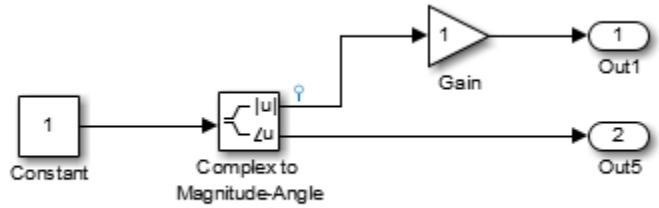
- The signal driving the root outport is a test point:



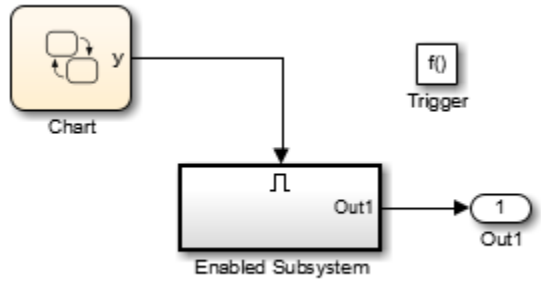
- The output port has a constant sample time, but the driving block has a non-constant sample time:



- 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 outport 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

Application	Setting
Safety precaution	`error`

See Also

Related Examples

- “Model Referencing”
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Model Referencing Diagnostics” on page 10-2

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 Referencing”
- Diagnosing Simulation Errors
- “Models with Model Referencing: Overriding Signal Logging Settings”
- To Workspace
- Scope,
- “Model Configuration Parameters: Model Referencing Diagnostics” on page 10-2

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 11-5	Select the diagnostic action to take if a source block (such as a Sine Wave block) specifies a sample time of -1.
"Multitask rate transition" on page 11-7	Select the diagnostic action to take if an invalid rate transition occurred between two blocks operating in multitasking mode.
"Single task rate transition" on page 11-9	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 11-11	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 11-13	Select the diagnostic action to take if Simulink software detects two tasks with equal priority that can preempt each other in the target system.
"Enforce sample times specified by Signal Specification blocks" on page 11-15	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.
"Sample hit time adjusting" on page 11-17	Select the diagnostic action to take if Simulink software makes a minor adjustment to a sample hit time while running the model.
"Unspecified inheritability of sample time" on page 11-19	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 12-2
- Data Validity Diagnostics on page 9-2
- Type Conversion Diagnostics on page 14-2
- Connectivity Diagnostics on page 8-2
- Compatibility Diagnostics on page 7-2
- Model Referencing Diagnostics on page 10-2

Sample Time Diagnostics Overview

Specify what diagnostic actions Simulink software should take, if any, when it detects a compilation error related to model sample times.

Configuration

Set the parameters displayed.

Tips

- To open the Sample Time pane, in the Simulink Editor, select **Simulation** > **Model Configuration Parameters** > **Diagnostics** > **Sample Time**.
- 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: Sample Time Diagnostics” on page 11-2

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 System 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 11-2

Multitask rate transition

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

Application	Setting
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 17-42
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Sample Time Diagnostics” on page 11-2

Single task rate transition

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.

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 17-42
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Sample Time Diagnostics” on page 11-2

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 17-42
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Sample Time Diagnostics” on page 11-2

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 11-2

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.

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
Efficiency	No impact
Safety precaution	`error`

See Also

Related Examples

- Diagnosing Simulation Errors
- Signal Specification
- “Model Configuration Parameters: Sample Time Diagnostics” on page 11-2

Sample hit time adjusting

Description

Select the diagnostic action to take if Simulink software makes a minor adjustment to a sample hit time while running the model.

Category: Diagnostics

Settings

Default: `none`

`none`

Simulink software takes no action.

`warning`

Simulink software displays a warning.

Tips

- Simulink software might change a sample hit time if that hit time is close to the hit time for another task. If Simulink software considers the difference to be due only to numerical errors (for example, precision issues or roundoff errors), it changes the sample hits of the faster task or tasks to exactly match the time of the slowest task that has that hit.
- Over time, these sample hit changes might cause a discrepancy between the numerical simulation results and the actual theoretical results.
- When this option is set to `warning`, the MATLAB Command Window displays a warning like the following when Simulink software detects a change in the sample hit time:

  ```
  Warning: Timing engine warning: Changing the hit time for ...
  ```

Command-Line Information

Parameter: `TimeAdjustmentMsg`
Value: `'none' | 'warning'`

Default: `'none'`

Recommended Settings

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

See Also

Related Examples

- Diagnosing Simulation Errors
- Solver Diagnostics on page 12-2
- “Model Configuration Parameters: Sample Time Diagnostics” on page 11-2

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 17-61.

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 17-61
- Solver Diagnostics on page 12-2
- “Model Configuration Parameters: Sample Time Diagnostics” on page 11-2

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 12-6	Select the diagnostic action to take if Simulink software detects an algebraic loop while compiling the model.
"Minimize algebraic loop" on page 12-8	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 12-10	Select the diagnostic action to take if Simulink software detects a block priority specification error.
"Min step size violation" on page 12-12	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 12-14	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 12-16	Select the diagnostic action to take if Simulink software changes a solver parameter setting.
"Extraneous discrete derivative signals" on page 12-18	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 12-20	Select the diagnostic action to take when a name is used for more than one state in the model.

Parameter	Description
"SimState interface checksum mismatch" on page 12-22	Use this check to ensure that the interface checksum is identical to the model checksum before loading the SimState.

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

Parameter	Description
"Allow symbolic dimension specification" on page 2-140	Specify whether Simulink propagates dimension symbols throughout the model and preserves these symbols in the propagated signal dimensions.
"Allowed unit systems" on page 2-54	Specify unit systems allowed in the model.
"Units inconsistency messages" on page 2-56	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 2-57	Allow automatic unit conversions in the model.
"Check runtime output of execution context" on page 2-69	Specify whether to display a warning if Simulink software detects potential output differences from previous releases.
"Check undefined subsystem initial output" on page 2-73	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 2-82	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.
"Block diagram contains disabled library links" on page 2-84	Select the diagnostic action to take when saving a model containing disabled library links.

Parameter	Description
"Block diagram contains parameterized library links" on page 2-86	Select the diagnostic action to take when saving a model containing parameterized library links.
"InitInArrayFormatMsg" on page 2-88	Message behavior when the initial state is an array
"Combine output and update methods for code generation and simulation" on page 2-156	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.

See Also

Related Examples

- "Algebraic Loops"
- Diagnosing Simulation Errors
- Sample Time Diagnostics on page 11-2
- Data Validity Diagnostics on page 9-2
- Type Conversion Diagnostics on page 14-2
- Connectivity Diagnostics on page 8-2
- Compatibility Diagnostics on page 7-2
- Model Referencing Diagnostics on page 10-2

Solver Diagnostics Overview

Specify what diagnostic actions Simulink software should take, if any, when it detects an abnormal condition with the solver.

Configuration

Set the parameters displayed.

Tips

- To open the **Diagnostics** pane, in the Simulink Editor, select **Simulation** > **Model Configuration Parameters** > **Diagnostics**.
- 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: Diagnostics” on page 12-2

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`

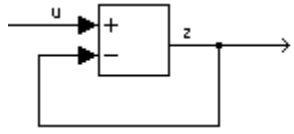
When Simulink software detects an algebraic loop, it displays a warning and tries to solve the algebraic loop. If the software cannot solve the algebraic loop, it reports an error and the simulation terminates.

`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 Loops 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 Loops
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Diagnostics” on page 12-2

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
Safety precaution	`error`

See Also

Related Examples

- Minimizing Artificial Algebraic Loops Using Simulink
- Diagnosing Simulation Errors
- Working with Test Points
- “Model Configuration Parameters: Diagnostics” on page 12-2

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 12-2

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 17-25 and “Maximum order” on page 17-34 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 17-38).

Command-Line Information

Parameter: `MinStepSizeMsg`
Value: `'warning' | 'error'`
Default: `'warning'`

Recommended Settings

Application	Setting
Debugging	No impact

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

See Also

Related Examples

- “Min step size” on page 17-25
- “Maximum order” on page 17-34
- “Number of consecutive min steps” on page 17-38
- “Purely Discrete Systems”
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Diagnostics” on page 12-2

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 17-50
- “Number of consecutive zero crossings” on page 17-55
- “Time tolerance” on page 17-52
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Diagnostics” on page 12-2

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: `'warning'`

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 12-2

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 12-2

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

Application	Setting
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 Runtime Data from Simulation”
- “Model Configuration Parameters: Diagnostics” on page 12-2

SimState interface checksum mismatch

Description

Use this check to ensure that the interface checksum is identical to the model checksum before loading the SimState.

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

- “Save and Restore Simulation State as SimState”
- “Model Configuration Parameters: Diagnostics” on page 12-2

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 13-6	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 13-8	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
"Invalid input data access in chart initialization" on page 13-10	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
"No unconditional default transitions" on page 13-12	Select the diagnostic action to take when a chart does not have an unconditional default transition to a state or a junction.
"Transition outside natural parent" on page 13-14	Select the diagnostic action to take when a chart contains a transition that loops outside of the parent state or junction.

Parameter	Description
"Undirected event broadcasts" on page 13-16	Select the diagnostic action to take when a chart contains undirected local event broadcasts.
"Transition action specified before condition action" on page 13-18	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 13-20	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 13-22	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 13-24	Select the diagnostic action to take when you can remove a self-transition on a leaf state.
"Execute-at-Initialization disabled in presence of input events" on page 13-26	Select the diagnostic action to take when Stateflow detects triggered or enabled charts that are not running at initialization.
"Use of machine-parented data instead of Data Store Memory" on page 13-28	Select the diagnostic action to take when Stateflow detects machine-parented data that can replace with chart-parented data of scope Data Store Memory.
"Unreachable execution path" on page 13-30	Select the diagnostic action to take when there are chart constructs not on a valid execution path.

See Also

Related Examples

- "Model Parameters"

- Diagnosing Simulation Errors
- Solver Diagnostics on page 12-2
- Sample Time Diagnostics on page 11-2
- Data Validity Diagnostics on page 9-2
- Type Conversion Diagnostics on page 14-2
- Connectivity Diagnostics on page 8-2
- Compatibility Diagnostics on page 7-2
- Model Referencing Diagnostics on page 10-2

Stateflow Diagnostics Overview

Specify the diagnostic actions to take for detection of unwanted chart designs.

Configuration

Set the parameters displayed.

Tips

- To open the Stateflow pane, in the Simulink Editor, select **Simulation** > **Model Configuration Parameters** > **Diagnostics** > **Stateflow**.
- 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: Stateflow Diagnostics” on page 13-2

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

- “Diagnostic for Detecting Unused Events” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 13-2

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

- “Best Practices for Creating Flow Charts” (Stateflow)
- “Backtrack in Flow Charts” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 13-2

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.

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)
Safety precaution	`error`

See Also

Related Examples

- “Execution of a Chart at Initialization” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 13-2

No unconditional default transitions

Description

Select the diagnostic action to take when a chart does not have an unconditional default transition to a state or a junction.

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

Application	Setting
Efficiency	No impact (for simulation) `none` (for production code generation)
Safety precaution	`error`

See Also

Related Examples

- “State Inconsistencies in a Chart” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 13-2

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: `SFTransitionOutsideNaturalParentDiag`
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 13-2

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`

Application	Setting
Safety precaution	`error`

See Also

Related Examples

- “Guidelines for Avoiding Unwanted Recursion in a Chart” (Stateflow)
- “Broadcast Events to Synchronize States” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 13-2

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`

Application	Setting
Safety precaution	`warning`

See Also

Related Examples

- “Transition Action Types” (Stateflow)
- “Transitions” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 13-2

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	`error`
Traceability	`error`
Efficiency	`error`
Safety precaution	`error`

See Also

Related Examples

- “Design Considerations for Moore Charts” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 13-2

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`

Application	Setting
Safety precaution	`error`

See Also

Related Examples

- “Set Stateflow Block Update Method” (Stateflow)
- “Control Chart Execution Using Temporal Logic” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 13-2

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 13-2

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

Application	Setting
Efficiency	No impact
Safety precaution	`error`

See Also

Related Examples

- “Model Configuration Parameters: Stateflow Diagnostics” on page 13-2

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.

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: `SFMachineParentedDataDiag`
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

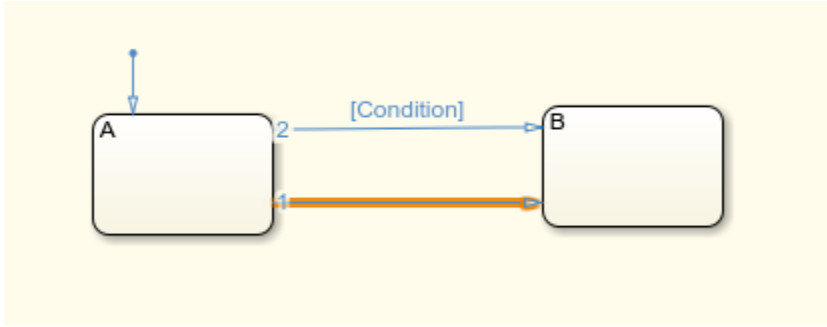
- “Best Practices for Using Data in Charts” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 13-2

Unreachable execution path

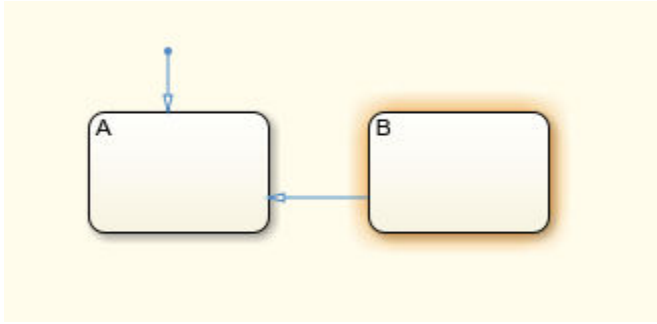
Description

Select the diagnostic action to take when there are chart constructs not on a valid execution path. These constructs can cause unreachable execution paths:

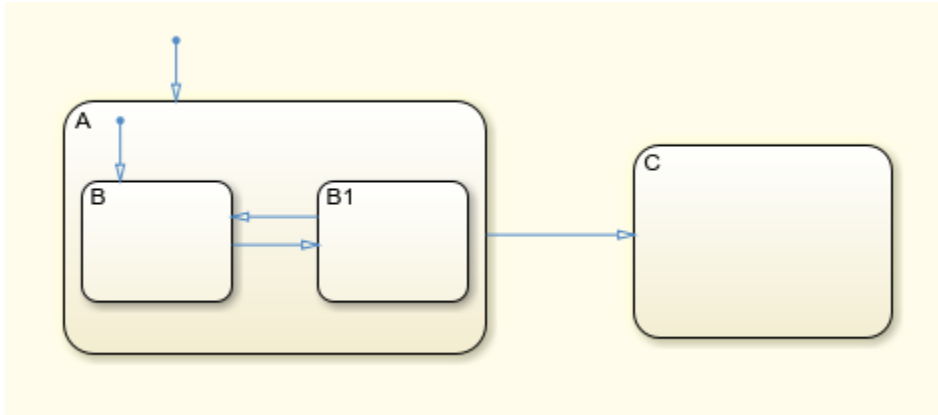
- Dangling transitions not connected to a destination object.
- Transition shadowing caused by an unconditional transition originating from a source that prevents other transitions from the same source from executing.



- States or junctions not connected as a destination from a valid transition.



- Unconditional paths out of states. In this chart, initially state `A` and state `B` are active. The chart then takes the unconditional transition to state `C`, and state `C` becomes active. The transition to state `B1` does not execute and state `B1` is unreachable.



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

Recommended Settings

Application	Setting
Debugging	`warning`

Application	Setting
Traceability	No impact
Efficiency	No impact (for simulation) `none` (for production code generation)
Safety precaution	`error`

See Also

Related Examples

- “Detection of Transition Shadowing” (Stateflow)
- “Model Configuration Parameters: Stateflow Diagnostics” on page 13-2
- “Modeling Rules That Stateflow Detects During Edit Time” (Stateflow)

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 14-5	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 14-7	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 14-9	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 14-11	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 14-13	Specifies diagnostic action to take when a fixed-point constant precision loss occurs during simulation.
"Detect overflow" on page 14-15	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 12-2
- Sample Time Diagnostics on page 11-2
- Data Validity Diagnostics on page 9-2
- Connectivity Diagnostics on page 8-2
- Compatibility Diagnostics on page 7-2
- Model Referencing Diagnostics on page 10-2

Type Conversion Diagnostics Overview

Specify the diagnostic actions that Simulink software should take when it detects a data type conversion problem while compiling the model.

Configuration

Set the parameters displayed.

Tips

- To open the Type Conversion pane, in the Simulink Editor, select **Simulation** > **Model Configuration Parameters** > **Diagnostics** > **Type Conversion**.
- 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: Type Conversion Diagnostics” on page 14-2

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 14-2

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 14-2

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 14-2

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
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 14-2

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
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 14-2

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
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 14-2

Model Referencing Parameters

Model Configuration Parameters: Model Referencing

The **Model Referencing** category includes parameters for configuring the inclusion of other models (Model blocks).

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

Parameter	Description
"Rebuild" on page 15-5	Select the method used to determine when to rebuild simulation and Simulink Coder targets for referenced models before updating, simulating, or generating code from this model.
"Never rebuild diagnostic" on page 15-16	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 15-18	Specify whether to use automatic parallel building of the model reference hierarchy whenever possible.
"MATLAB worker initialization for builds" on page 15-20	Specify how to initialize MATLAB workers for parallel builds.
"Enable strict scheduling checks for referenced models" on page 15-22	This parameter enables these checks for referenced models: • 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 15-24	Specify how many references to this model can occur in another model.
"Pass fixed-size scalar root inputs by value for code generation" on page 15-26	Specify whether a model that calls (references) this model passes its scalar inputs to this model by value.

Parameter	Description
“Minimize algebraic loop occurrences” on page 15-29	Try to eliminate artificial algebraic loops from a model that involve the current referenced model
“Propagate all signal labels out of the model” on page 15-31	Pass propagated signal names to output signals of Model block.
“Propagate sizes of variable-size signals” on page 15-34	Select how variable-size signals propagate through referenced models.
“Model dependencies” on page 15-37	You can add user-created dependencies to the set of known target dependencies by using the **Model dependencies** parameter.

See Also

Related Examples

- “Overview of Model Referencing”
- Model Dependencies

Model Referencing Pane Overview

Specify the options for including other models in this model, this model in other models, and for building simulation and code generation targets.

Configuration

Set the parameters displayed.

Tips

- To open the Model Referencing pane, in the Simulink Editor, select **Simulation** > **Model Configuration Parameters** > **Model Referencing**.
- The Model Referencing pane 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 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: Model Referencing” on page 15-2

Rebuild

Description

Select the method used to determine when to rebuild simulation and Simulink Coder targets for referenced models before updating, simulating, or generating code from this model.

There are four rebuild options. Two options, `Always` and `Never`, either always rebuild the model reference target or never rebuild the target, respectively. The other two options, `If any changes detected` and `If any changes in known dependencies detected`, cause Simulink to check the model and its dependencies to determine whether or not to rebuild the model reference target. As part of this checking, Simulink:

- Automatically identifies a set of "known" target dependencies that it examines for changes.
- May compute the model's structural checksum, which reflects changes to the model that can affect simulation results.

For additional background information to help you determine which rebuild option setting to use, see the "Definitions" and "Tips" sections.

Category: Model Referencing

Settings

Default: `If any changes detected`

`Always`

Always rebuild targets referenced by this model before simulating, updating, or generating code from this model.

`If any changes detected`

Rebuild a target for a referenced model if Simulink detects a change that could affect simulation results. To do this, Simulink first looks for changes to the target dependencies and to the model, and, if none are found, it then computes the structural checksum of the model to check that the model reference target is up to date.

`If any changes in known dependencies detected`

Rebuild a target if Simulink detects a change in target dependencies or in both the model and its structural checksum. If Simulink does not detect a change in target dependencies or the model, it does *not* compute the structural checksum of the model and does *not* rebuild the model reference target. You must list all user-created dependencies in the **Configuration Parameters > Model Referencing > Model dependencies** parameter.

`Never`

Never rebuild targets referenced by this model before simulating, updating, or generating code from this model.

Definitions

Known target dependencies

Known target dependencies are files and data outside of model files that Simulink examines for changes when checking to see if a model reference target is up to date. Simulink automatically computes a set of known target dependencies. Simulink examines the known target dependencies to determine whether they have changed, which it can do quickly. 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** on page 15-37 parameter
- External files used by Stateflow, a MATLAB Function block, or a MATLAB System block

Potential target dependencies

Potential dependencies are files and data outside of model files, as well as model configuration settings, that Simulink examines for changes when checking to see if a model reference target is up to date. Simulink automatically computes the set of potential dependencies. Simulink examines the potential dependencies, which it can do quickly. Examples of potential dependencies are:

- Changes to global variables
- Changes to targets of models referenced by this model

- The **Configuration Parameters > Diagnostics > Data Validity > Signal resolution** parameter is set to either `Explicit and warn implicit` or `Explicit and implicit`.

Simulink examines each potential target dependency to determine whether the state of that dependency is a trigger for causing a structural checksum check.

User-created dependencies

Although Simulink automatically examines every known target dependency, you can have files that can impact the simulation results of your model that Simulink does not automatically identify. Some 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

As part of determining whether a model reference target is up to date, Simulink may compute the structural checksum of a model, which reflects changes to 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. As a result, the structural checksum reflects changes to the model that can affect simulation results, including 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 the `Simulink.BlockDiagram.getChecksum` documentation.

Tips

- You do not need to have the same rebuild option setting for every model in a model reference hierarchy. When you simulate, update, or generate code for a model, the rebuild option setting for that model applies to all models that it references.
- To improve rebuild detection speed and accuracy, use the **Model dependencies** on page 15-37 parameter to specify user-created dependencies. If you use the `If any`

`changes in known dependencies detected` rebuild option, then specify all user-created dependencies for your model in the **Model dependencies** on page 15-37 parameter.

- Each rebuild option setting has benefits and limitations, depending on your rebuild goal. The following table lists the options in the order of the thoroughness of rebuild detection. For detailed information about how Simulink determines whether a model reference target is out of date, see the Change Detection Processing table, which is part of the next tip.

Benefits and Limitations of Each Option

Rebuild Goal	Rebuild Option Setting	Notes
Make all the model reference targets up to date.	`Always`	Requires the most processing time. Can trigger unnecessary builds before simulating, updating, or generating code from a referenced model. Before you deploy a model, use the `Always` setting.
Perform extensive detection of changes to dependencies of the referenced models.	`If any changes detected`	Default. Reduces the number of rebuilds, compared to the `Always` setting. Detects changes in the dependencies of the target, as well as changes in the structural checksum of the referenced model. The structural checksum can detect changes that occur in user-created dependencies that are not specified with the **Model dependencies** on page 15-37 parameter.

Rebuild Goal	Rebuild Option Setting	Notes
Reduce time required for rebuild detection.	`If any changes in known dependencies detected`	Reduces the number of rebuilds, compared to the `If any changes detected` option. Ignores cosmetic changes, such as annotation changes, in the referenced model and its libraries. Subset of the checks performed by the `If any changes detected` option. Invalid simulation results may occur if you do not specify with the **Model dependencies** on page 15-37 parameter every user-created dependency.

Rebuild Goal	Rebuild Option Setting	Notes
Avoid rebuilds during model development.	`Never`	Least amount of processing time, but requires that you ensure that the model reference targets are up to date. If you are certain that the model reference targets are up to date, you can use this option to avoid target dependency checking when simulating, updating, or generating code from a model. May lead to invalid results if referenced model targets are not in fact up to date. 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** on page 15-16 parameter. To manually rebuild model reference targets, use the `slbuild` function.

- To detect whether to perform a rebuild, Simulink uses different processing for each **Rebuild** setting. The following table summarizes the main types of change detection checks that Simulink performs.

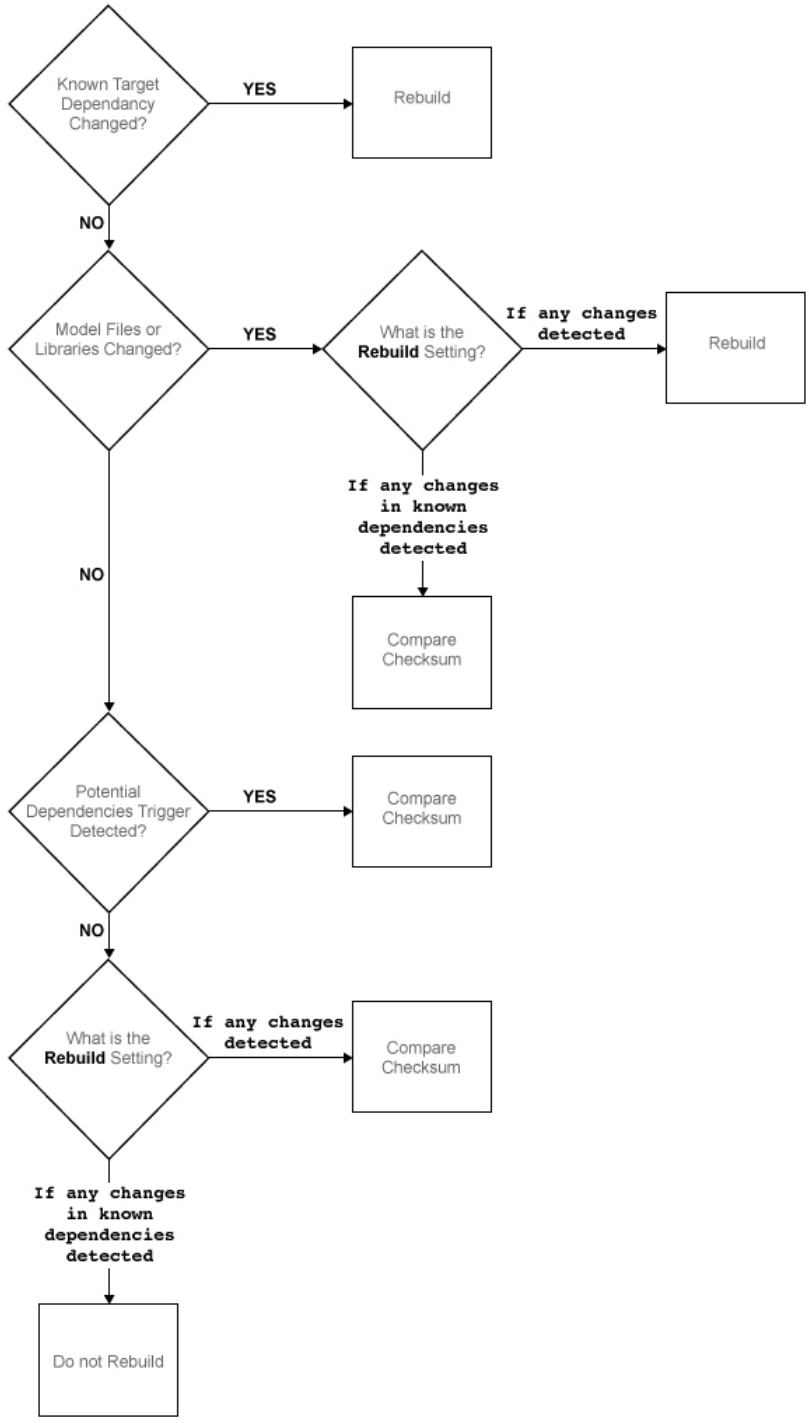
Change Detection Processing

Rebuild Option Setting	Simulink Change Detection Processing
`Always`	Does no change detection processing. Always rebuilds targets referenced by this model before simulating, updating, or generating code from this model.

`If any changes detected`
 	
and
 	
`If any changes in known dependencies detected`
	See the flow chart, below.

`Never`	Change detection processing determined by the **Never rebuild diagnostic** on page 15-16 parameter.

The following 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`. The "Compare Checksum" boxes indicate that Simulink detects whether the structural checksum has changed. If the structural checksum has changed, then Simulink performs a rebuild.


Known Target Dependancy Changed?
YES
Rebuild
NO
Model Files or Libraries Changed?
YES
What is the Rebuild Setting?
If any changes detected
Rebuild
If any changes in known dependencies detected
Compare Checksum
NO
Potential Dependencies Trigger Detected?
YES
Compare Checksum
NO
What is the Rebuild Setting?
If any changes detected
Compare Checksum
If any changes in known dependencies detected
Do not Rebuild

- The following examples illustrate differences between the `If any changes detected` and `If any changes in known dependencies detected` options.

  If you change a MATLAB file that is executed as part of a callback script (or other user-created dependency) that you have not listed in the **Model dependencies** parameter:

  - `If any changes detected` – Causes a rebuild, because the change to the file changes 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.

  If you move a block in a model:

  - `If any changes detected` – Causes a rebuild, because the model has changed.
  - `If any changes in known dependencies detected` – Does not cause a rebuild, because this change does not change the model's structural checksum.

Dependency

Selecting `Never` enables the **Never rebuild diagnostic** parameter.

Command-Line Information

Parameter: `UpdateModelReferenceTargets`
Value: `'Force' | 'IfOutOfDateOrStructuralChange' | 'IfOutOfDate' | 'AssumeUpToDate'`
Default: `'IfOutOfDateOrStructuralChange'`

`UpdateModelReferenceTargets` Value	Equivalent Rebuild Value
`'Force'`	`Always`
`'IfOutOfDateOrStructuralChange'`	`If any changes detected`
`'IfOutOfDate'`	`If any changes in known dependencies detected`
`'AssumeUpToDate'`	`Never`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	`If any changes detected` or `Never` If you use the `Never` setting, then set the **Never rebuild diagnostic** parameter to `Error if rebuild required`.

See Also

`Simulink.BlockDiagram.getChecksum`

Related Examples

- Model Dependencies
- “Model Configuration Parameters: Model Referencing” on page 15-2

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 15-5.

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

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Model Referencing” on page 15-2

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.

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

Related Examples

- “Reduce Update Time for Referenced Models”
- “Reduce Build Time for Referenced Models” (Simulink Coder)
- “Model Configuration Parameters: Model Referencing” on page 15-2

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.

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
Efficiency	No impact
Safety precaution	No impact

See Also

Related Examples

- “Reduce Update Time for Referenced Models”
- “Reduce Build Time for Referenced Models” (Simulink Coder)
- “Model Configuration Parameters: Model Referencing” on page 15-2

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

Related Examples

- “Execution Order for Function-Call Root-Level Inport Blocks”
- “Scheduling Restrictions for Referenced Export-Function Models”
- “Model Configuration Parameters: Model Referencing” on page 15-2

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 multiply referenced, 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 Models with Multiple Referenced Model Instances".

Command-Line Information

Parameter: `ModelReferenceNumInstancesAllowed`
Value: `'Zero' | 'Single' | 'Multi'`
Default: `'Multi'`

Recommended Settings

Application	Setting
Debugging	No impact

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

See Also

Related Examples

- Diagnosing Simulation Errors
- “Model Configuration Parameters: Model Referencing” on page 15-2

Pass fixed-size scalar root inputs by value for code generation

Description

Specify whether a model that calls (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 calls (references) this model passes scalar inputs to this model by value.

 Off

The calling 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. If you use the default setting of `Enable all as errors` for the **Configuration Parameters > Diagnostics > Connectivity > Context-dependent inputs** parameter, then 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 8-25.

- 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 8-25.

See Also

Related Examples

- “Using Function-Call Subsystems”
- “Generate Reentrant Code from Subsystems” (Simulink Coder)
- “Model Configuration Parameters: Model Referencing” on page 15-2

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

Application	Setting
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- Model block
- “Algebraic Loops”
- “Model Blocks and Direct Feedthrough”
- Diagnosing Simulation Errors
- “Model Configuration Parameters: Model Referencing” on page 15-2

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

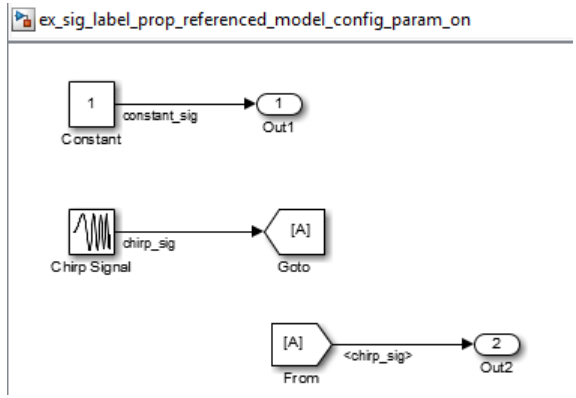
Simulink propagates signal names to output signals of the Model block.

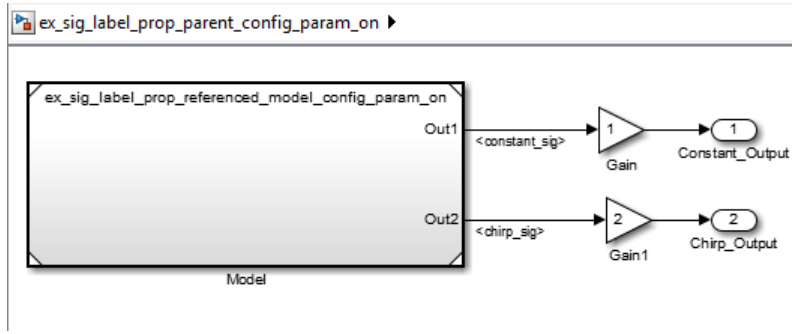
☐ Off

Simulink does not propagate signal names to output signals of the Model block.

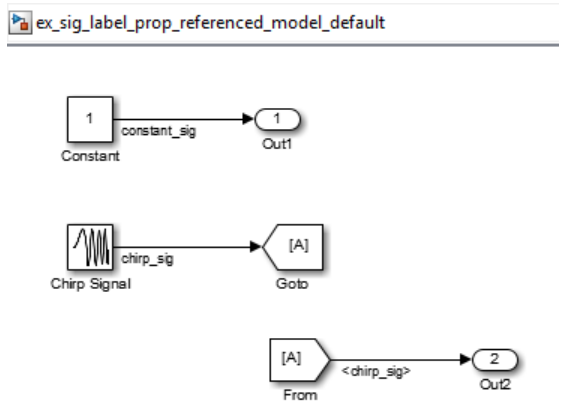
Tips

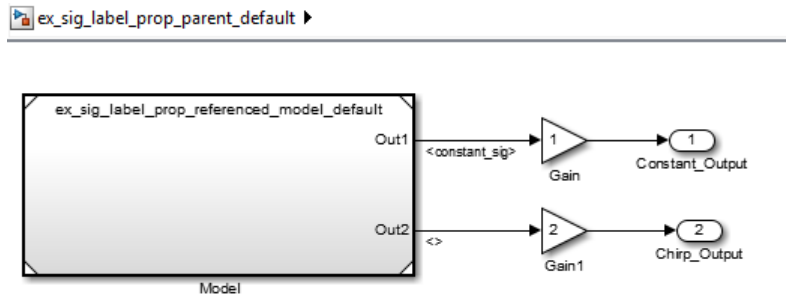
- By default, each instance of a referenced model propagates signal labels. Clear the setting for any instance that you do not want to propagate signal labels.
- The following models illustrate the behavior when you use the default setting of the **Propagate all signal labels out of the model** parameter 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
Traceability	No impact
Efficiency	No impact
Safety precaution	No recommendation

See Also

Related Examples

- Model block
- “Signal Label Propagation”
- “Model Configuration Parameters: Model Referencing” on page 15-2

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.	Switch or Enable Subsystem block whose parameter **Propagate sizes of variable-size signals** is set to `During execution`
2	States require resetting when the input signal size changes.	Unit Delay block in an Enabled Subsystem whose parameter **Propagate sizes of variable-size signals** is set to `Only when enabling`
3	Output signal size depends only on 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:

- One or more blocks in category 1, and all other blocks are in category 3, select `During execution`.

- One or more blocks in category 2, and all another blocks are in category 3, select `Only when enabling`.
- Blocks in category 1 and 2, report an error.
- All blocks in category 3 with a conditionally executed subsystem that is not an Enable, Function-Call, or Action subsystem, report an error. Simulink, in this case, cannot determine when to propagate sizes of variable-size signals.
- All blocks in category 3 with only conditionally executed subsystems that are an Enable, Function-Call, or Action subsystem, support 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

Related Examples

- “Model Configuration Parameters: Model Referencing” on page 15-2

Model dependencies

Description

Although Simulink automatically examines every known target dependency, you can have files that can impact the simulation results of your model that Simulink does not automatically identify. Some 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.

Simulink examines the files specified with the **Model dependencies** parameter when determining whether the model reference target is up to date. If the **Rebuild** on page 15-5 parameter is set to:

- `Always`, then the listed files are not examined.
- Either `If any changes detected` or `If any changes in known dependencies detected`, then changes to listed files cause the model reference target to rebuild.
- `Never`, and the **Never rebuild diagnostic** on page 15-16 parameter is set to either `Warn if rebuild required` or `Error if rebuild required`, then changes to listed files cause Simulink to report a warning or error.

Category: Model Referencing

Settings

Default: `''`

- Specify the 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, then specify the path to the dependent file, as described below.

  - 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.
- File names must include a file extensions (for example, `.m` or `.mat`)
- File names and paths can include spaces.
- You can use the following characters in the character vectors:

  - 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 model dependencies other than those that Simulink checks automatically as part of the its rebuild detection. For details, see the **Rebuild** on page 15-5 parameter documentation.
- If the **Rebuild** setting is `If any changes in known dependencies detected`, to prevent invalid simulation results, add every user-created dependency (for example, MATLAB code files or MAT-files).
- Using the Simulink Manifest Tools can help you to identify model dependencies. 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

Related Examples

- “Rebuild” on page 15-5
- “Model Configuration Parameters: Model Referencing” on page 15-2

Simulation Target Parameters

Model Configuration Parameters: Simulation Target

The **Simulation Target** category includes parameters for configuring the simulation target for a model that contains MATLAB Function blocks, Stateflow charts, or Truth Table blocks.

Parameter	Description
"Parse custom code symbols" on page 16-6	Specify whether or not to parse the custom code and report unresolved symbols in a model. This setting applies to all C charts in the model, including library link charts.
"Source file" on page 16-8	Enter code lines to appear near the top of a generated source code file.
"Header file" on page 16-9	Enter code lines to appear near the top of a generated header file.
"Initialize function" on page 16-11	Enter code statements that execute once at the start of simulation.
"Terminate function" on page 16-13	Enter code statements that execute at the end of simulation.
"Include directories" on page 16-15	Specify a list of folder paths that contain files you include in the compiled target.
"Source files" on page 16-17	Specify a list of source files to compile and link into the target.
"Libraries" on page 16-19	Specify a list of static libraries that contain custom object code to link into the target.
"Reserved names" on page 16-21	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.
"Defines" on page 16-23	Specify preprocessor macro definitions to be added to the compiler command line.

Parameter	Description
"Use local custom code settings (do not inherit from main model)" on page 2-138	Specify if a library model can use custom code settings that are unique from the main model. (This parameter is read-only)

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

Parameter	Description
"Echo expressions without semicolons" on page 2-130	Enable run-time output in the MATLAB Command Window, such as actions that do not terminate with a semicolon.
"Simulation target build mode" on page 2-136	Specifies how you build the simulation target for a model that contains MATLAB Function blocks, Stateflow charts, or Truth Table blocks.
"Ensure responsiveness" on page 2-121	Enables responsiveness checks in code generated for MATLAB Function blocks.
"Generate typedefs for imported bus and enumeration types" on page 2-134	Determines `typedef` handling and generation for imported bus and enumeration data types in Stateflow and MATLAB Function blocks.
"Ensure memory integrity" on page 2-132	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 2-124	Allow recursive functions in code that is generated for MATLAB code that contains recursive functions.
"Compile-time recursion limit for MATLAB functions" on page 2-123	For compile-time recursion, control the number of copies of a function that are allowed in the generated code.

Parameter	Description
"Dynamic memory allocation in MATLAB Function blocks" on page 2-126	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.
"Dynamic memory allocation threshold in MATLAB Function blocks" on page 2-128	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.
Allow setting breakpoints during simulation	Enable debugging and animation during simulation of a model that contains MATLAB Function blocks, Stateflow charts, State Transition blocks, or Truth Table blocks.

See Also

Related Examples

- "Speed Up Simulation" (Stateflow)

Simulation Target: General Tab Overview

Configure the simulation target for a model that contains MATLAB Function blocks, Stateflow charts, Truth Table blocks, or State Transition Table blocks.

Configuration

Set the parameters that appear.

Tip

To open the Simulation Target pane, in the Simulink Editor, select **Simulation** > **Model Configuration Parameters** > **Simulation Target**.

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: Simulation Target” on page 16-2

Parse custom code symbols

Description

Specify whether or not to parse the custom code and report unresolved symbols in a model. This setting applies to all C charts in the model, including library link charts.

Category: Simulation Target

Settings

Default: On

On

Enables parsing of custom code to report unresolved symbols in C charts of your model.

Off

Disables parsing of custom code.

This option only applies to C charts, not charts that use MATLAB as the action language.

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

- Including Custom C Code (Stateflow)
- Resolving Symbols in Stateflow Charts (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 16-2

Source file

Description

Enter code lines to appear near the top of a generated source code file.

Category: Simulation Target

Settings

Default: `''`

Code lines appear near the top of the generated *`model`*`.c` source file, outside of any function.

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

- Including Custom C Code (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 16-2

Header file

Description

Enter code lines to appear near the top of a generated header file.

Category: Simulation Target

Settings

Default: `''`

Code lines appear near the top of the generated *`model`*`.h` header file.

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

- Including Custom C Code (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 16-2

Initialize function

Description

Enter code statements that execute once at the start of simulation.

Category: Simulation Target

Settings

Default: `''`

Code appears inside the model's initialize function in the *`model`*`.c` file.

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

- Including Custom C Code (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 16-2

Terminate function

Description

Enter code statements that execute at the end of simulation.

Category: Simulation Target

Settings

Default: `''`

Code appears inside the model's terminate function in the *`model`*`.c` file.

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

- Including Custom C Code (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 16-2

Include directories

Description

Specify a list of folder paths that contain files you include in the compiled target.

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

- Including Custom C Code (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 16-2

Source files

Description

Specify a list of source files to compile and link into the target.

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 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

- Including Custom C Code (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 16-2

Libraries

Description

Specify a list of static 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.

Tip

- The file name is sufficient if the file is in the current MATLAB folder or in one of the include directories.

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

- Including Custom C Code (Stateflow)
- “Model Configuration Parameters: Simulation Target” on page 16-2

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
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 16-2

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, `-DFOO=1` and `FOO=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 16-2

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.

Parameter	Description
"Start time" on page 17-8	Specify the start time for the simulation or generated code as a double-precision value, scaled to seconds.
"Stop time" on page 17-9	Specify the stop time for the simulation or generated code as a double-precision value, scaled to seconds.
"Type" on page 17-11	Select the type of solver you want to use to simulate your model.
"Solver" on page 17-14	Select the solver you want to use to compute the states of the model during simulation or code generation.
"Max step size" on page 17-21	Specify the largest time step that the solver can take.
"Initial step size" on page 17-23	Specify the size of the first time step that the solver takes.
"Min step size" on page 17-25	Specify the smallest time step that the solver can take.
"Relative tolerance" on page 17-27	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 17-29	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.

Parameter	Description
"Shape preservation" on page 17-32	At each time step use derivative information to improve integration accuracy.
"Maximum order" on page 17-34	Select the order of the numerical differentiation formulas (NDFs) used in the `ode15s` solver.
"Solver reset method" on page 17-36	Select how the solver behaves during a reset, such as when it detects a zero crossing.
"Number of consecutive min steps" on page 17-38	Specify the maximum number of consecutive minimum step size violations allowed during simulation.
"Solver Jacobian Method" on page 17-40	Specify the method to compute the Jacobian matrix for an implicit solver.
"Treat each discrete rate as a separate task" on page 17-42	Specify whether Simulink executes blocks with periodic sample times individually or in groups.
"Automatically handle rate transition for data transfer" on page 17-44	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 17-46	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 17-48	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.

Parameter	Description
“Zero-crossing control” on page 17-50	Enables zero-crossing detection during variable-step simulation of the model. For most models, this speeds up simulation by enabling the solver to take larger time steps.
“Time tolerance” on page 17-52	Specify a tolerance factor that controls how closely zero-crossing events must occur to be considered consecutive.
“Number of consecutive zero crossings” on page 17-55	Specify the number of consecutive zero crossings that can occur before Simulink software displays a warning or an error.
“Algorithm” on page 17-57	Specifies the algorithm to detect zero crossings when a variable-step solver is used.
“Signal threshold” on page 17-59	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 17-61	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 17-64	Specify the step size used by the selected fixed-step solver.
“Sample time properties” on page 17-66	Specify and assign priorities to the sample times that this model implements.
“Extrapolation order” on page 17-69	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.

Parameter	Description
“Number Newton's iterations” on page 17-71	Specify the number of Newton's method iterations used by the `ode14x` 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 17-73	Enable concurrent tasking behavior for model.

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

Parameter	Description
“Enable decoupled continuous integration” on page 2-148	Removes the coupling between continuous and discrete rates.

See Also

Related Examples

- “Specify Simulation Start and Stop Time”
- “About Solvers”

Solver Overview

Specify the simulation start and stop time, and the solver configuration for the simulation. Use the Solver pane to set up a solver for a model's active configuration set.

A solver computes a dynamic system's states at successive time steps over a specified time span, using information provided by the model. 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.

Configuration

1. Select a solver type from the **Type** list.
2. Select a solver from the **Solver** list.
3. Set the parameters displayed for the selected type and solver combination.
4. Apply the changes.

Tips

- To open the Solver pane, in the Simulink Editor, select **Simulation** > **Model Configuration Parameters** > **Solver**.
- 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.

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

- “Solver Pane” on page 17-2

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.

Command-Line Information

Parameter: `StartTime`
Type: character vector
Value: any valid value
Default: `'0.0'`

See Also

Related Examples

- “Specify Simulation Start and Stop Time”
- “Solver Pane” on page 17-2

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”.

Command-Line Information

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”
- “Specify Simulation Start and Stop Time”
- “Solver Pane” on page 17-2

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**

Command-Line Information

Parameter: `SolverType`
Value: `'Variable-step' | 'Fixed-step'`
Default: `'Variable-step'`

See Also

Related Examples

- "Choose a Solver"

- “Purely Discrete Systems”
- “Solver Pane” on page 17-2

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 17-14
- “Variable-step Solvers” on page 17-16

The default setting for new models is `VariableStepAuto`.

Fixed-step Solvers

Default: `FixedStepAuto`

`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. 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)
```

`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.

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.

Tips

- Identifying the optimal solver for a model requires experimentation. For an in-depth discussion, see "About Solvers".
- The optimal solver balances acceptable accuracy with the shortest simulation time.
- Simulink software uses a discrete solver for any 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 `ode`*n*, as *n* increases.
- As computational complexity increases, the accuracy of the results also increases.

Dependencies

Selecting the `ode1 (Euler)`, `ode2 (Huen)`, `ode 3 (Bogacki-Shampine)`, `ode4 (Runge-Kutta)`, `ode 5 (Dormand-Prince)`, 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 `ode14x (extrapolation)` enables the following parameters:

- **Fixed-step size (fundamental sample time)**
- **Extrapolation order**
- **Number 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: `Solver`
Value: `'VariableStepAuto' | 'VariableStepDiscrete' | 'ode45' | 'ode23' | 'ode113' | 'ode15s' | 'ode23s' | 'ode23t' | 'ode23tb' | 'FixedStepAuto' | 'FixedStepDiscrete' |'ode8'| 'ode5' | 'ode4' | 'ode3' | 'ode2' | 'ode1' | 'ode14x'`
Default: `'ode45'`

See Also

Related Examples

- “Solvers”
- “About Solvers”
- “Purely Discrete Systems”
- “Solver Pane” on page 17-2

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

  $$h_{\max} = \frac{t_{stop} - t_{start}}{50}$$

- For Sine and Signal Generator source blocks, Simulink calculates the max step size using this heuristic:

  $$h_{\max} = \min\left(\frac{t_{stop} - t_{start}}{50}, \left(\frac{1}{3}\right)\left(\frac{1}{Freq_{\max}}\right) \right)$$

  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`.

Command-Line Information

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 17-2

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`.

Command-Line Information

Parameter: `InitialStep`
Type: character vector
Value: any valid value
Default: `'auto'`

Recommended Settings

Application	Setting
Debugging	No impact

Application	Setting
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 17-2

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 12-12).
- 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`.

Command-Line Information

Parameter: `MinStep`
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”
- “Min step size violation” on page 12-12
- “Solver Pane” on page 17-2

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".

Command-Line Information

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 17-2

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.

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".

Command-Line Information for Configuration Parameters

Parameter: `AbsTol`
Type: character vector | numeric value
Value: `'auto'` | positive real scalar
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

- “Error Tolerances for Variable-Step Solvers”
- “Improve Simulation Performance Using Performance Advisor”
- “Solver Pane” on page 17-2

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.

Command-Line Information

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 17-2

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`.

Command-Line Information

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 17-2

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)`

Command-Line Information

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 17-2

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 17-25).
- 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 12-12).

Dependencies

This parameter is enabled only if you set:

- Solver **Type** to `Variable-step`.
- **Solver** to a continuous variable step solver.

Command-Line Information

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 12-12
- “Min step size” on page 17-25
- “Solver Pane” on page 17-2

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.

Command-Line Information

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 17-2

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 rate transition** and **Multitask rate transition** 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.

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 17-2

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) in the Simulink Coder documentation 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.

Command-Line Information

Parameter: `AutoInsertRateTranBlk`
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

Related Examples

- “Rate Transition Block Options” (Simulink Coder)
- “Solver Pane” on page 17-2

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.

Command-Line Information

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 17-2

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.

Command-Line Information

Parameter: `PositivePriorityOrder`
Value: `'on' | 'off'`
Default: `'off'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact

Application	Setting
Safety precaution	No impact

See Also

Related Examples

- “Rate Transitions and Asynchronous Blocks” (Simulink Coder)
- “Solver Pane” on page 17-2

Zero-crossing control

Description

Enables zero-crossing detection during variable-step simulation of the model. For most models, this speeds up simulation by enabling the solver to take larger time steps.

Category: Solver

Settings

Default: `Use local settings`

`Use local settings`

Specifies that zero-crossing detection be enabled on a block-by-block basis. For a list of applicable blocks, see “Simulation Phases in Dynamic Systems”

To specify zero-crossing detection for one of these blocks, open the block's parameter dialog box and select the **Enable zero-crossing detection** option.

`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 this option 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 local zero-crossing detection setting for individual blocks.

Dependencies

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

Selecting either `Use local settings` or `Enable all` enables the following parameters:

- **Time tolerance**
- **Number of consecutive zero crossings**
- **Algorithm**

Command-Line Information

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 17-55
- “Consecutive zero-crossings violation” on page 12-14
- “Time tolerance” on page 17-52
- “Solver Pane” on page 17-2

Time tolerance

Description

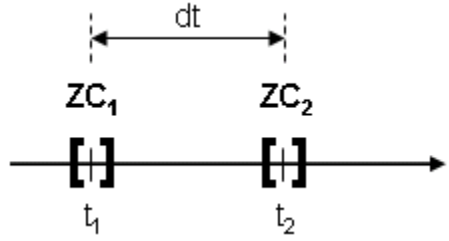
Specify a tolerance factor that controls how closely zero-crossing events must occur to be considered consecutive.

Category: Solver

Settings

Default: `10*128*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 < RelTolZC * t2`

  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 12-14).

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

This parameter is enabled only if **Zero-crossing control** is set to either `Use local settings` or `Enable all`.

Command-Line Information

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 17-55

- “Zero-crossing control” on page 17-50
- “Consecutive zero-crossings violation” on page 12-14
- “Solver Pane” on page 17-2

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 12-14).
- Simulink software defines zero crossings as consecutive if the time between events is less than a particular interval (see “Time tolerance” on page 17-52).

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

This parameter is enabled only if **Zero-crossing control** is set to either `Use local settings` or `Enable all`.

Command-Line Information

Parameter: `MaxConsecutiveZCs`
Type: character vector
Value: any valid value
Default: `'1000'`

Recommended Settings

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

See Also

Related Examples

- “Zero-Crossing Detection”
- “Zero-crossing control” on page 17-50
- “Consecutive zero-crossings violation” on page 12-14
- “Time tolerance” on page 17-52
- “Solver Pane” on page 17-2

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 17-59 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

- This parameter is enabled only if the solver **Type** is set to `Variable-step`.
- Selecting `Adaptive` enables the **Signal threshold** parameter.

Command-Line Information

Parameter: `ZeroCrossAlgorithm`
Value: `'Nonadaptive' | 'Adaptive'`
Default: `'Nonadaptive'`

Recommended Settings

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

See Also

Related Examples

- “Zero-Crossing Detection”
- “Zero-crossing control” on page 17-50
- “Consecutive zero-crossings violation” on page 12-14
- “Time tolerance” on page 17-52
- “Number of consecutive zero crossings” on page 17-55
- “Solver Pane” on page 17-2

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`

`Auto`

The signal threshold is determined automatically by the adaptive algorithm.

`String`

Use the specified 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.

Dependency

This parameter is enabled if the zero-crossing **Algorithm** is set to `Adaptive`.

Command-Line Information

Parameter: `ZCThreshold`
Value: `'auto'` | `any real number greater than or equal to zero`

Default: `'auto'`

Recommended Settings

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

See Also

Related Examples

- “Zero-Crossing Detection”
- “Zero-crossing control” on page 17-50
- “Consecutive zero-crossings violation” on page 12-14
- “Time tolerance” on page 17-52
- “Number of consecutive zero crossings” on page 17-55
- “Solver Pane” on page 17-2

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.

- “Inherit Sample Times for Model Referencing”
- “Inherited Sample Time for Referenced Models” (Simulink Coder)
- “Create and Reference Conditional 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 17-64).

`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**

Command-Line Information

Parameter: `SampleTimeConstraint`
Value: `'unconstrained' | 'STIndependent' | 'Specified'`
Default: `'unconstrained'`

Recommended Settings

Application	Setting
Debugging	No impact
Traceability	No impact
Efficiency	No impact
Safety precaution	`Specified` or `Ensure sample time independent`

See Also

Related Examples

- “Inherit Sample Times for Model Referencing”
- “Inherited Sample Time for Referenced Models” (Simulink Coder)
- “Create and Reference Conditional Referenced Models”
- “Function-Call Models”
- “Fixed-step size (fundamental sample time)” on page 17-64
- “Execute Multitasking Models” (Simulink Coder)
- “Solver Pane” on page 17-2

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

  $$h_{\max} = \frac{t_{stop} - t_{start}}{50}$$

- For Sine and Signal Generator source blocks, if the stop time is `Inf`, Simulink calculates the step size using this heuristic:

  $$h_{\max} = \min\left((0.2), \left(\frac{1}{3}\right)\left(\frac{1}{Freq_{\max}}\right) \right)$$

  Otherwise, the step size is:

  $$h_{\max} = \min\left(\frac{t_{stop} - t_{start}}{50}, \left(\frac{1}{3}\right)\left(\frac{1}{Freq_{\max}}\right) \right)$$

  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`.

Command-Line Information

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

- “Modeling Dynamic Systems”
- “Solver Pane” on page 17-2

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.

Command-Line Information

Parameter: `SampleTimeProperty`
Type: structure
Value: any valid matrix
Default: `[]`

Note If you specify `SampleTimeProperty` at the command line, 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 17-2

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.

Command-Line Information

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

- “Choose a Fixed-Step Solver”
- “Solver Pane” on page 17-2

Number Newton's iterations

Description

Specify the number of Newton's method iterations 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: 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 `ode14x (extrapolation)` from the **Solver** list.

Command-Line Information

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

- “Choose a Fixed-Step Solver”
- “Purely Discrete Systems”
- “Solver Pane” on page 17-2

Allow tasks to execute concurrently on target

Description

Enable concurrent tasking behavior for model.

Category: Solver

Settings

Default: On

 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`. 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**

Command-Line Information

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 22-2
- “Solver Pane” on page 17-2

Library Browser

Use the Library Browser

In this section...

Libraries Pane

Use the libraries pane to locate blocks by navigating block libraries. The pane displays a tree view of the libraries installed on your system. You can navigate the tree using your mouse or keyboard. Use the arrow keys to move through the library and expand or collapse libraries. When you select a library, its contents appear in the blocks pane.

Add Blocks Used Recently

The Library Browser provides quick access to the blocks you have used most recently. At the bottom of the libraries pane, click **Recently Used** to display the blocks in the blocks pane.

Note Only blocks added to your model from the Library Browser appear under **Recently Used**.

Refresh the Library Browser

To refresh the libraries displayed in the Library Browser, right-click in the libraries pane and select **Refresh Library Browser**. The Library Browser updates to display any libraries or blocks added to or deleted from the MATLAB path since the library browser was last opened or refreshed.

Refresh your library browser if you:

- Modify existing libraries or resave them in `.slx` file format.
- Update repository information for a library.
- Move or delete your library files.
- Add a library.

- Change your Library Browser customizations. See “Customize Library Browser Appearance”.

Blocks Pane

The blocks pane in the Library Browser displays the contents of the library selected in the libraries pane. You can use the blocks pane to navigate libraries, view block parameters or help, and create instances of library blocks in models.

Navigate Libraries

To open a library in the blocks pane, double-click the library. To return to the parent, from the block context menu, select **Go to parent**.

View Block Description, Parameters, and Help

To display the description and library path of a block, hover over the block.

To view the block parameters, double-click the block or, from the block context menu, select **Block parameters**.

To display help for a library block, from the block context menu, select **Help for the <name> block**.

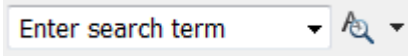
Add Blocks to Models from the Library

To add an instance of a library block to an open model using the Library Browser:

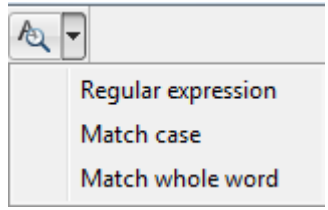
- Select the block in the blocks pane and drag it into the model window.
- From the block context menu, select **Add block to model <model_name>**, where **<model_name>** is the currently active model. If no model is open, use this command to create a model and add the block to it.

Search for Blocks in the Library Browser

1 Enter the search string in the search text box or select from the recent search list.



2 Use the search button menu to specify the search options you want to use, for example, match whole words.



3 Press **Enter** to start the search.

The blocks pane displays the blocks found, grouped by library. To see where the search string matched, hover over a block. To navigate to a block's library, from the block context menu, select **Select in library view**.

Library Browser Keyboard Shortcuts

Task	Shortcut
Open a model	**Ctrl+O**
Open Library Browser from a model	**Ctrl+Shift+L**
Move selection down in the Blocks or Libraries pane	**Down arrow**
Move selection up in the Blocks or Libraries pane	**Up arrow**
Expand a node in the Libraries pane	**Right arrow**
Collapse a node in the Libraries pane	**Left arrow**
Refresh Libraries pane	**F5**
Show parent library in Blocks pane	**Esc**
Select a block found with the search tool in the Blocks pane	**Ctrl+R**
Insert the selected block in a new model	**Ctrl+I**
Increase zoom in the Blocks pane	**Ctrl++**
Decrease zoom in the Blocks pane	**Ctrl+-**
Reset zoom to default in the Blocks pane	**Alt+1**
Find a block	**Ctrl+F**
Close	**Ctrl+W**

Signal Properties Dialog Box

Signal Properties Dialog Box Overview

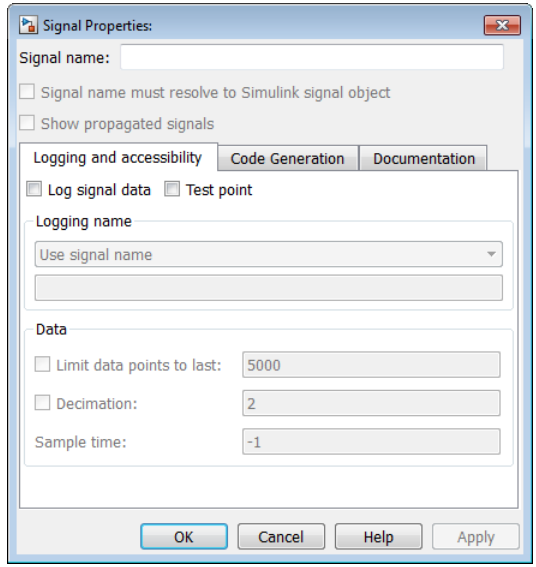
The **Signal Properties** dialog box lets you display and edit signal properties. To display the dialog box, either

- Select the line that represents the signal whose properties you want to set and then choose **Properties** from the signal's context menu.

  or

- Select a block that outputs or inputs the signal and from the block's context menu, select **Signals & Ports** and then either **Input Port Signal Properties** or **Output Port Signal Properties**, then select the port to which the signal is connected from the resulting menu.

The **Signal Properties** dialog box appears.



The dialog box includes the following controls.

- “Signal Properties Controls” on page 19-4
- “Logging and Accessibility Options” on page 19-6
- “Code Generation Options” on page 19-9

Signal Properties Controls

Signal name

Name of signal.

To name a signal programmatically, see “Name a Signal Programmatically”.

Signal name must resolve to Simulink signal object

Specifies that either the base MATLAB workspace, the model workspace, or the data dictionary must contain a `Simulink.Signal` object with the same name as this signal. Simulink software displays an error message if it cannot find such an object when you update or simulate the model containing this signal.

Note `Simulink.Signal` objects in the model workspace must have their storage class set to `Auto`. See “Model Workspaces” for more information.

When **Signal name must resolve to Simulink signal object** is enabled, a signal resolution icon appears by default to the left of any label on the signal. The icon looks like this:

See “Signal to Object Resolution Indicator” for more information.

This property appears only if you set the model configuration parameter **Signal resolution** to a value other than `None`.

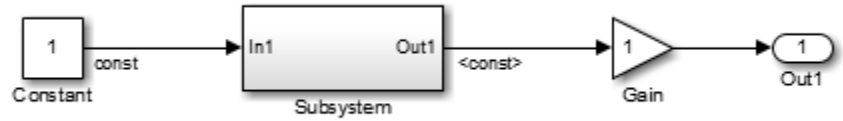
To set this option programmatically, use the port parameter `MustResolveToSignalObject`. See “Use Signal Objects”.

Show propagated signals

Note This option is available only for signals that originate from blocks that support signal label propagation. For a list of the blocks, see “Blocks That Support Signal Label Propagation”.

Enabling this property causes Simulink to create a propagated signal label.

For example, in the following model, the output signal from the Subsystem block is configured for signal label propagation. The propagated signal label (`<const>`) is based on the name of the upstream output signal of the Constant block (`const`).



For more information, see “Signal Label Propagation”.

To set this option programmatically, use the port parameter `SignalPropagation`. See “Display Propagated Signal Labels”.

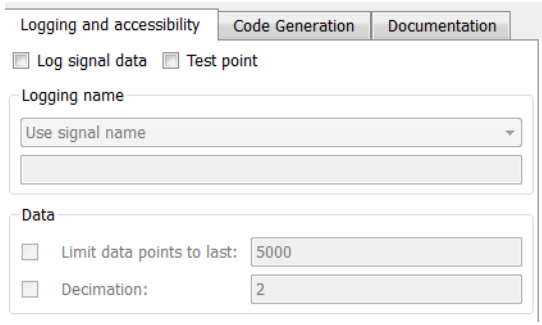
See Also

More About

- “Signal Properties Dialog Box Overview” on page 19-2

Logging and Accessibility Options

Select the **Logging and accessibility** tab on the **Signal Properties** dialog box to display controls that enable you to specify signal logging and accessibility options for this signal.



Log signal data

Select this option to cause Simulink software to save this signal's values to the MATLAB workspace during simulation. See “Export Signal Data Using Signal Logging” for details.

To set this option programmatically, use the port parameter `DataLogging`. See “Programmatic Interface”.

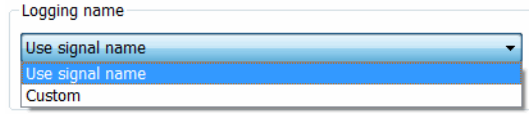
Test point

Select this option to designate this signal as a test point. See “Test Points” for details.

To set this property programmatically, use the port parameter `TestPoint`. See “Configure Signals as Test Points”.

Logging name

This pair of controls, consisting of a list box and an edit field, specifies the name associated with logged signal data.

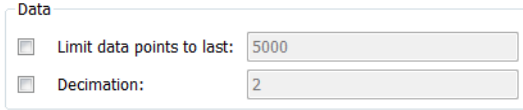


Simulink software uses the signal's signal name as its logging name by default. To specify a custom logging name, select `Custom` from the list box and enter the custom name in the adjacent edit field.

To set these properties programmatically, use the port parameters `DataLoggingNameMode` and `DataLoggingName`. See "Signal-Specific Logging Name Specified Programmatically".

Data

This group of controls enables you to limit the amount of data that Simulink software logs for this signal.



The options are as follows.

Limit data points to last

Discard all but the last `N` data points where `N` is the number entered in the adjacent edit field.

To set these properties programmatically, use the port parameters `DataLoggingLimitDataPoints` and `DataLoggingMaxPoints`. To set signal properties programmatically by using port parameters, see "Programmatically Specify Signal Properties".

When you specify a limit for the number of data points, signal data appears in the Simulation Data Inspector and on Dashboard Scope blocks when a simulation pauses or completes, with only the last `N` data points displayed. Other visualization Dashboard blocks are disabled, with a warning message displayed on the block.

Decimation

Log every `N`th data point where `N` is the number entered in the adjacent edit field. For example, suppose that your model uses a fixed-step solver with a step size of `0.1 s`. if you select this option and accept the default decimation value (`2`), Simulink software records data points for this signal at times `0.0`, `0.2`, `0.4`, etc.

To set these properties programmatically, use the port parameters `DataLoggingDecimateData` and `DataLoggingDecimation`. To set signal properties programmatically by using port parameters, see "Programmatically Specify Signal Properties".

The **Decimation** value you specify applies to the data sent to the Simulation Data Inspector and to Dashboard blocks in your model.

See Also

More About

- "Signal Properties Dialog Box Overview" on page 19-2

Code Generation Options

The following controls set properties that Simulink Coder uses to generate code from the model. If you are not going to generate code from the model, ignore them.

Signal object class

Choose a custom storage class package by selecting a signal object class that the target package defines. For example, to apply custom storage classes from the built-in package `mpt`, select `mpt.Signal`. Unless you use an ERT-based code generation target with Embedded Coder, custom storage classes do not affect the generated code.

If the class that you want does not appear in the drop-down list, select `Customize class lists`. For instructions, see "Apply Custom Storage Classes Directly to Signal Lines, Block States, and Outport Blocks" (Embedded Coder).

To apply storage classes interactively or programmatically, see "Control Signals and States in Code by Applying Storage Classes" (Simulink Coder). For information about custom storage classes, see "Control Data Representation by Applying Custom Storage Classes" (Embedded Coder).

Storage class

Select a storage class or custom storage class for the signal. To apply storage classes interactively or programmatically, see "Control Signals and States in Code by Applying Storage Classes" (Simulink Coder). For information about custom storage classes, see "Control Data Representation by Applying Custom Storage Classes" (Embedded Coder).

Type qualifier

Enter a storage type qualifier for this signal such as `const` or `volatile`.

This parameter is hidden unless you previously set its value. To enable this parameter, set **Storage class** to `ExportedGlobal`, `ImportedExtern`, `ImportedExternPointer`, or `SimulinkGlobal`.

Type qualifier will be removed in a future release. To apply storage type qualifiers to data, use custom storage classes and memory sections. Unless you use an ERT-based

code generation target with Embedded Coder, custom storage classes and memory sections do not affect the generated code.

See Also

More About

- “Signal Properties Dialog Box Overview” on page 19-2

Data Transfer Options for Concurrent Execution

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.

In this section...

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

More About

- “Signal Properties Dialog Box Overview” on page 19-2

Documentation Options

Description

In this field, enter a description of the signal.

The description that you specify in the Signal Properties dialog box does not appear in the generated code. To add a signal description as a comment in the generated code, you must use a Simulink signal object. For more information, see `Simulink.Signal`.

Document link

In the field that displays documentation for the signal, enter a MATLAB expression. To display the documentation, click **Document Link**. For example, entering the expression

```
web(['file:///' which('foo_signal.html')])
```

causes the MATLAB software default Web browser to display `foo_signal.html` when you click the field label.

To set this property programmatically, use the programmatic port parameter `DocumentLink`:

```
% Get a handle to the block output port
% that creates the target signal.
portHandles = get_param('myModel/myBlock','portHandles');
outportHandle = portHandles.Outport;

% Set the document link.
set_param(outportHandle,'DocumentLink',...
'web([''file:///'' which(''foo_signal.html'')])')
```

See Also

More About

- “Signal Properties Dialog Box Overview” on page 19-2

Simulink Preferences Window

Set Simulink Preferences

Simulink Preferences Window Overview

You can use Simulink Preferences to specify Simulink editing environment options and default behaviors. Your settings affect the behavior of all Simulink models, including those currently open, and all subsequent models. Your preference settings are preserved for the next time you use the software. The Simulink Preferences window has these panes:

- "Simulink Preferences General Pane" on page 20-3

  Set preferences for generated file folders; background colors for print or export; Model block, callback, and sample time legend display.
- "Simulink Preferences Editor Pane" on page 20-22

  Configure the Simulink Editor.
- "Simulink Preferences Model File Pane" on page 20-11

  Set preferences for file change, autosave, version notifications, and other behaviors relating to model files.

To open the Simulink Preferences dialog box:

- From a Simulink Editor menu, select **File > Simulink Preferences**.
- In the MATLAB Command Window, enter

```
slprivate('showprefs')
```

See Also

- "Create a Template from a Model"
- "Simulink Preferences General Pane" on page 20-3
- "Simulink Preferences Editor Pane" on page 20-22
- "Simulink Preferences Model File Pane" on page 20-11

Simulink Preferences General Pane

Simulink General Preferences Overview

Set preferences to specify the location for generated file folders and background colors for print or export. You also set these preferences to specify information about Model blocks, callbacks, and sample time legend display.

See Also

- "Simulink Preferences Model File Pane" on page 20-11
- "Simulink Preferences Editor Pane" on page 20-22

Folders for Generated Files

Use these preferences to control the location of model build artifacts. By default, build artifacts are placed in the current working folder (`pwd`) at the start of a diagram update or code generation. For more information, see:

- "Simulation cache folder" on page 20-3
- "Code generation folder" on page 20-4
- "Simulation cache folder" on page 20-3

Simulation cache folder

Specify the root folder in which to put model build artifacts used for simulation.

Settings

Default:`''`

Specify a valid folder path. If you do not specify a path, build artifacts are placed in the current working folder (`pwd`) when you update a diagram.

Tips

- You can specify an absolute or relative path to the folder. For example:

  - `C:\Work\mymodelsimcache` and `/mywork/mymodelsimcache` specify absolute paths.
  - `mymodelsimcache` is a path relative to the current working folder (`pwd`). The software converts a relative path to a fully qualified path when you set the preference. For example, if `pwd` is `'/mywork'`, the result is `/mywork/mymodelsimcache`.
  - `../test/mymodelsimcache` is a path relative to `pwd`. If `pwd` is `'/mywork'`, the result is `/test/mymodelsimcache`.
- The simulation cache folder is where Simulink cache files are created. For details about these cache files, see "Reuse Simulation Builds for Faster Simulations".

Command-Line Information

Parameter: `CacheFolder`
Type: character vector
Value: valid folder path
Default: `''`

See Also

- "Simulation Target Output File Control"
- "Reuse Simulation Builds for Faster Simulations"

Code generation folder

Specify the root folder in which to put Simulink Coder code generation files.

Settings

Default:`''`

Specify a valid folder path. If you do not specify a path, build artifacts are placed in the current working folder (`pwd`) when you start to generate code.

Tip

You can specify an absolute or relative path to the folder. For example:

- `C:\Work\mymodelgencode` and `/mywork/mymodelgencode` specify absolute paths.

- `mymodelgencode` is a path relative to the current working folder (`pwd`). The software converts a relative path to a fully qualified path at the time the preference is set. For example, if `pwd` is `'/mywork'`, the result is `/mywork/mymodelgencode`.
- `../test/mymodelgencode` is a path relative to `pwd`. If `pwd` is `'/mywork'`, the result is `/test/mymodelgencode`.

Command-Line Information

Parameter: `CodeGenFolder`
Type: character vector
Value: valid folder path
Default: `''`

See Also

"Build Folder and Code Generation Folders" (Simulink Coder)

Code generation folder structure

Specify structure of generated code folder.

Settings

Default: `Model specific`

`Model specific`

Place generated code in subfolders within a model-specific folder.

`Target environment subfolder`

If you configured models for different target environments, place generated code for each model in a separate subfolder. The name of the subfolder corresponds to the target environment.

Command-Line Information

Parameter: `CodeGenFolderStructure`
Value: `'ModelSpecific' | 'TargetEnvironmentSubfolder'`
Default: `'ModelSpecific'`

See Also

"Manage Build Process Folders" (Simulink Coder)

Background Color

Use these preferences to control the background color for printing, exporting to another format, and models copied to the clipboard for export to another application.

- "Print" on page 20-6
- "Export" on page 20-6
- "Clipboard" on page 20-7

Print

Use a white canvas (background) or the canvas color of the model when printing a model.

Settings

Default: `White`

`White`

Use a white canvas.

`Match Canvas Color`

Match the canvas color of the model.

Command-Line Information

Parameter: `PrintBackgroundColorMode`
Value: `'White' | 'MatchCanvas'`
Default: `'White'`

See Also

"Share Models"

Export

Match the canvas (background) color of the model, use a white canvas, or use a transparent canvas for model files that you export to another file format, such as `.png` or `.jpeg`.

Settings

Default: `Match Canvas Color`

`Match Canvas Color`

Match the canvas color of the model.

`White`

Use a white canvas.

`Transparent`

Use a transparent canvas, so that whatever is behind the canvas is visible.

Command-Line Information

Parameter: `ExportBackgroundColorMode`
Value: `'White' | 'MatchCanvas' | 'Transparent'`
Default: `'MatchCanvas'`

See Also

"Copy Model Views to Third-Party Applications"

Clipboard

Match the canvas (background) color of the model, use a white canvas, or use a transparent canvas for model files that you export to another application.

Settings

Default: `Match Canvas Color`

`Match Canvas Color`

Match the canvas color of the model.

`White`

Use a white canvas.

`Transparent`

Use a transparent canvas, so that whatever is behind the canvas image shows through.

Command-Line Information

Parameter: `ClipboardBackgroundColorMode`
Value: `'White' | 'MatchCanvas' | 'Transparent'`
Default: `MatchCanvas`

See Also

"Print Models to Image File Formats"

Warn when opening Model blocks with Normal Mode Visibility set to off

Show a warning when you open a model from Model blocks that have normal mode visibility set to off.

All instances of a normal mode referenced model are part of the simulation. However, Simulink displays only one instance in a model window. That instance is determined by the normal mode visibility setting. Normal mode visibility includes the display of Scope blocks and data port values. When you open a model from a Model block that has normal mode visibility set to off, the referenced model shows data from the instance of the model has normal mode visibility set to on.

Settings

Default: On

 On

After simulation, Simulink displays a warning if you try to open a referenced model from a Model block that has normal mode visibility set to off. Simulink does not open the that instance referenced by the Model block, but instead opens the instance that has normal mode visibility set to on. The instance that has normal mode visibility set to on has different input data sources than the instance referenced by the Model block that you opened.

 Off

No warning is displayed if, after simulation, you try to open a referenced model from a Model block that has normal mode visibility set to off.

Tip

The warning that appears includes an option to suppress the display of the warning in the future. Enabling that option sets this preference to off. Use this preference to resume the display of the warning.

See Also

"Simulate Models with Multiple Referenced Model Instances"

Show callback tracing

Specify whether to display the model callbacks that Simulink invokes when simulating a model.

Settings

Default: Off

 On

Display the model callbacks in the MATLAB Command Window as they are invoked.

When you open or simulate a model, you can determine the callbacks invoked and their order.

 Off

Do not display model callbacks.

Command-Line Information

Parameter: `CallbackTracing`
Value: `'on' | 'off'`
Default: `'off'`

Open the sample time legend when the sample time display is changed

Specify whether to display the sample time legend whenever the sample time display changes.

Settings

Default: On

 On

Display the sample time legend whenever you change the sample time display by selecting Colors, Annotations, or All from the **Sample Time Display** menu. The model is updated and the legend opens.

 Off

Do not display the sample time legend whenever the sample time display changes.

Command-Line Information

Parameter: `OpenLegendWhenChangingSampleTimeDisplay`
Value: `'on' | 'off'`
Default: `'on'`

See Also

"View Sample Time Information"

Simulink Preferences Model File Pane

Simulink Model File Preferences Overview

Set preferences for file change, autosave, version notifications, and other behaviors relating to model files

These options affect the behavior of all Simulink models.

See Also

- “Simulink Preferences General Pane” on page 20-3
- “Simulink Preferences Editor Pane” on page 20-22

File format for new models and libraries

Settings

Default:`SLX`

Specify the default file format for new models and libraries.

`MDL`

Save new models and libraries in MDL format.

`SLX`

Save new models and libraries in SLX format.

Command-Line Information

Parameter: `ModelFileFormat`
Value: `'mdl' | 'slx'`
Default: `slx`

Tip

You can choose model file format when using **Save As**.

See Also

“Save Models in the SLX File Format”

Save a thumbnail image inside SLX files

Specify whether to save a small screenshot of the model to display in the Current Folder browser preview pane.

Settings

Default: On

 On

When saving the model, include a small screen shot of the model inside the SLX file. You can view the screen shot for a selected model in the Current Folder browser preview pane.

 Off

Do not save a screenshot of the model.

Tip

If your model is very large and you want to reduce the time taken to save the model, then you can turn this preference off to avoid saving thumbnail model images.

Command-Line Information

Parameter: `SaveSLXThumbnail`
Value: `'on' | 'off'`
Default: `on`

Change Notification

Use these preferences to specify notifications if the model has changed on disk when you update, simulate, edit, or save the model. When updating or simulating, you can choose whether to warn, error, reload if unmodified, or show a dialog box that lets you choose to reload or ignore. For more information, see “Model File Change Notification”.

You can set these options under **Change Notification**:

- “Updating or simulating the model” on page 20-13
- “Action” on page 20-14
- “First editing the model” on page 20-15

- "Saving the model" on page 20-15

Updating or simulating the model

Specify whether to notify if the model has changed on disk when updating or simulating the model.

Settings

Default: On

 On

Notify if the model has changed on disk when updating or simulating the model. Select the action to take in the **Action** list.

 Off

Do not notify if the model has changed on disk when updating or simulating the model.

Tip

To programmatically check whether the model has changed on disk since it was loaded, use the function `slIsFileChangedOnDisk`.

Dependency

This parameter makes **Action** available.

Command-Line Information

Parameter: `MDLFileChangedOnDiskChecks`
Type: struct, field name: CheckWhenUpdating
Value: `true` | `false` | `1` | `0`
Default: `true`

See Also

"Model File Change Notification"

Action

Select the action to take if the file has changed on disk since it was loaded.

Settings

Default: `Warning`

`Warning`

Displays a warning in MATLAB command window

`Error`

Displays an error. If simulating programmatically, the error appears in the MATLAB command window. If simulating interactively, the error appears in a Simulation Diagnostics window.

`Reload model (if unmodified)`

Reloads if the model is unmodified. If the model is modified, the prompt dialog box appears.

`Show prompt dialog`

Shows prompt dialog box in which you can choose to close and reload or ignore the changes.

Tip

To programmatically check whether the model has changed on disk since it was loaded, use the function `slIsFileChangedOnDisk`.

Dependency

This parameter is enabled by the **Updating or simulating the model** parameter.

Command-Line Information

Parameter: `MdlFileChangedOnDiskHandling`
Value: `'Warning' | 'Error' | 'Reload model (if unmodified)' | 'Show prompt dialog'`
Default: `'Warning'`

See Also

“Model File Change Notification”

First editing the model

Specify whether to notify if the file has changed on disk when editing the model.

Settings

Default: On

 On

Displays a warning if the file has changed on disk when you modify the block diagram. Any interactive operation that modifies the block diagram (e.g., adding a block) causes a warning dialog box to appear. Any programmatic operation that causes the block diagram to be modified (e.g., a call to `set_param`) causes a warning in the MATLAB Command Window

 Off

Do not check for changes on disk when first editing the model.

Tip

To programmatically check whether the model has changed on disk since it was loaded, use the function `slIsFileChangedOnDisk`.

Command-Line Information

Parameter: `MDLFileChangedOnDiskChecks`
Type: struct, field name: CheckWhenEditing
Value: `true` | `false` | `1` | `0`
Default: `true`

See Also

"Model File Change Notification"

Saving the model

Specify whether to notify if the file has changed on disk when saving the model.

Settings

Default: On

 On

Notify if the file has changed on disk when you save the model.

- Saving the model in the Simulink Editor causes a dialog box to appear. In the dialog box, you can choose to overwrite or save with a new name.
- The `save_system` function displays an error, unless you use the `OverwriteIfChangedOnDisk` option.

 Off

Do not check for changes on disk when saving the model.

Tip

To programmatically check whether the model has changed on disk since it was loaded, use the function `slIsFileChangedOnDisk`.

Command-Line Information

Parameter: `MDLFileChangedOnDiskChecks`
Type: struct, field name: CheckWhenSaving
Value: `true` | `false` | `1` | `0`
Default: `true`

See Also

"Model File Change Notification"

Autosave Options

Use the autosave preferences to specify whether to automatically save a backup copy of the model before updating or simulating, or when overwriting with a newer version of Simulink.

For more information, see these options:

- "Save before updating or simulating the model" on page 20-17
- "Save backup when overwriting a file created in an older version of Simulink" on page 20-18

Save before updating or simulating the model

Specify whether to automatically save a backup copy of the model before updating or simulating.

Settings

Default: On

☑ On

If the model has unsaved changes, automatically save a backup copy of the model before updating or simulating. This autosave copy can be useful for crash recovery.

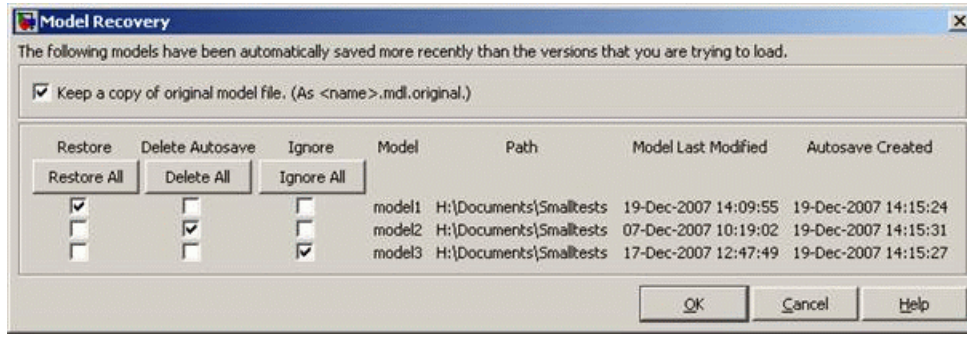
The copy is saved in the same directory as the model, with the name *MyModel*`.slx.autosave` or *MyModel*`.mdl.autosave`.

☐ Off

Do not automatically save a copy before updating or simulating.

Tips

- If you open or load a model that has a more recent autosave copy available, then after the model loads, a dialog box prompts to restore, ignore, or discard the autosave copy. If multiple models area involved, then the Model Recovery dialog box appears.



For each model in the list, you can select a check box to specify whether to **Restore**, **Delete Autosave**, or **Ignore**. Or you can click the **Restore All**, **Delete All** or **Ignore All** button to select that option for all listed models.

Option	Result
Restore	Overwrite the original model file with the autosave copy, and delete the autosave copy. Simulink will close the model and reload from the restored file. If you select the check box to **Keep a copy of original model file**, you can save copies of the original model files named *MyModel*`.slx.original` or *MyModel*`.mdl.original`.
Delete Autosave	Delete the autosave copy.
Ignore	Leave the model and the autosave copy untouched. This setting is the default. The next time you open the model, the Model Recovery dialog will reappear and you can choose to restore or delete autosave files.

- Closing a modified model deletes any autosave copy.
- Autosave does not occur for models that are part of the MATLAB installation, so you will not create autosave copies of those models.
- Autosave does not occur if the autosave file or location is read only.
- Autosave does not occur in Parallel Computing Toolbox™ workers.

Caution If a segmentation violation occurred, then the last autosave file for the model reflects the state of the autosave data prior to the segmentation violation. Because Simulink models might be corrupted by a segmentation violation, Simulink does not autosave a model after a segmentation violation occurs.

Command-Line Information

Parameter: `AutoSaveOptions`
Type: struct, field name: SaveOnModelUpdate
Value: `true` | `false` | `1` | `0`
Default: `true`

Save backup when overwriting a file created in an older version of Simulink

Specify whether to automatically save a backup copy of the model when overwriting with a newer version of Simulink.

Settings

Default: On

 On

If saving the model with a newer version of Simulink, automatically save a backup copy of the model. This backup copy can be useful for recovering the original file in case of accidental overwriting with a newer version.

The backup copy is saved in the same directory as the model, with the name `*MyModel*.slx.*Version*` or `*MyModel*.mdl.*Version*`, where `*Version*` is the last version that saved the model, e.g., `R2010a`.

 Off

Do not automatically save a backup copy when overwriting a model with a newer version of Simulink.

Tip

To recover the original model, rename the backup copy to `*MyModel*.mdl` or `*MyModel*.slx` by deleting the `*Version*` suffix.

Command-Line Information

Parameter: `AutoSaveOptions`
Type: struct, field name: SaveBackupOnVersionUpgrade
Value: `true | false | 1 | 0`
Default: `true`

Notify when loading an old model

Specify whether to notify when loading a model last saved in a older version of Simulink software.

Settings

Default: Off

 On

Print a message in the command window when loading a model last saved in an old version of Simulink software.

Off

No notification when loading old models.

Tips

- Run the Upgrade Advisor to convert the block diagram to the format of the current version of Simulink software.
- For advice on upgrading a model to the current version of Simulink, see “Model Upgrades”.

Command-Line Information

Parameter: `NotifyIfLoadOldModel`
Value: `'on' | 'off'`
Default: `off`

Do not load models created with a newer version of Simulink

Specify whether to load a model last saved in a newer version of Simulink software.

Settings

Default: On

On

Do not load any model last saved in a newer version of Simulink software, and print an error message in the command window.

Off

Load models last saved in a newer version of Simulink software, and print a warning message in the command window.

Tip

If possible, use the **Save As** command to convert the block diagram to the format of the desired version of Simulink software. The **Save As** command allows you to save a model created with the latest version of the Simulink software in formats used by earlier versions. See “Export a Model to a Previous Simulink Version”.

Command-Line Information

Parameter: `ErrorIfLoadNewModel`
Value: `'on' | 'off'`
Default: `on`

Do not load models that are shadowed on the MATLAB path

Specify whether to load a model that is shadowed by another file of the same name higher on the MATLAB path.

Settings

Default: Off

 On

Do not load any model that is shadowed by another file of the same name higher on the MATLAB path, and print an error message in the command window. This preference applies when you try to open or load a model or library by either:

- Selecting a file in the current folder browser
- Calling `open_system` or `load_system` with a path to a file in a different folder to the current folder

 Off

Load shadowed models, and print a warning message in the command window.

Command-Line Information

Parameter: `ErrorIfLoadShadowedModel`
Value: `'on' | 'off'`
Default: `off`

See Also

“Shadowed Files”.

Simulink Preferences Editor Pane

Simulink Editor Preferences Overview

Configure the Simulink Editor. These options affect the behavior of all Simulink models. The options relate to the how models appear in terms of the visual theme, the scroll wheel behavior, and the toolbar configuration.

See Also

- “Simulink Preferences General Pane” on page 20-3
- “Simulink Preferences Model File Pane” on page 20-11

Use classic diagram theme

Cause Simulink diagrams to appear in the Simulink Editor using the visual theme that was used in the Simulink Editor before R2012b.

If you check **Use classic diagram theme**, Simulink does not display content preview. For details, see “Preview Content of Hierarchical Items”.

Line crossing style

Change the default display for signal lines that cross. By default, straight signal lines that cross each other but are not connected display a slight gap before and after the vertical line where it intersects the horizontal line. This display style is **Tunnel**.

The **Line Hop** format shows a bend where the vertical line intersects the horizontal line. Simulink adjusts the side the bend appears on to avoid overlapping with a block icon. If having the bend on either side overlaps with a block, Simulink uses a solid line.

The **None** format uses solid lines. This format can provide a slight performance improvement for updating very large models. If you enable the **Simulink Preferences** > **Editor Defaults** > **Use classic diagram theme** preference, Simulink uses a solid line.

Scroll wheel controls zooming

Use the scroll wheel on the mouse to zoom in and out without the **Ctrl** key modifier. On Macintosh platforms with an Apple Magic Trackpad, if you enable **Scroll wheel controls zooming**, a panning gesture causes zooming.

Enable smart editing features

Use smart editing cues to perform common model editing tasks quickly. These features appear based on context. Select this check box to enable:

- Quick insert — Add a block to a model by typing a block name.
- Tear-off block addition — Add a complementary block from a block tear-off cue. For example, when you add a GoTo block, you can use a tear-off to add a corresponding From block.
- Multiselection actions — Perform actions from the prompt that appears when you select multiple blocks.
- Single-selection actions — Perform actions from the prompt that appears when you select a block or a signal.

By default, these features are enabled.

Edit key parameter when adding new blocks

When you add a block to a model, a prompt appears so that you can enter a key parameter. This feature is enabled by default. Clear this check box if you do not want the prompt to appear.

Toolbar Configuration

Use these preferences to control the buttons that appear on the toolbar. For more information, see these options:

- “Simulation cache folder” on page 20-3
- “Code generation folder” on page 20-4

File Toolbar

Specify whether to display the **New** and **Save** buttons, the **New**, **Open**, and **Save** buttons, or no file buttons in the toolbar.

Print

Specify whether to show the **Print** button in the toolbar.

Cut/Copy/Paste

Specify whether to show the **Cut**, **Copy**, and **Paste** buttons in the toolbar.

Undo/Redo

Specify whether to show the **Undo** and **Redo** buttons in the toolbar.

Browse Back/Forward/Up

Specify whether to show the browsing buttons in the toolbar.

Library/Model Configuration/Model Explorer

Specify whether to show the **Library**, **Model Configuration**, and **Model Explorer** buttons in the toolbar.

Refresh Blocks

Specify whether to show the **Refresh Blocks** in the toolbar.

Update Diagram

Specify whether to show the **Update Diagram** button in the toolbar.

Simulation

Specify whether to show the simulation controls in the toolbar.

Fast Restart

Specify whether to show the **Fast Restart** button with the simulation controls in the toolbar.

Debug Model

Specify whether to show the **Debug Model** button in the toolbar.

Model Advisor

Specify whether to show the **Model Advisor** button in the toolbar.

Build

Specify whether to show the **Build** button in the toolbar.

Find

Specify whether to show the **Find** button in the toolbar.

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. Close **OK**.

Simulink Mask Editor

Mask Editor Overview

In this section...

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,

- Click **Diagram > Mask > Create Mask**.
- Right-click the block and select **Mask > Create Mask**.

To edit mask,

- Click **Diagram > Mask > 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:

- “Icon & Ports Pane” on page 21-3: To create block mask icons.
- “Parameters & Dialog Pane” on page 21-13: To design mask dialog boxes.
- “Initialization Pane” on page 21-26: To initialize a masked block using MATLAB code.

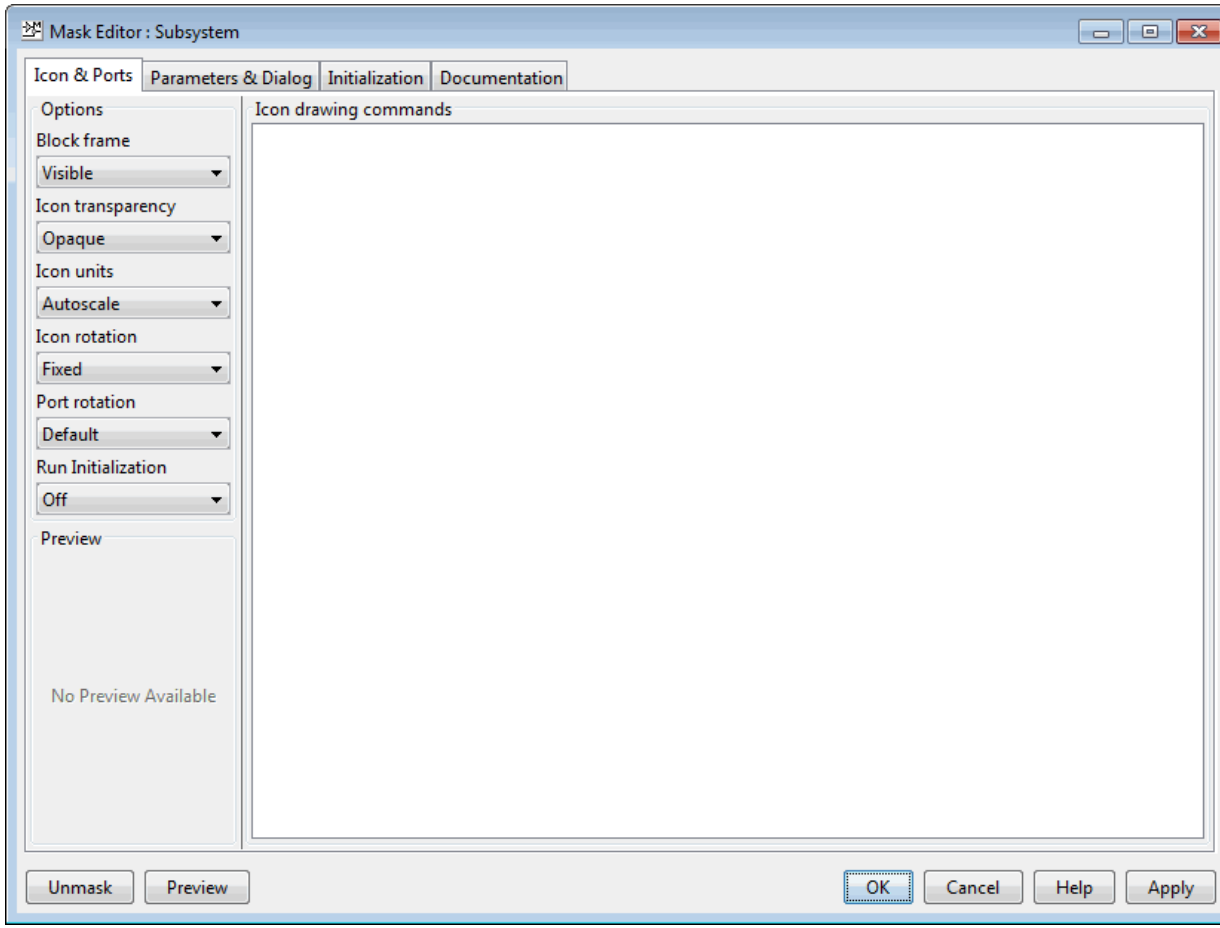
- "Documentation Pane" on page 21-29: To add description and help about the block mask.

Note For information on creating and editing a block mask from command line, see "Control Masks Programmatically".

Icon & Ports Pane

- "Options" on page 21-5
- "Preview" on page 21-10
- "Icon drawing commands" on page 21-10

The **Icon & Ports** pane helps you to create a block icon that contains descriptive text, state equations, image, and graphics.



The **Icon & Ports** pane is divided into these sections:

- "Options" on page 21-5: Provides a list of different controls that can be applied on the mask icon.
- "Preview" on page 21-10: Displays the preview of the block mask icon.
- "Icon drawing commands" on page 21-10: Enables you to draw mask icon by using MATLAB code.

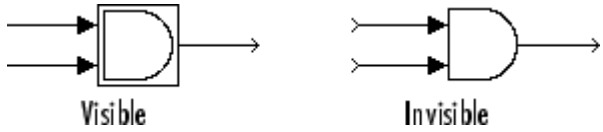
Note You can create static and dynamic block mask icon. For more information, see "Draw Mask Icon" and slexMaskDisplayAndInitializationExample.

Options

Options available in the left pane are a list of controls that allow you to specify attributes on the mask icon. These options are,

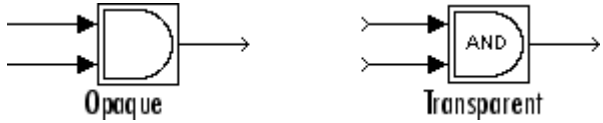
Block frame

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.

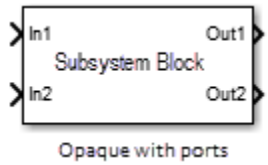


Icon transparency

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.



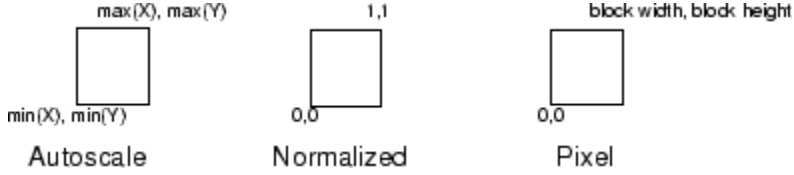
For a subsystem block, if you set the icon transparency to `Opaque with ports` the port labels are visible.



Note 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 you 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`

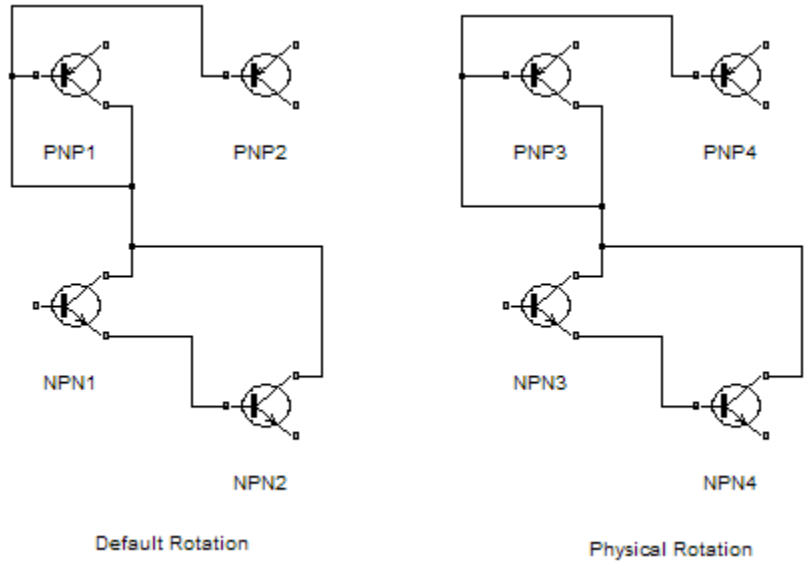
  Ports rotate with the block without being reordered after a clockwise rotation.

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.

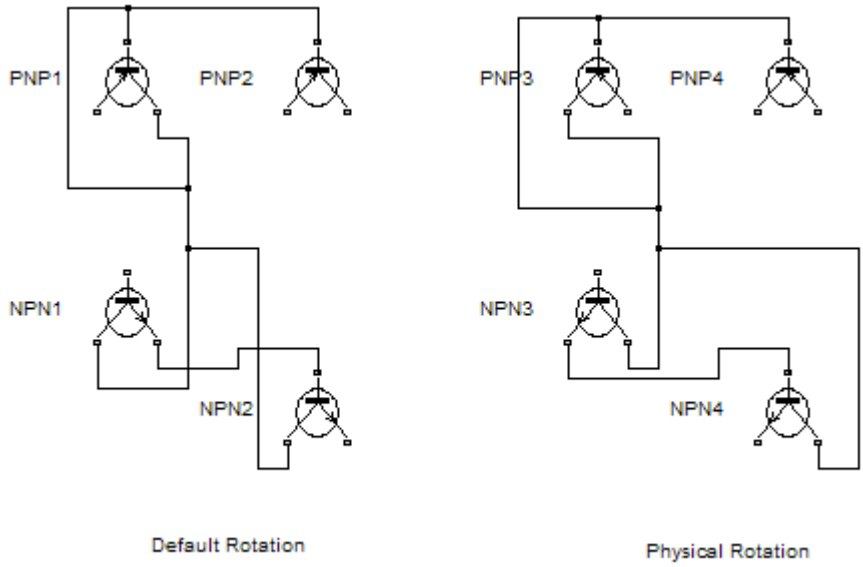
Similarly, the physical rotation option is appropriate for electronic, mechanical, hydraulic, and other modeling applications where blocks represent physical components

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 a Block” 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**: Always executes the mask initialization commands. 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

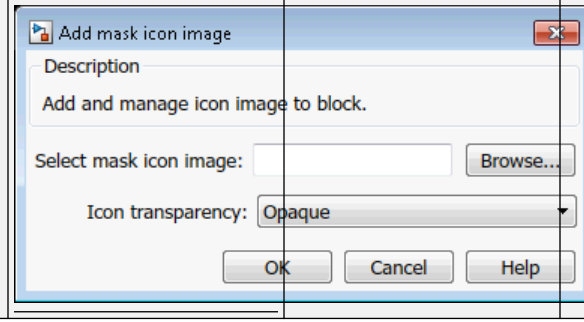
This section displays the preview of block mask icon. Block mask preview is available only if the mask contains an icon drawing.

When you add an icon drawing command and click **Apply**, the preview image refreshes and is displayed in the **Preview** section of **Icon & Ports** pane.

Icon drawing commands

The Icon drawing commands text box available in the center pane enables you to add code to draw the block icon. You can use the list of commands mentioned in the Mask icon drawing commands tables 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	`color('red'); port_label('output',1,'Text')`	
`disp`	Display text on the masked icon.	`disp('Gain')`	
`dpoly`	Display transfer function on masked icon	`dpoly([0 0 1], [1 2 1], 'z')`	
`droots`	Display transfer function on masked icon	`droots([-1], [-2 -3], 4)`	
`fprintf`	Display variable text centered on masked icon	`fprintf('Sum = %d', 7)`	
`image`	Display RGB image on masked icon **Note** To add mask icon from the user interface, click **Mask > Add Mask Icon** in the context menu. 	`image('b747.jpg')`	

Drawing Command	Description	Syntax Example	Preview
`patch`	Draw color patch of specified shape on masked icon	`patch([0 10 20 30 30 0], [10 30 20 25 10 10],[1 0 0])`	
`plot`	Draw graph connecting series of points on masked icon	`plot([10 20 30 40], [10 20 10 15])`	
`port_label`	Draw port label on masked icon	`port_label('output', 1, 'xy')`	
`text`	Display text at specific location on masked icon **Note** You must select `Pixels` in the **Icon units** box.	`text(5,10, 'Gain')`	

Note Simulink does not support mask drawing commands within anonymous functions.

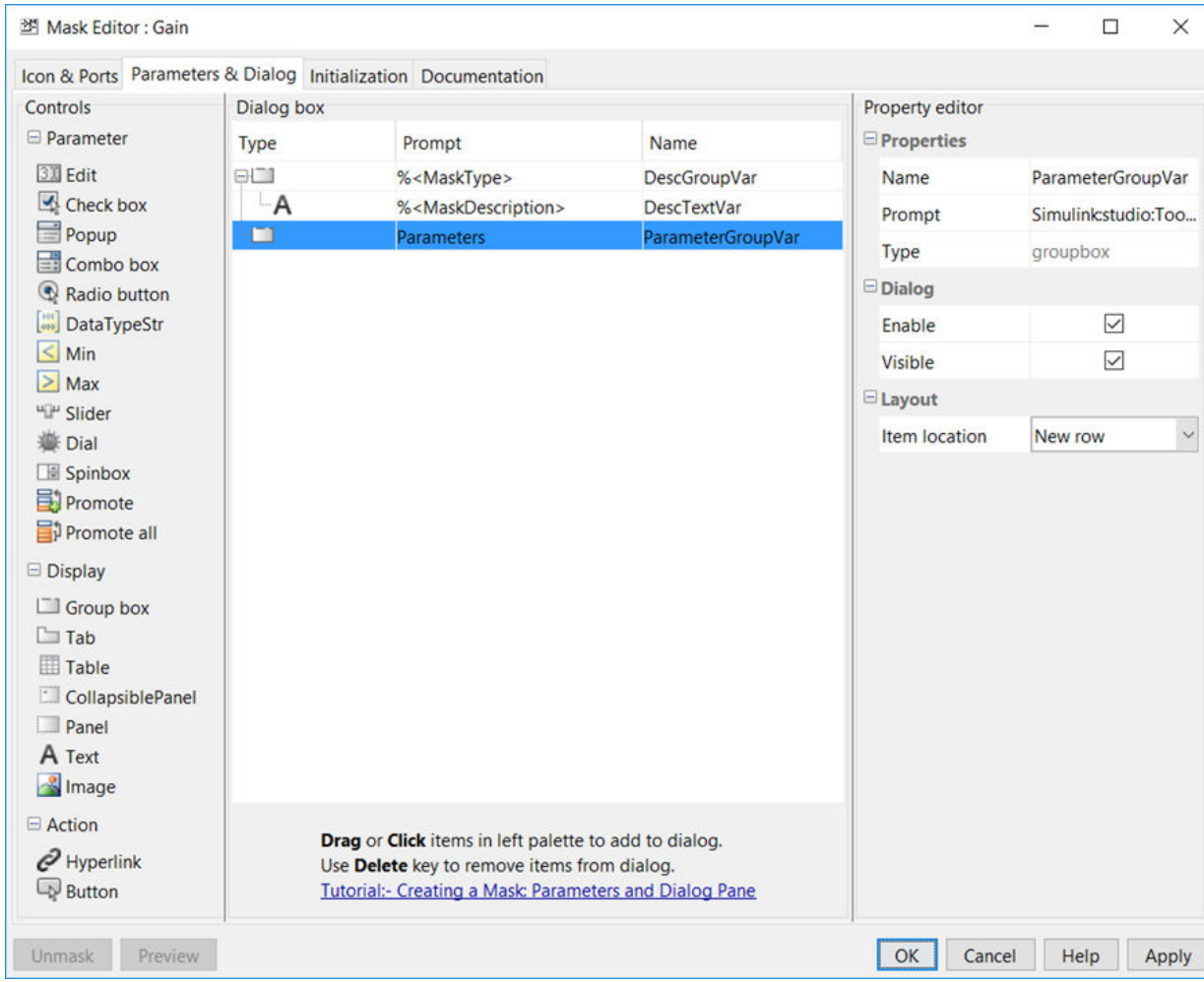
The drawing commands execute in the same sequence as they are added in the **Icon drawing commands** text box. Drawing commands have access to all variables in the mask workspace. If any drawing command cannot successfully execute, the block icon displays question masks  .

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.

Parameters & Dialog Pane

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 21-16	Controls 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.
		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 21-21	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 21-22	The **Property editor** allows you to view and set the properties for the **Parameters**, **Display**, 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.

Section	Section Description	Sub-Section	Sub-Section Description
		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 out the different controls and their description.

Controls Table

Controls		Description
Parameters		
	Edit	Allows you to enter a parameter value by typing it into the field.
	Check box	Accepts a Boolean value.
	Popup	Allows you to select a parameter value from a list of possible values.
	Combo box	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. The values provided for the **Combo box** parameter are not evaluated. For more information, see the Combo box example in slexMaskParameterOptions Example.
	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	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**. You can also control the slider range dynamically. For more information, see slexMaskParameterOptions Example.
	Dial	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**. You can also control the dial range dynamically. For more information, see slexMaskParameterOptions Example.

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.
	DataTypeStr	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 21-38.
	Min	Specifies a minimum value for the **DataTypeStr** parameter.
	Max	Specifies a maximum value for the **DataTypeStr** parameter.
	Promote parameter	Allows you to selectively promote block parameters from underlying blocks to the mask. Click the **Type options** field 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.

Controls		Description
	Promote all	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.
Display		
	Panel	Container to group of dialog controls. You use a **Panel** for logical grouping of dialog controls.
	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”.

Controls		Description
	CollapsiblePanel	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.
	Text	Text displayed in the mask dialog box.
	Image	Image displayed in the mask dialog box.
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 slexMaskParameterOptions Example.

Dialog box

You can build a hierarchy of dialog controls by dragging them from a **Controls** section to the **Dialog box**. You can also click the palettes on the **Controls** section to add the required control to the **Dialog box**. You can add a maximum of 32 levels of hierarchy in the **Dialog box**.

The **Dialog box** displays three fields: **Type**, **Prompt**, and **Name**.

- The **Type** field shows the type of the dialog control and cannot be edited. It also displays a sequence number for parameter dialog controls.

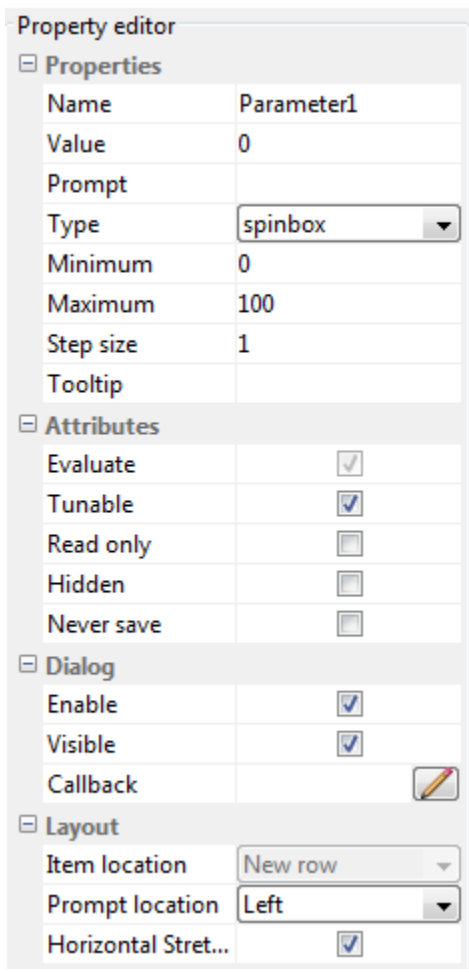
- The **Prompt** field shows the prompt text for the dialog control.
- The **Name** field is auto-populated and uniquely identifies the dialog controls. You can choose to add a different value (valid MATLAB name) in the **Name** field.

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 21-34.

Property editor

The **Property editor** allows you to view and set the properties for **Parameter**, **Display**, and **Action** dialog controls. The **Property editor** for **Parameter** is shown:



You can set the following properties for **Parameter**, **Action**, and **Display** dialog controls. For more information, see the Property editor table.

Property editor

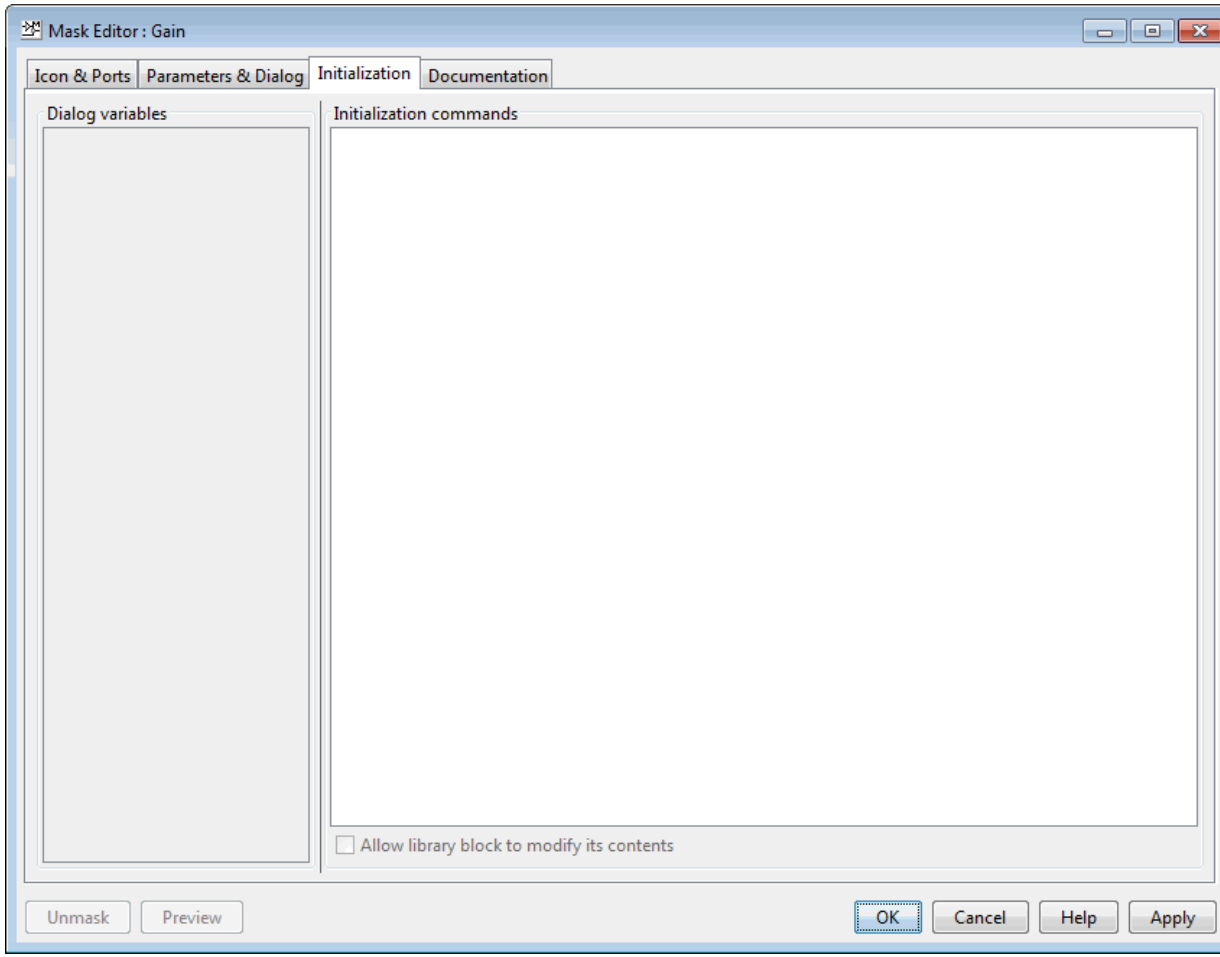
Property	Description
Properties	
Name	Uniquely identifies the dialog control in the mask dialog box. The **Name** property must be set for all dialog controls.
Value	Value of the **Parameter** dialog control. The **Value** property applies only to the **Parameter** dialog controls.
Prompt	Label text 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 change the **Type** field only for the **Parameter** dialog controls.
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	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.
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.

Property	Description
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**.
Attributes	
Evaluate	Simulink uses the value of a mask parameter as you type it in the mask dialog box, or it can evaluate what you specify and use as the result of the evaluation. Select the **Evaluate** option for a parameter to specify parameter evaluation (the default). To suppress evaluation, clear the option.
Tunable	By default, you can change a mask parameter value during simulation. To prohibit 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 Editor. We can specify the type of parameter where tunable is disabled. For information about parameter tuning and the blocks that support it, see “Tune and Experiment with Block Parameter Values”.
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.
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.

Property	Description
Callback	MATLAB code that you want Simulink to execute when a user applies a change to the selected control. Simulink uses the base workspace to execute the callback code.
Layout	
Item location	Allows you to set the location for the dialog control to appear in the current row or a new row.
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.

Initialization Pane

The **Initialization** pane allows you to add MATLAB commands that initialize the masked block.



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 **Initialization** 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.

Allow library block to modify its contents

This check box is enabled only if the masked subsystem resides in a library. Checking this option allows the block's initialization code to modify the contents of the masked subsystem by adding or deleting blocks and setting the parameters of those blocks. Otherwise, an error is generated when a masked library block tries to modify its contents in any way.

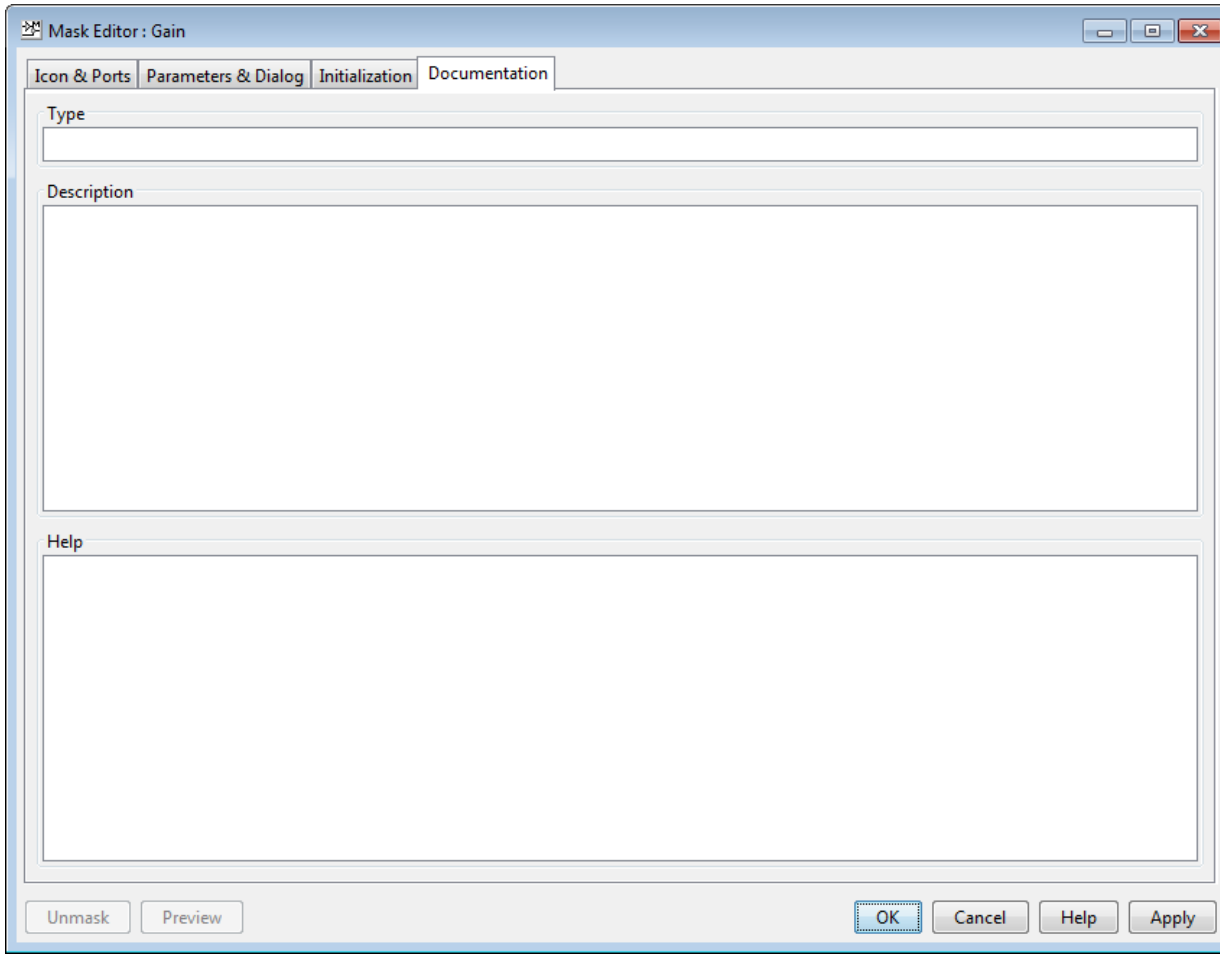
Rules for Initialization commands

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.

Documentation Pane

The **Documentation** pane enables you to define or modify the type, description, and help text for a masked block.



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 an URL

If the first line of the **Mask help** field is an URL, Simulink passes the URL to your default web browser. The URL can begin with `http:`, `www:`, `file:`, `ftp:`, or `mailto:`. Examples:

```
http://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 **Mask 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 **Mask help** field is an `eval` command, Simulink passes the command to MATLAB, which performs the specified evaluation. Example:

```
eval('!Word 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 **Mask help** field is not an URL, or a `web` or `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.

Additional Options

Following buttons appear on the **Mask Editor**:

- The **Preview** button applies the changes you made, and opens the mask dialog box.
- The **OK** button applies the mask settings and closes the **Mask Editor**.
- The **Cancel** button closes the **Mask Editor** without applying any changes you made to the mask.
- The **Help** button displays online information about the **Mask Editor**.
- The **Apply** button applies the mask settings and leaves the **Mask Editor** open.
- The **Unmask** button deletes the mask and closes the **Mask Editor**. To create the mask again, select the block and choose **Mask > Create Mask**.

See Also

More About

- “Masking Fundamentals”
- “Create a Simple Mask”
- “Block Masks”
- Creating a Mask: Parameters and Dialog Pane (4 min, 19 sec)

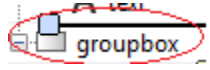
Dialog Control Operations

In this section...

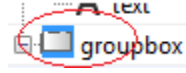
Moving dialog controls in the Dialog box

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:

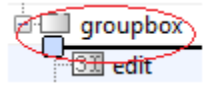
- **Drag and drop on the container dialog control in the Dialog box**
  - **Drop before it**: Adds the dialog control as a sibling before the current dialog control.

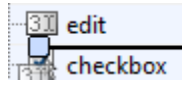
  - **Drop on it**: Adds to the container as a child at the end.

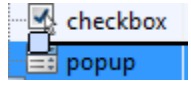
  - **Drop after it**: Adds the dialog control as a sibling after the current dialog control.

- **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.

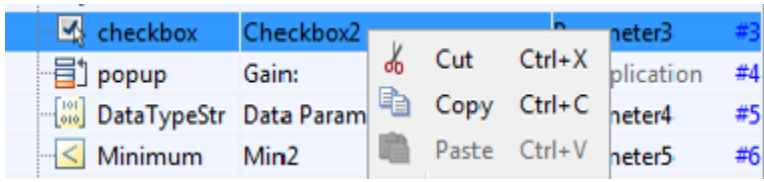
  - **Drop after it**: Adds the dialog control after the current dialog control.

- **Drag and drop into Dialog box blank area**
  - The element is added to the root level node.

Cut, Copy, and Paste Controls

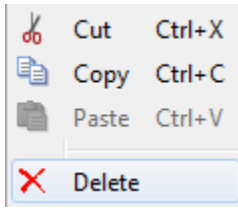
You can cut, copy, and paste dialog controls on the **Dialog box** using the context menu.



Delete nodes

Right-click the control that you want to delete in the **Dialog box**. Select, 

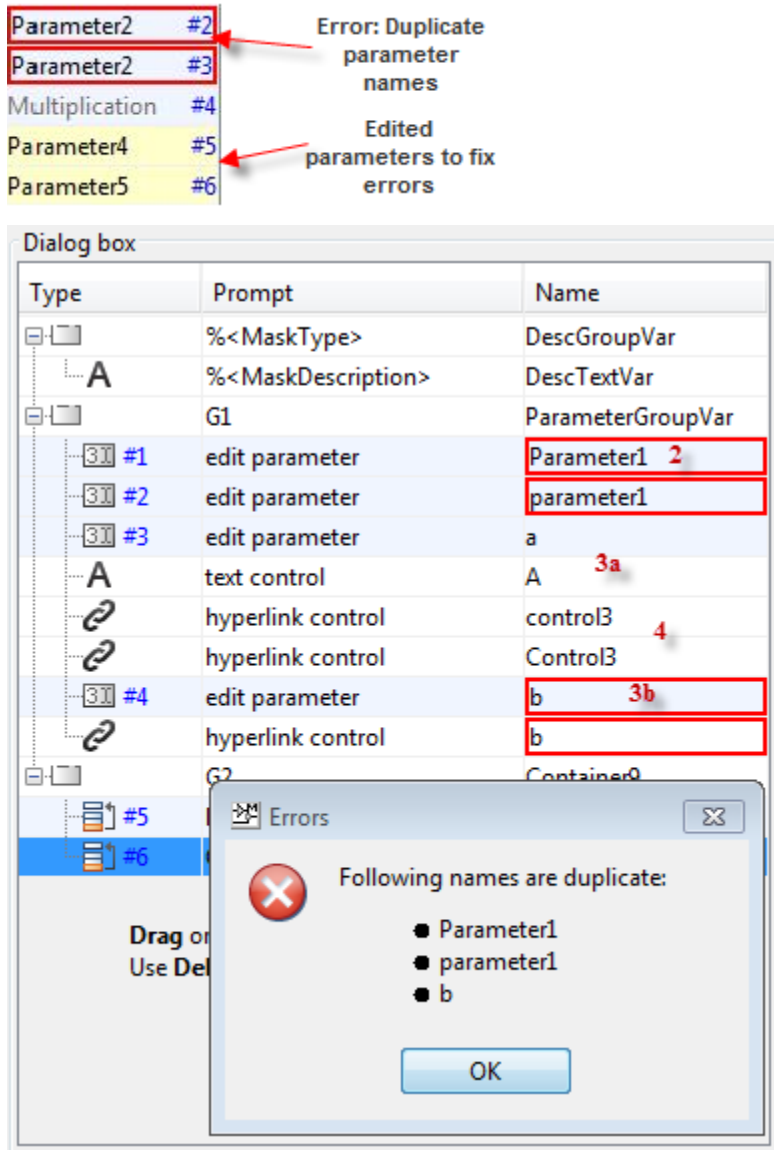
from the context menu. For example, to delete a **Check box** dialog control, right-click and select **Delete**:



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

"Block Masks"

Specify Data Types Using DataTypeStr Parameter

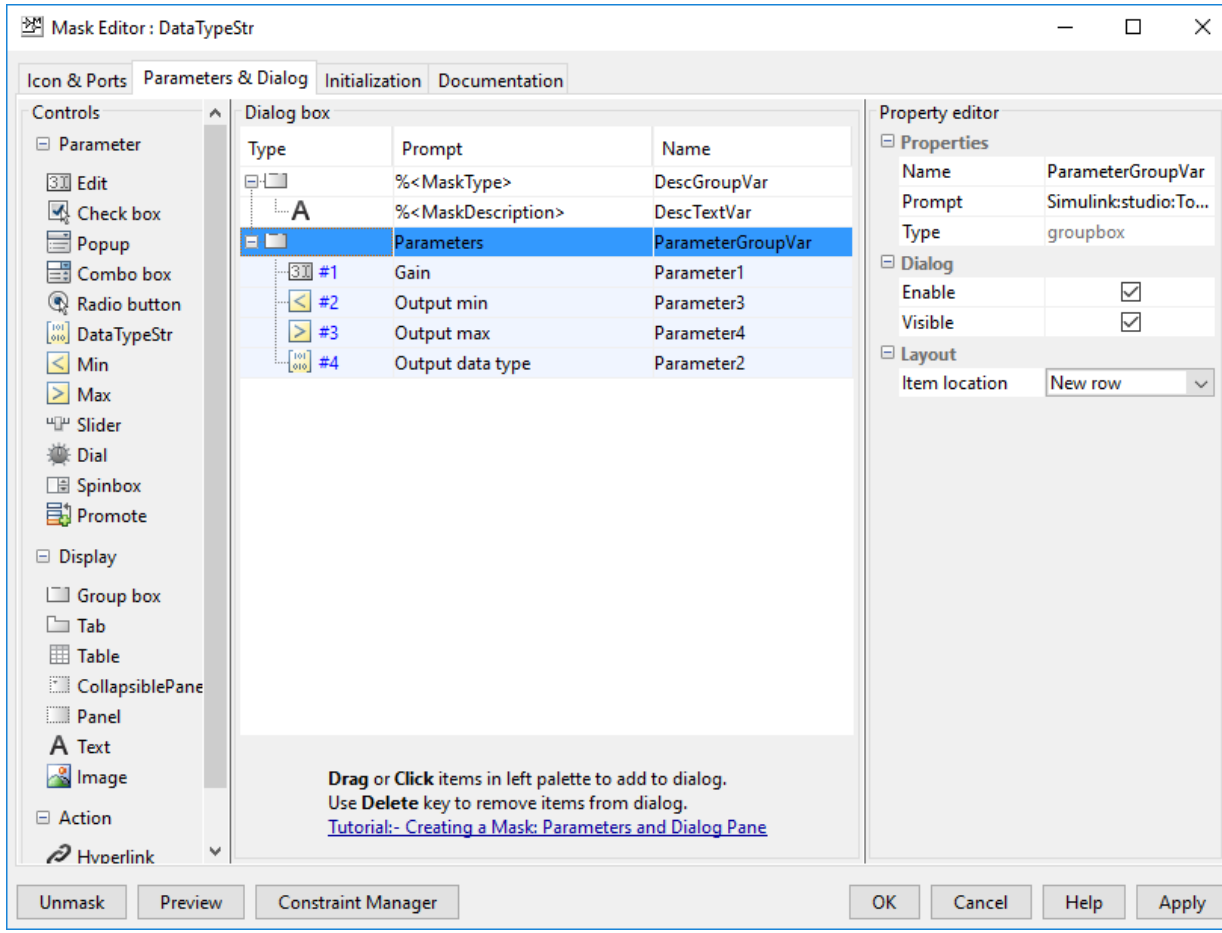
Similar to any mask parameter, the **DataTypeStr** parameter can be added on a mask dialog box from the Mask Editor. Adding the **DataTypeStr** parameter to the mask dialog box allows the end user of the block to specify the acceptable data types for the associated **Edit** type parameter. While defining 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 **DataTypeStr** 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 **DataTypeStr** parameter. **DataTypeStr** 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. Right-click the Subsystem block and select **Mask** > **Create Mask**.

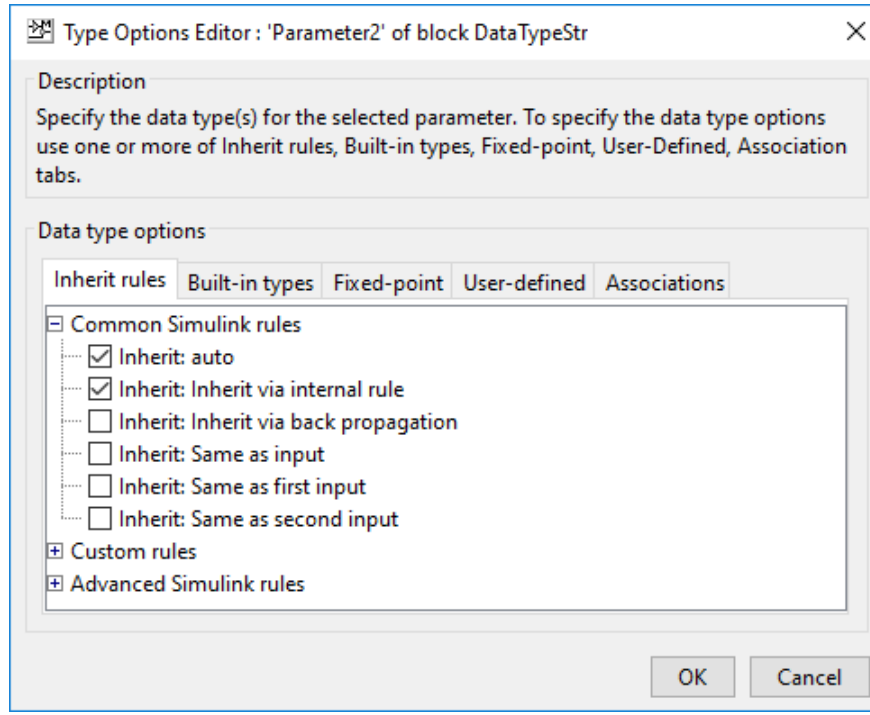
   Note If you are editing an existing mask, to open the Mask Editor, select **Mask** > **Edit Mask**.
3. In the Mask Editor, click the **Parameters & Dialog** pane and add the **Edit**, **Min**, **Max**, **DataTypeStr** parameters.



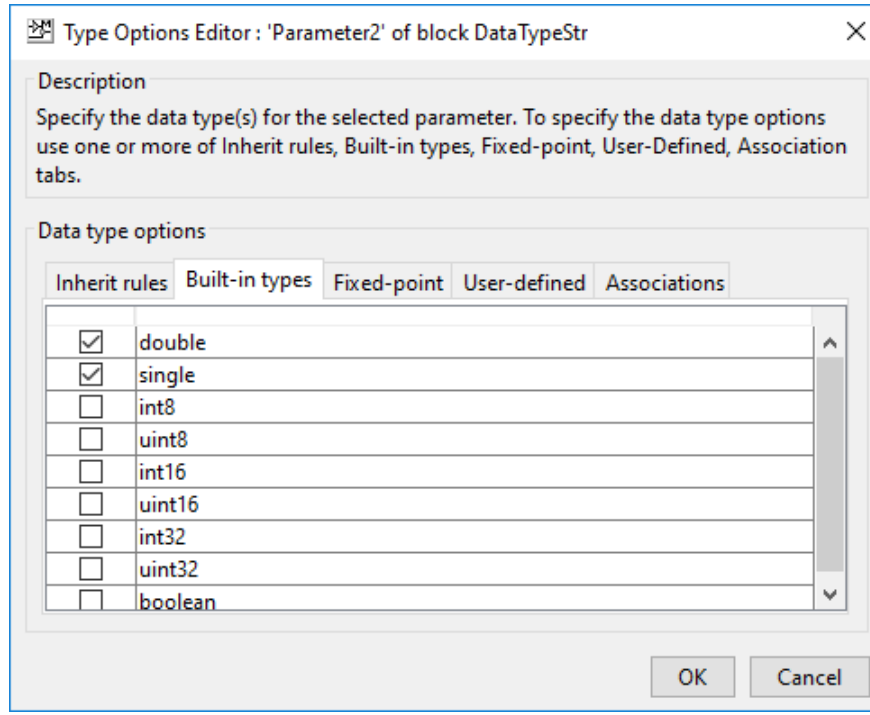
4 To specify data types for the **Edit** parameter, select **DataTypeStr** in the **Dialog box** section of the Mask Editor and click the button next to **Type options** in the **Property editor** pane. The **Type options** editor has a tabbed user interface containing these tabs for data type rules.

   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 Custom rules are listed under **Inherit rules** tab only if there are any custom

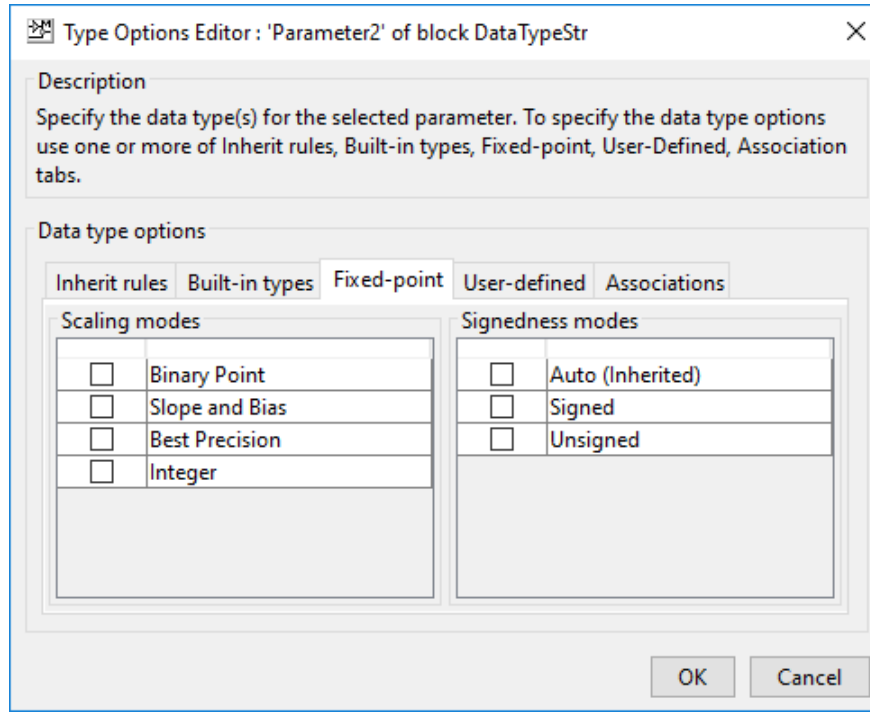
inheritance rules registered on the MATLAB search path. For definitions of some Inherit rules, see “Data Type Inheritance Rules”.



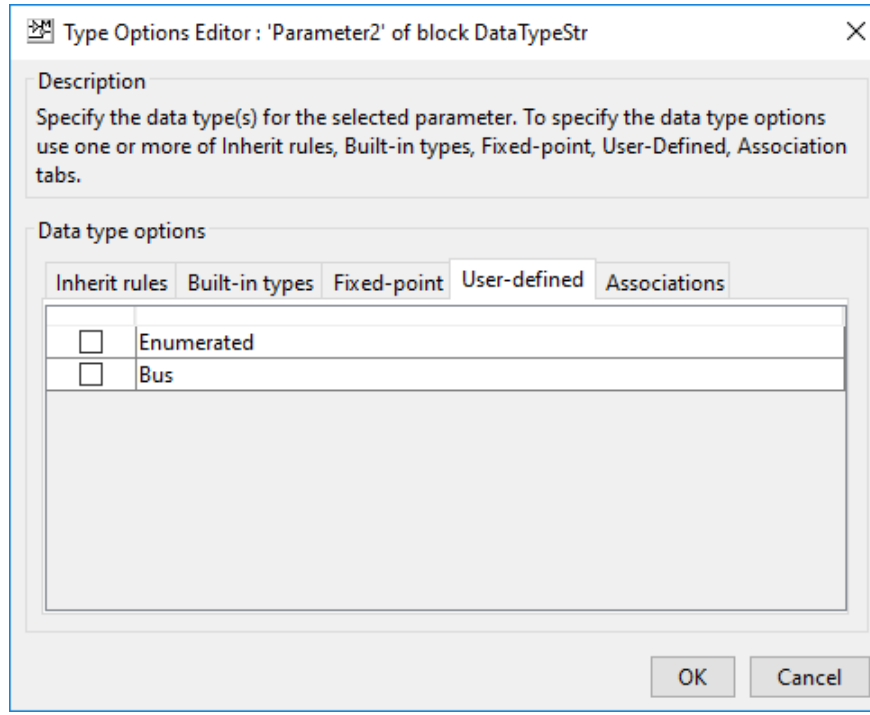
b **Built-in types**: Specify one or more built-in Simulink data types, such as `double` or `single`. For more information, see “Data Types Supported by Simulink”.



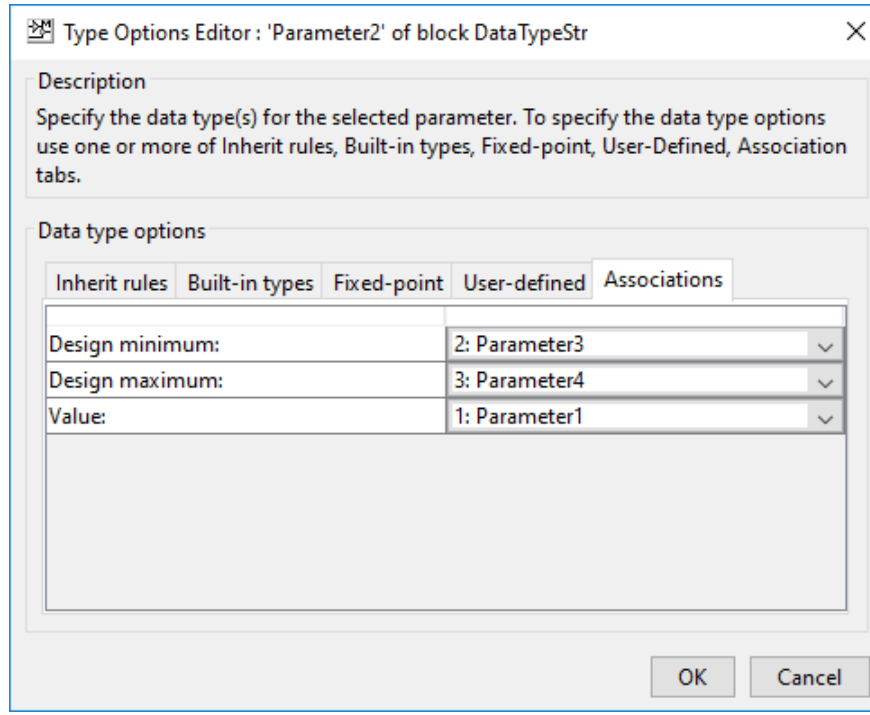
c **Fixed-point**: Specify the scaling and signed modes for a fixed-point data type. For more information, see “Specifying a Fixed-Point Data Type”.



d **User-defined**: Specify a bus object or enumerated (enum) data type, or both. For more information, see “Specify an Enumerated Data Type” and “Specify a Bus Object Data Type”.



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: [1×4 Simulink.MaskParameter]
                   BaseMask: [0×0 Simulink.Mask]
       ParameterConstraints: [0×0 Simulink.Mask.Constraints]
       BlockConstraintRules: [0×0 Simulink.Mask.BlockConstraints]
  ConstraintParamAssociator: [0×0 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: {0×1 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`.

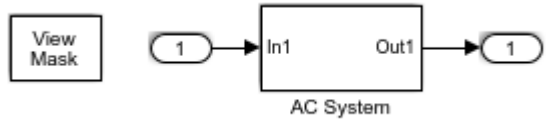
See Also

"Block Masks"

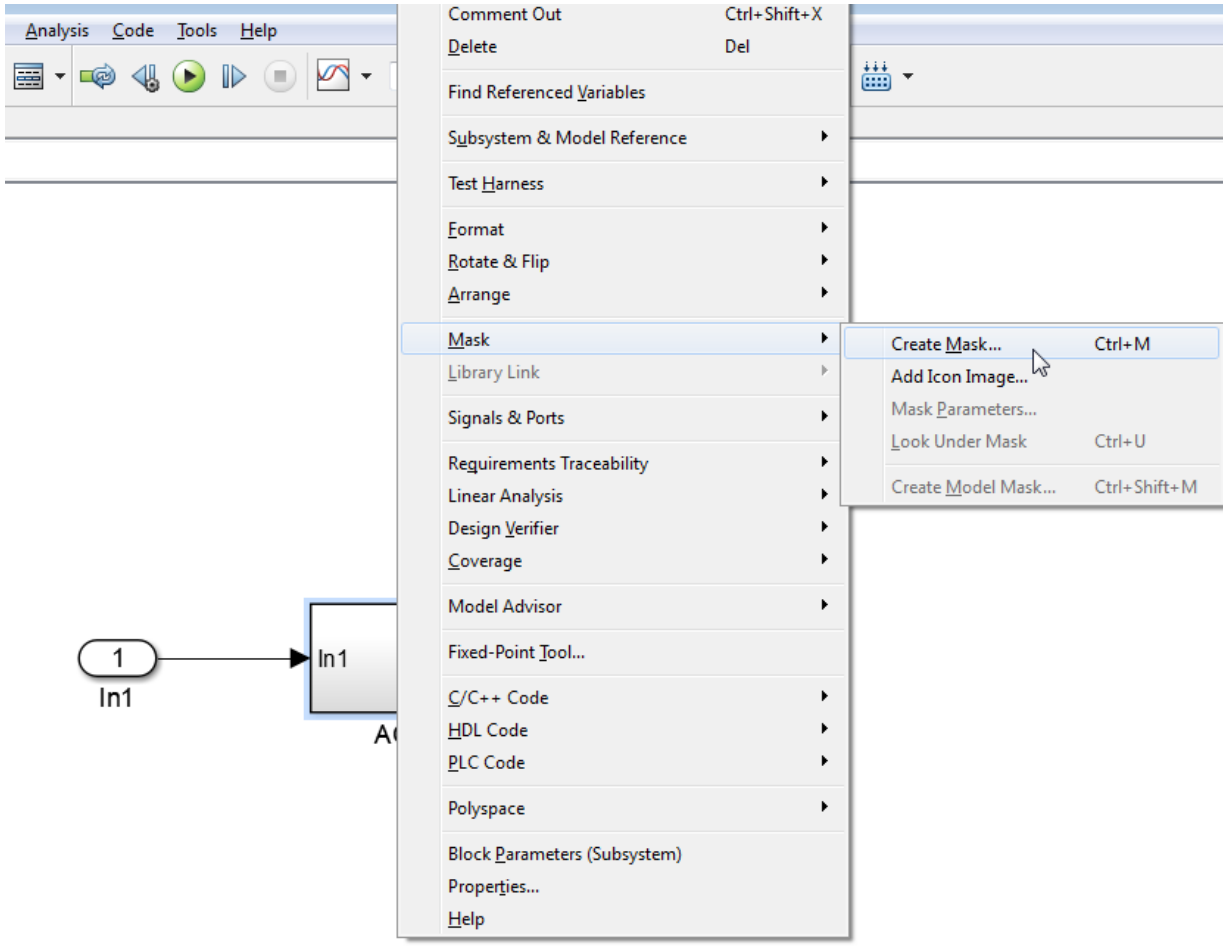
Design a Mask Dialog Box using the Parameters & Dialog Pane

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 block logic and create a custom interface for the block.

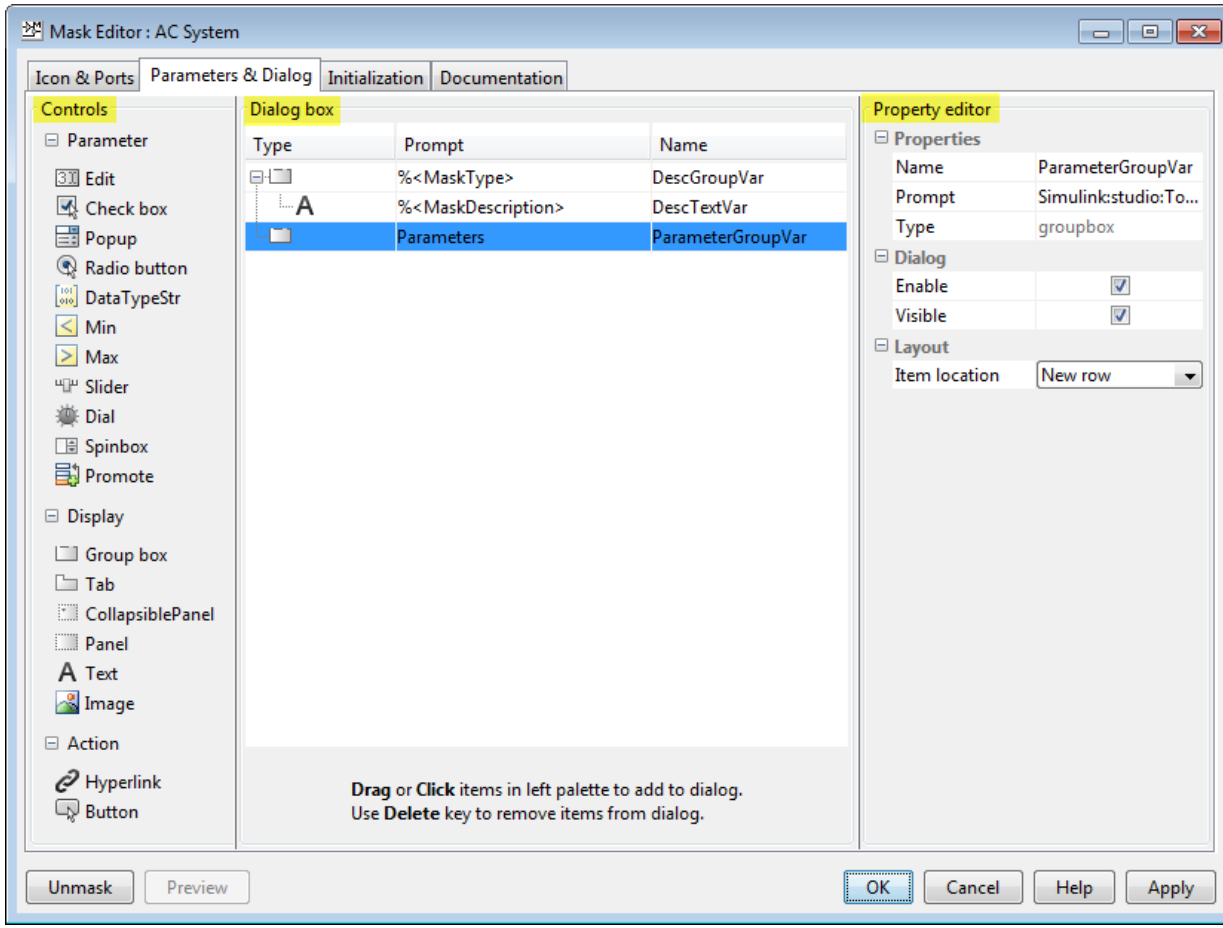
Consider a model containing a Subsystem block called AC system. This Subsystem contains an air conditioning system.



Apply a mask on this subsystem block.



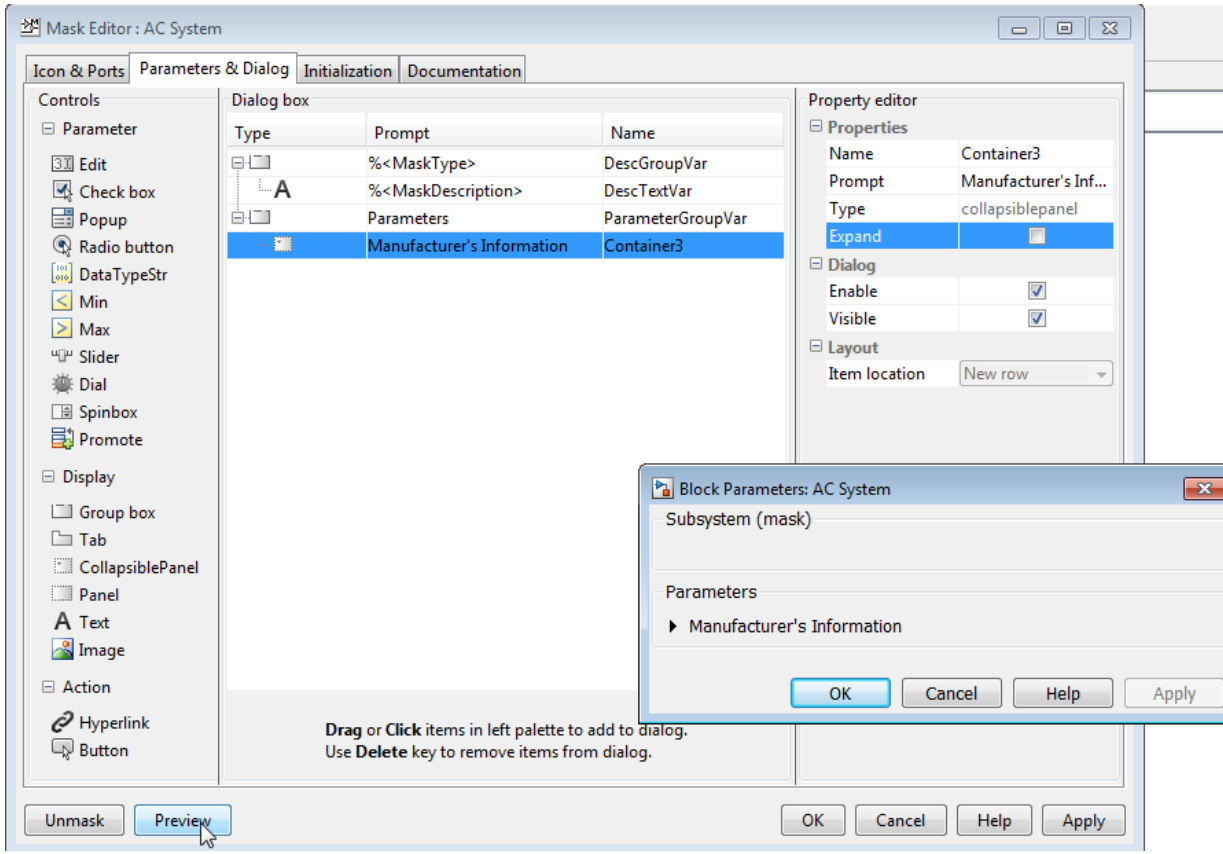
In the Mask Editor, use the Parameters & Dialogs pane to add controls on the mask dialog box and manage the mask dialog box layout. Select items from the **Controls** section to add parameters to the mask dialog box. Use the **Property editor** section to edit parameter properties.



For example, click **Collapsible Panel** from the **Controls** section.

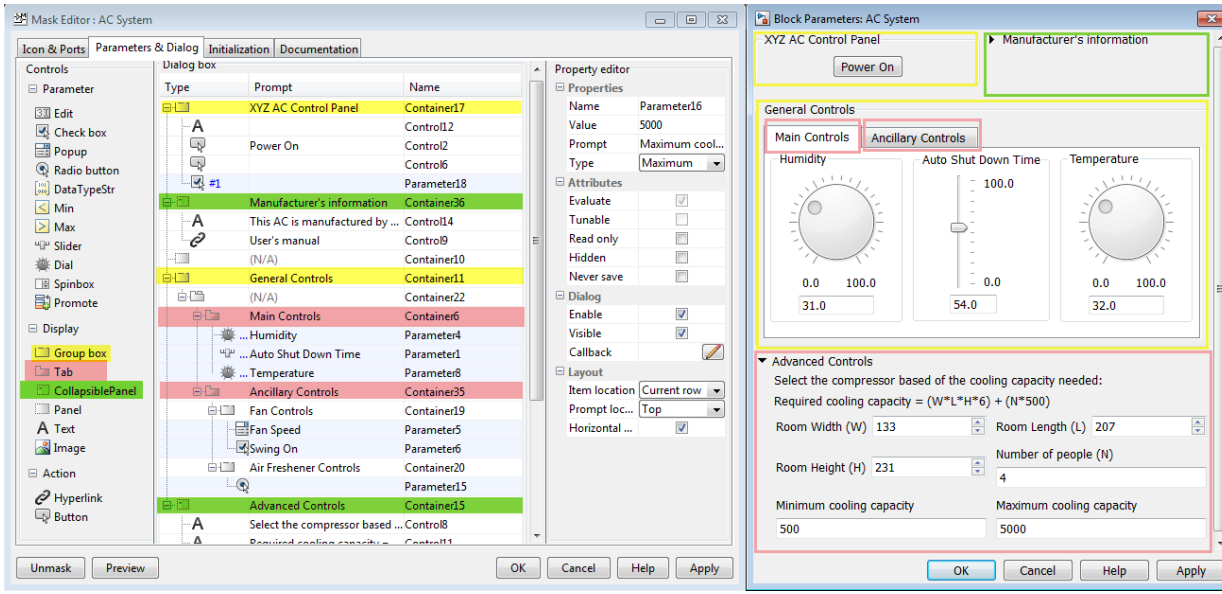
Observe that a collapsible panel container is now added in the **Dialog box** section. In the **Prompt** column, type a value to be displayed on the mask dialog box. For example, `Manufactures Information` . The **Name** column gets populated automatically when a control is added. You can change this value.

Edit the properties of collapsible panel in the **Property editor**. Click **Preview** to view the mask dialog box as you build it.

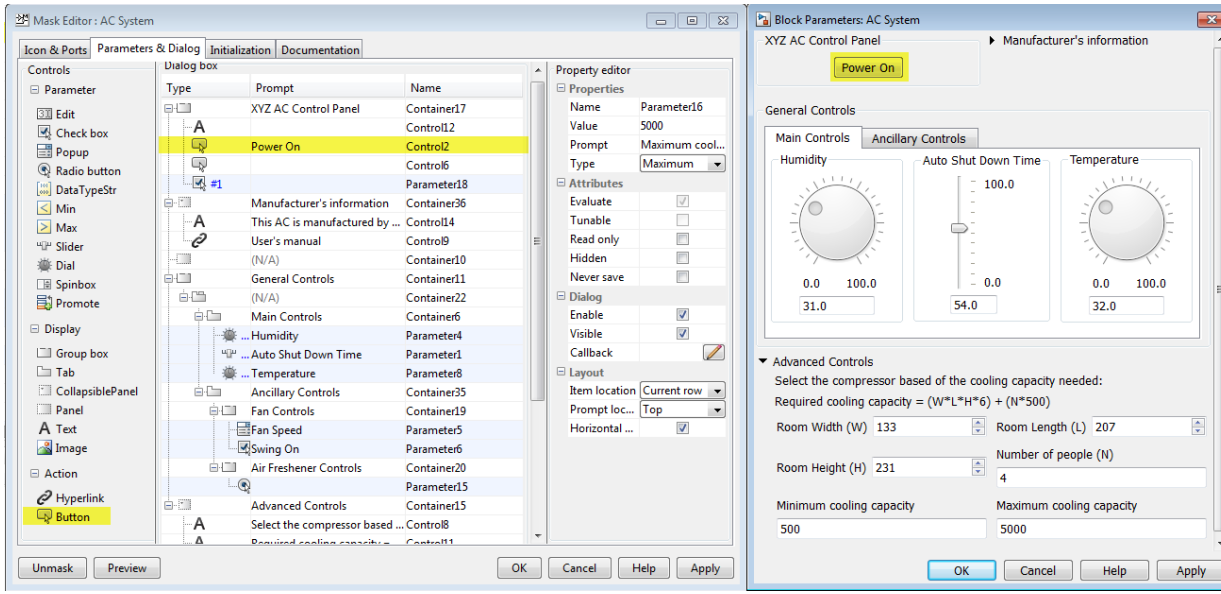


Similarly, you can add and configure various controls from the Mask Editor to build the mask dialog box.

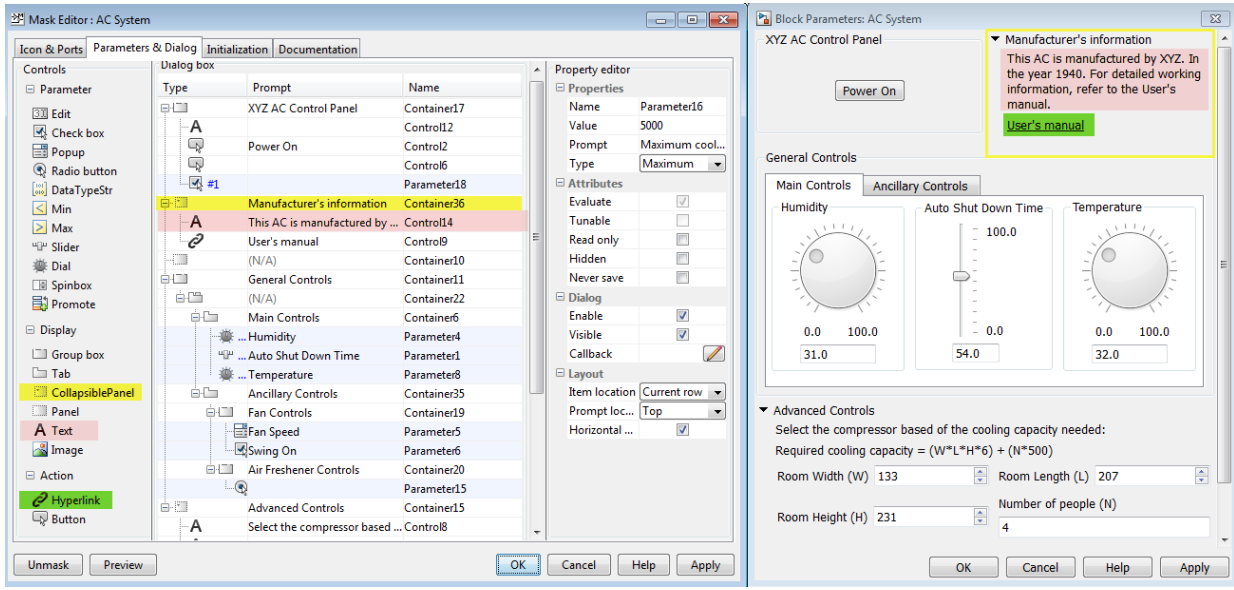
Observe the mask layout. Containers like group boxes, panels, collapsible panels, and tabs group the controls together. Here, yellow represents **Group Box**, pink represents **Tab**, and green represents the **Collapsible Panel**.



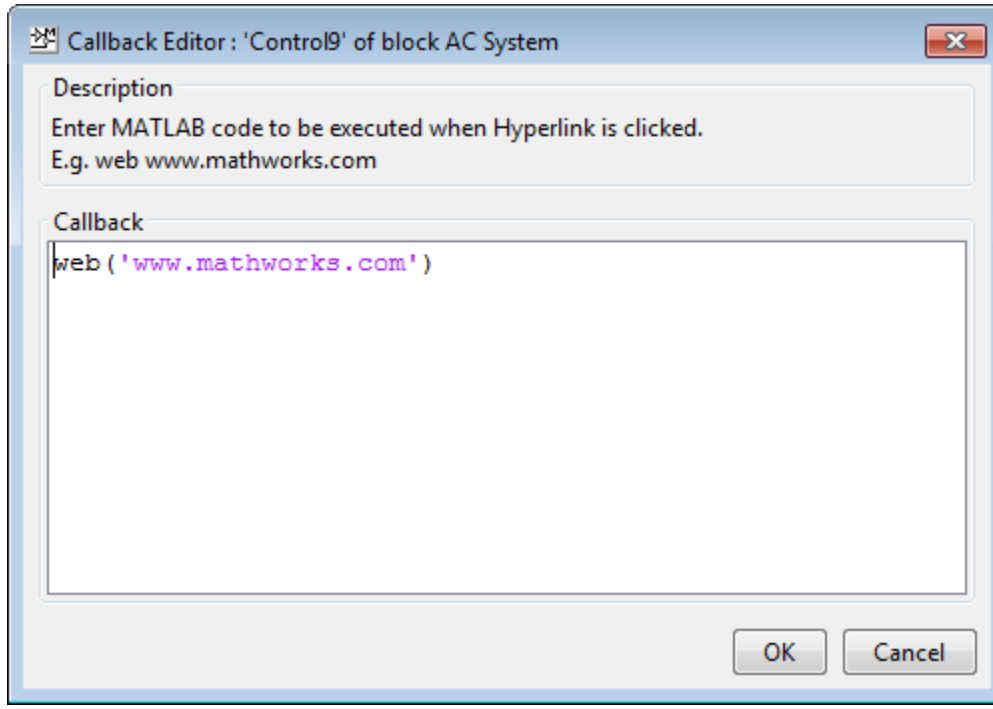
The **Button** controls type is used to create the **Power On** button on the mask dialog box. To manage the button placement, apply the Horizontal Stretch property. You can also add callback code to be executed when the button is pressed.



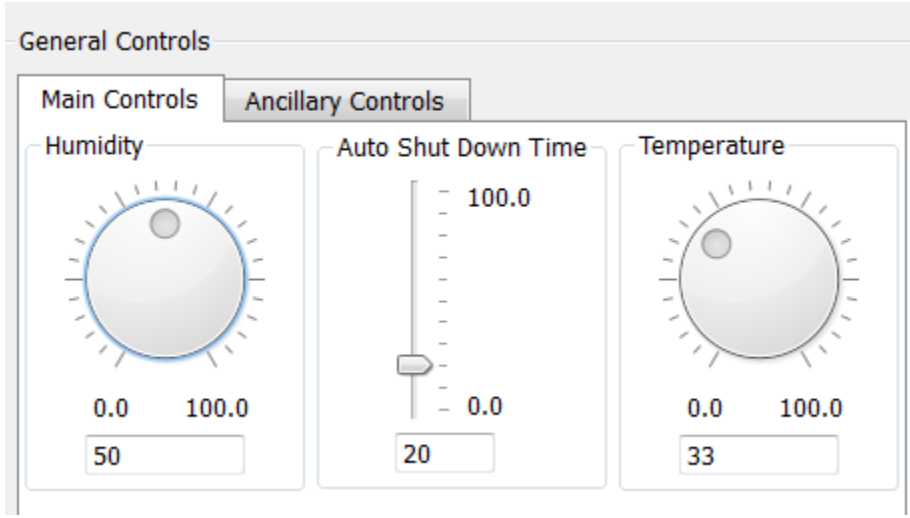
The collapsible panel for **Manufacturer's information** contains **Text** and **Hyperlink** control types.



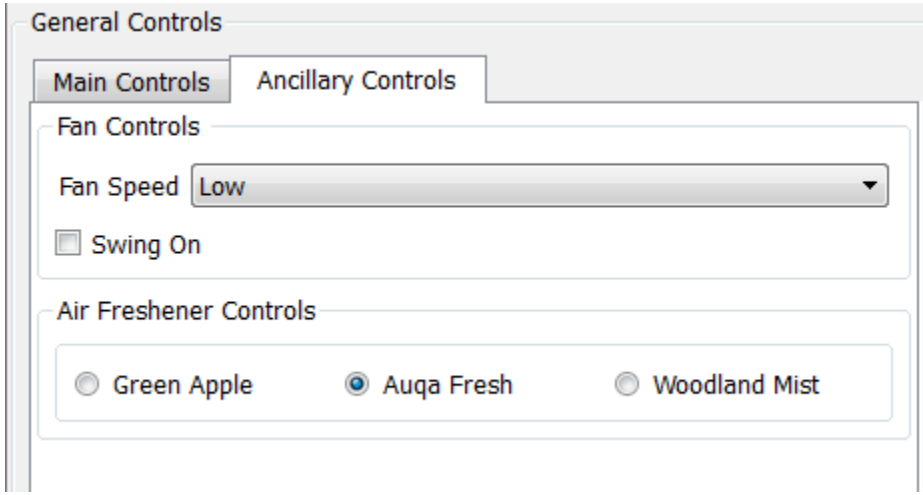
You can add MATLAB code as a callback for the hyperlink.



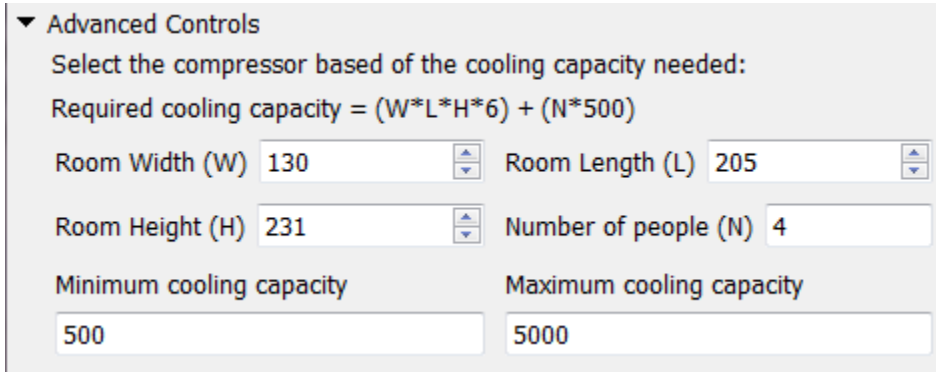
The **General Controls** section contains tabs to segregate and categorize information under **Main Controls** and **Ancillary Controls**. The **Main Controls** tab uses **Dials** and **Slider** to accept inputs for air conditioner parameter. You can edit the property of dial and slider in the property 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.



See Also

More About

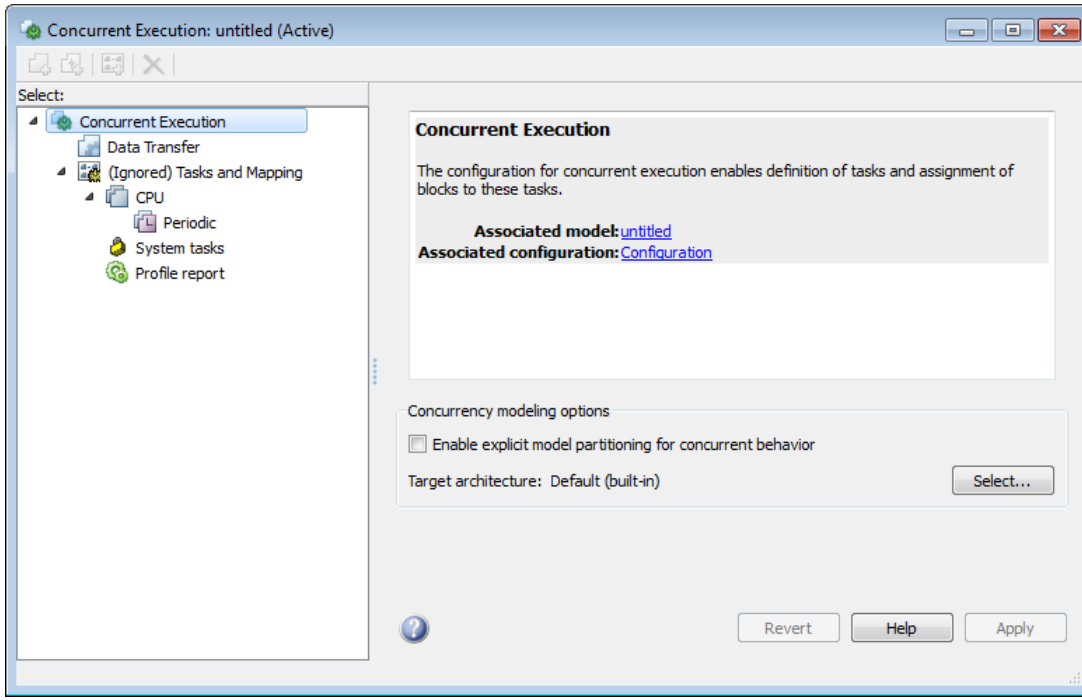
- “Mask Editor Overview” on page 21-2
- “Mask Editor Overview” on page 21-2

Concurrent Execution Window

Concurrent Execution Window: Main Pane

In this section...

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 22-6

  Configure data transfer methods between tasks.

- Tasks and Mapping

  Map blocks to tasks.
- "CPU Pane" on page 22-10

  Set up software nodes.
- Periodic on page 22-13

  Name periodic tasks.
- Task on page 22-16

  Define and configure a periodic task that the target operating system executes.
- Interrupt on page 22-19

  Define aperiodic event handler that executes in response to hardware or software interrupts.
- System Task Pane on page 22-23

  Display system tasks.
- System Task on page 22-24

  Display periodic system tasks.
- System Interrupt on page 22-27

  Display interrupt system tasks.
- "Profile Report Pane" on page 22-29

  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 Configuration Settings dialog.

1. Open the **Configuration Settings** from the **Simulation** menu of the current model.
2. Expand the **Additional Parameters** section and select **Allow tasks to execute concurrently on target**.

This option is only visible 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.

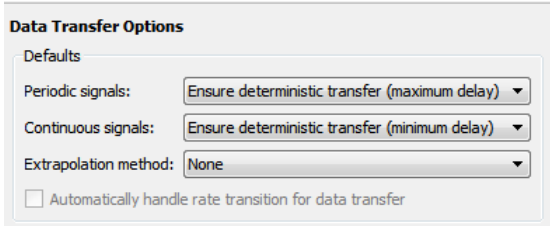
Clearing this check box

- 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



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: `FPGA`*`N`*

- 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

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”.

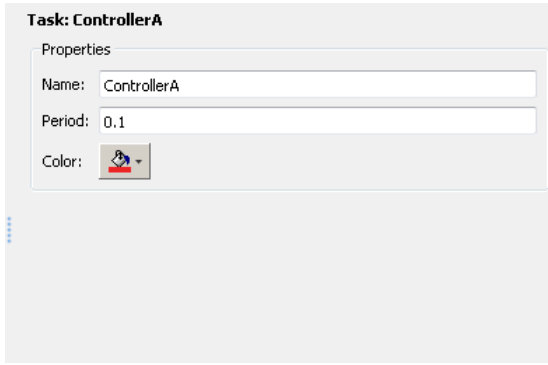
See Also

- “Define a Custom Architecture File”
- “Configure Your Model for Concurrent Execution”

Task Pane

In this section...

Task Pane Overview



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 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

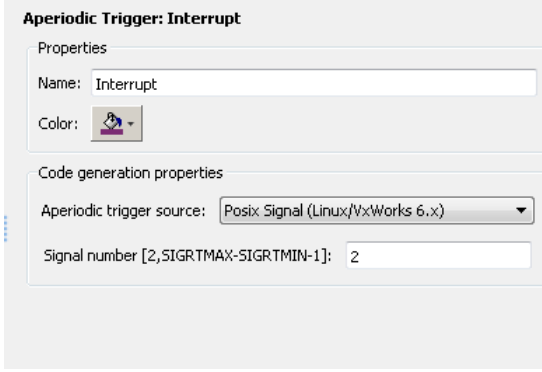
“Configure Your Model for Concurrent Execution”

Interrupt Pane

In this section...

Interrupt Pane Overview

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 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”.

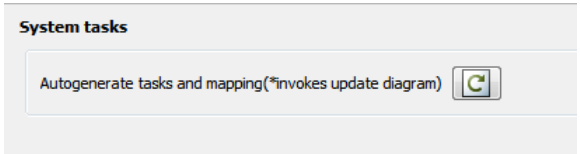
See Also

“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

Display periodic system tasks.

See Also

“Configure Your Model for Concurrent Execution”

Name

Specify a default name for the periodic system task configuration.

Settings

Default: `DiscreteN`

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

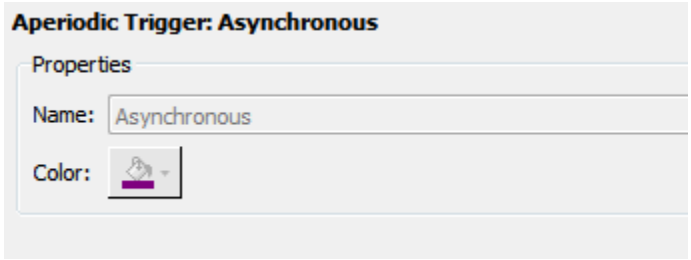
"Configure Your Model for Concurrent Execution"

System Interrupt Pane

In this section...

System Interrupt Pane Overview

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

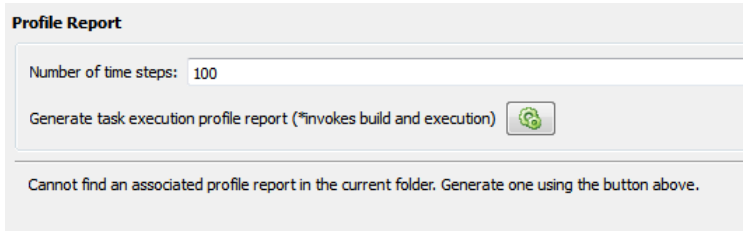
"Configure Your Model for Concurrent Execution"

Profile Report Pane

In this section...

Profile Report Pane Overview

Generate and examine profile report for model.



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

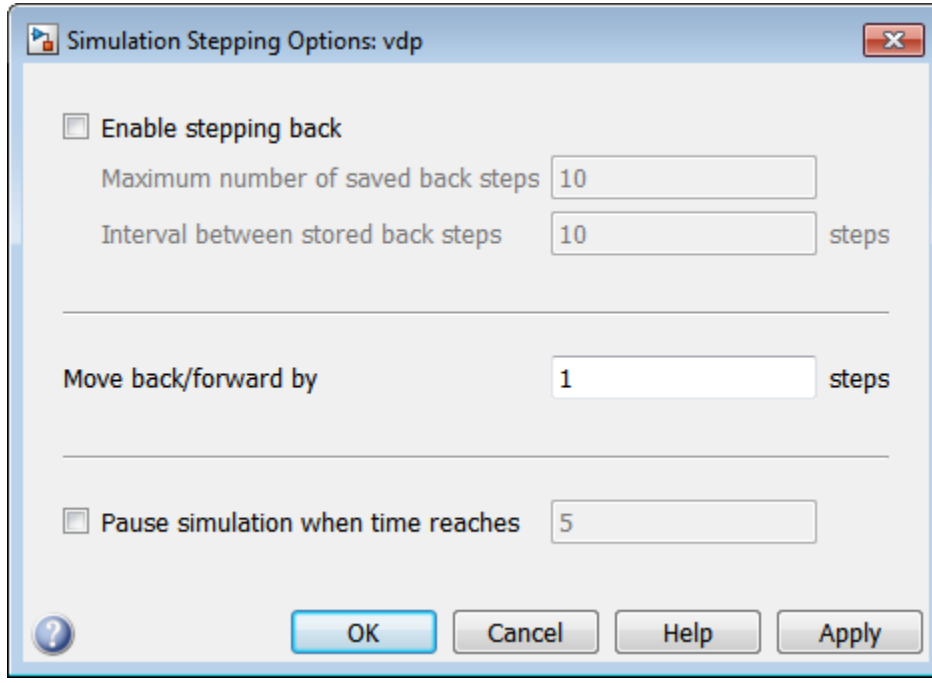
See “Programmatic Interface for Concurrent Execution”.

See Also

“Configure Your Model for Concurrent Execution”

Simulink Simulation Stepper

Simulation Stepping Options



In this section...

Simulation Stepping Options Overview

Use the Simulation Stepping Options dialog box to configure the time and the manner of manually stepping through a simulation.

Configuration

This pane appears when you select **Simulation > Stepping Options**.

1. Set the time at which you wish to pause the simulation
2. To step backwards through a simulation, select **Enable stepping back** and specify the total number and frequency of snapshots.
3. Specify the increment of steps by which the simulation steps either forward or backwards.
4. To pause simulation at a particular time, select **Pause simulation when time reaches** check box and enter the pause time.

Tips

- To start the Simulation Stepping Options dialog box from the Simulink toolbar, click  .
- 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”

Pause simulation when time reaches

Pause simulation when time reaches the specified time(s).

Settings

Default: Off

 On

Enable stepping back.

 Off

Disable stepping back.

Selecting this check box enables the associated text box. In this text box, enter the time at which simulation is to be paused.

Default: 5

Minimum: 0

- This value can be a scalar value, or a vector of times. Specifying a vector of pause times is equivalent to specifying multiple separate pause times for a single simulation.

  You can specify pause times as variables in the model or MATLAB workspace.
- The stepper does not alter the course of the simulation. As a consequence, specifying a value for a pause time does not necessarily pause the simulation at exactly that time. Instead, the simulation pauses at whatever simulation time is closest to the requested pause time, without going below it.

See Also

“How Simulation Stepper Helps With Model Analysis”

Simulink Variant Manager

Variant Manager Overview

In this section...

The Variant Manager is a central tool that allows you to manage various variation points that are modeled using variant blocks in a system model.

A model hierarchy may contain several variant blocks, each with many variant choices, combinations of which correspond to particular configurations of the system. Switching between variant choices and validating them manually can be complicated and erroneous.

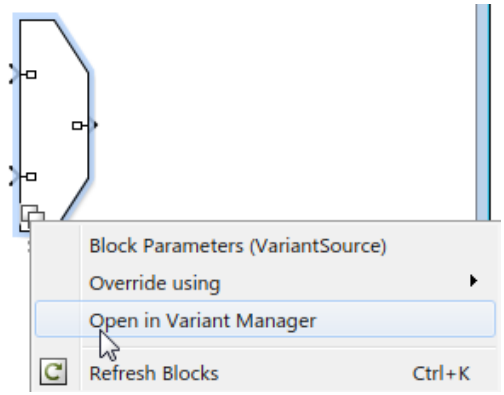
Use the Variant Manager to create predefined configurations for a model, and use the model under any of the configurations. You can create the configurations by combinations of different variant choices across the model hierarchy.

Using the Variant Manager, you can:

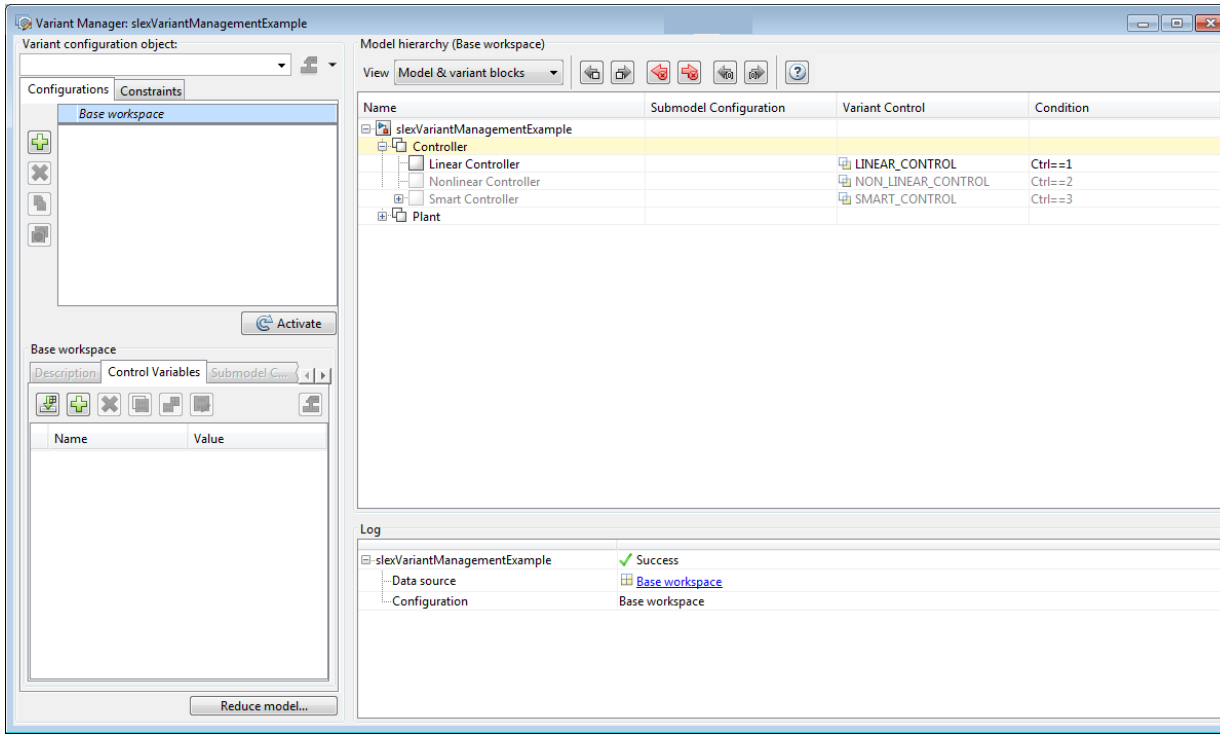
- Define, validate, and visualize variant configurations.
- Define and validate constraints for the model.
- Specify the default active configuration.
- Set control variables to either integer, enumeration values, or `Simulink.Parameter` objects.
- Associate `Simulink.VariantConfigurationData` object with the model.
- Validate a variant configuration or model without updating the model.
- Reduce a variant configuration to reduce the model.

Consider the model Variant Management. To open the Variant Manager, you can:

- Right-click the variant badge and select **Open in Variant Manager**.



- Right-click the variant block, and in the context menu, click **Variant > Open in Variant Manager**.
- Select the variant block, and click **Diagram > Variant > Open in Variant Manager**.
- Click **Open block in Variant Manager** available on the variant block's **Block Parameter** dialog box.

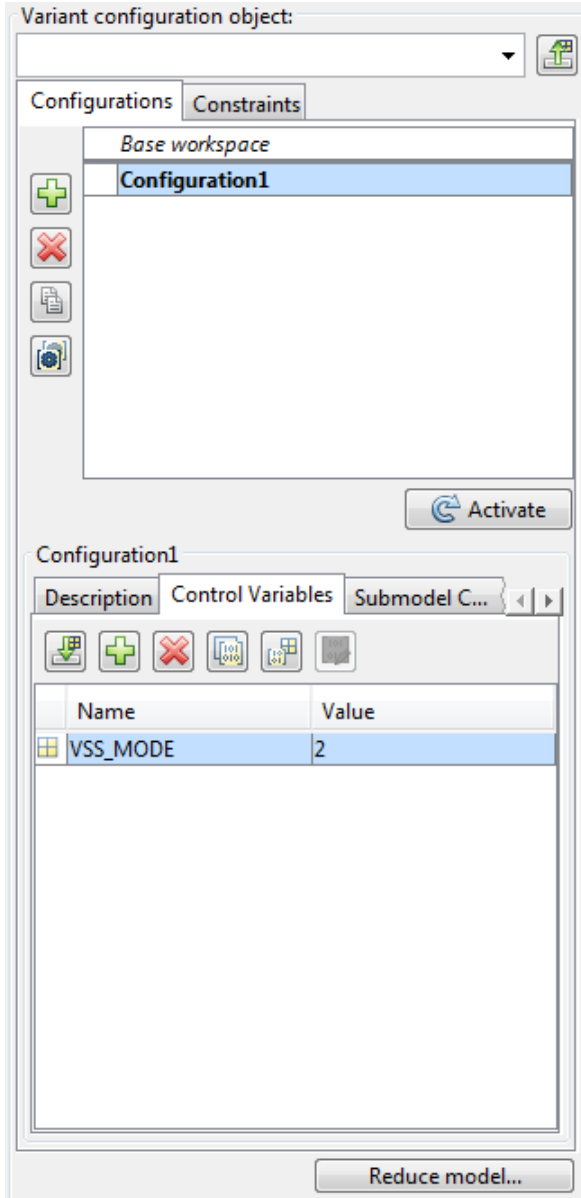


The **Variant Manager** window is divided into three panes:

- The “Variant Configuration Data” on page 24-4 pane which enables you to define variant configurations and constraints, and export them as variant configuration data objects.
- The “Model Hierarchy” on page 24-9 pane which enables you to visualize the variant hierarchy.
- The “Log” on page 24-13 pane which displays information on the source of control variables and validation errors.

Variant Configuration Data

Use this pane to create configurations, define control variables, associate referenced model configurations, and define constraints. The configurations and associated data are stored in a variant configuration data object.


Variant configuration object:
Configurations
Constraints
Base workspace
Configuration1
Activate
Configuration1
Description
Control Variables
Submodel C...
Name
Value
VSS_MODE
2
Reduce model...

Variant Configuration Object

After you add a variant configuration, type a name for the variant configuration object in the **Variant configuration object** box. You can use the drop-down menu to load a variant configuration object from a file or refresh a loaded variant configuration object. You can load a variant configuration object either as a `MAT` file or a MATLAB script (`.m` file). To store the variant configuration object in the base workspace and associate it with the model, click **Export**  . To store the variant configuration object in a folder of your choice either as a `MAT`-file or a MATLAB script (`.m` file), click **Save As** from the drop-down menu of the **Export** button.

Configurations

The **Configurations** tab is divided into upper and the lower panes. You can use the upper pane to add, delete, or copy a variant configurations. You can also set a default configuration.

The upper pane has these buttons to manage a variant configuration.

Button	Description
	Add variant configuration
	Delete variant configuration
	Duplicate variant configuration
	Set/Clear default active configuration

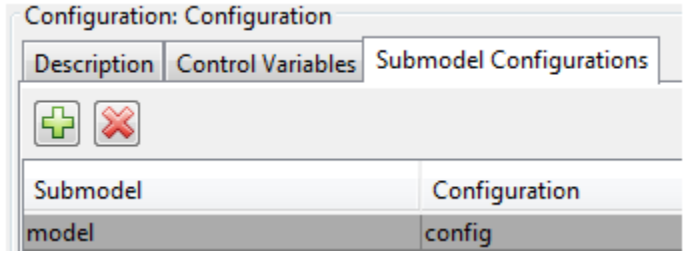
After a variant configuration is added, you can use the lower pane on the **Configurations** tab to add a description, control variables, and the submodel configurations for it. You can add control variables and export them to the base workspace even when a variant configuration is not added. The lower pane contains these tabs:

- **Description** — Provide a description for the selected variant configuration.
- **Control Variables** — Add, delete, copy or import control variables. Toggle data type and import control variables from the workspace.

Button	Description
	Add control variable
	Delete the selected control variable
	Create a copy of the selected control variable
	Toggle type of a control variable A control variable can be either a character vector or a `Simulink.Parameter` object.
	Edit `Simulink.Parameter` control variables. This option gets activated when the selected control variable is a `Simulink.Parameter` object.
	Import control variables from base workspace

- **Submodel Configurations** — Define variant configuration for a referenced model.

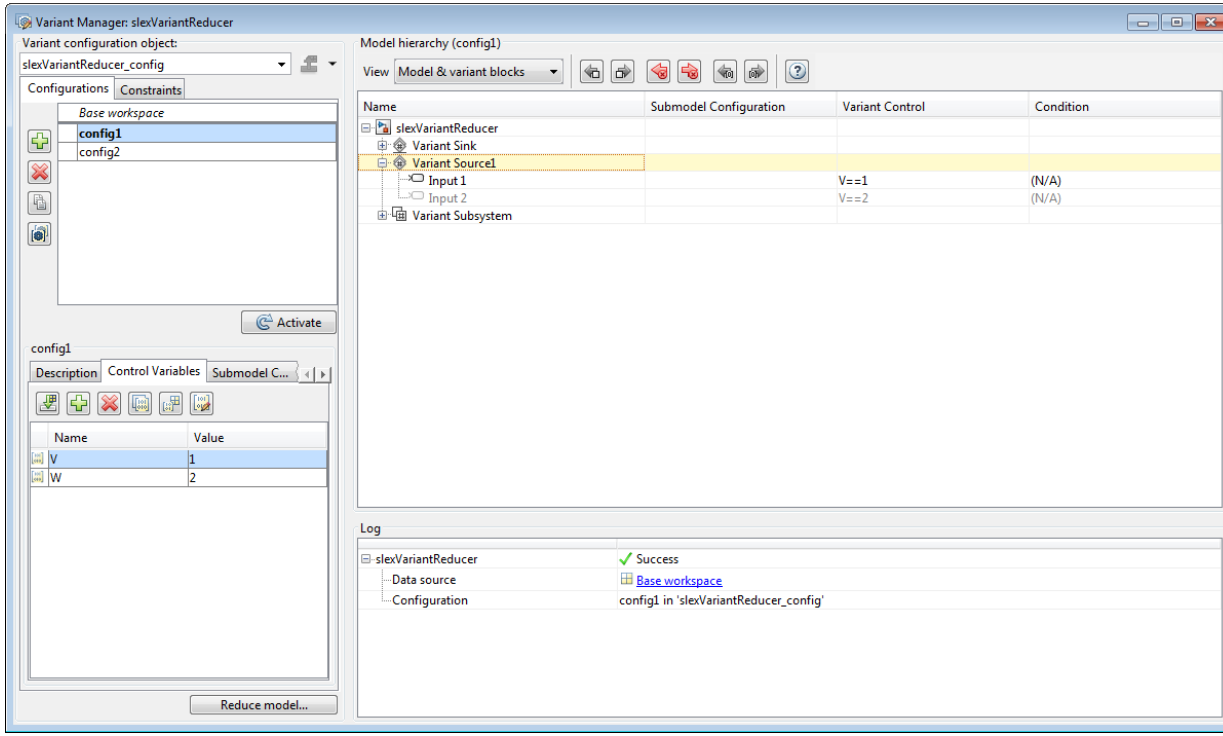
  Add  or delete  a referenced model configuration.



Activate Configuration

To refresh and activate the variant model for a configuration, select a configuration from the list of **Configurations** and click **Activate**. If you click the **Activate** button without selecting any configuration, the values for the **Control Variables** are picked from the base workspace. For such cases, the **Control Variables** that are defined in the Variant Manager overrides the corresponding values in the base workspace.

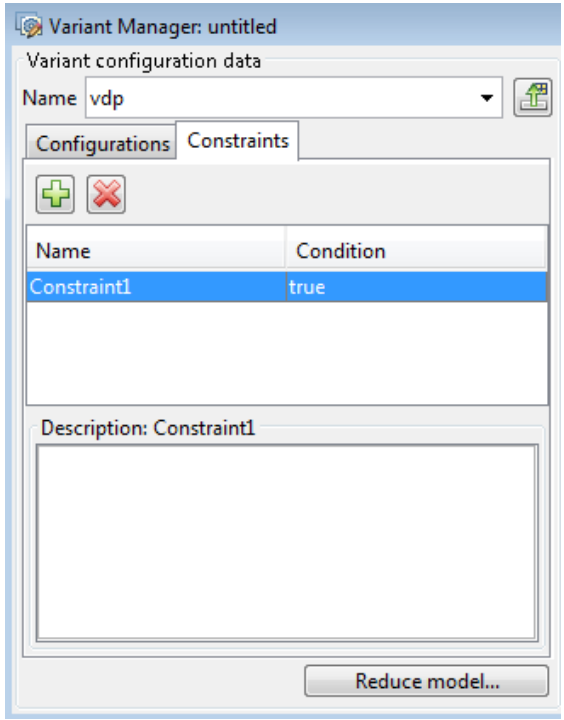
When you open the Variant Manager for a parent model that contains referenced models (Submodel), only the parent model is validated. The referenced models (Submodel) are validated only when you activate or expand (click +) the reference models.



Constraints

Use the **Constraints** tab to add or delete the model-level constraints. Similar to the **Configurations** tab, the **Constraints** tab also contains upper and lower panes.

The upper pane displays the name and condition of the constraints while the lower pane displays the description. The condition expression of the constraint must be satisfied by all variant configurations in the model.



Model Hierarchy

You can visualize and explore the variant hierarchy of a model and edit the properties of variant blocks, variant choices, and variant objects from the **Model hierarchy** pane. This pane displays the **Name**, **Submodel Configuration**, **Variant Control**, and associated **Conditions** of variant objects used as variant controls.

Browse the hierarchy using the navigation icons. The controls on the **Model hierarchy** pane allow you to perform the following actions:

- Refresh and validate hierarchy.
- Display only variant blocks.
- Navigate among active, invalid, and overridden variant choices.

Show

Selectively display blocks in the variant hierarchy:

- Select **Model and variant blocks** to display only model reference and variant blocks.
- Select **All hierarchical blocks** to display all hierarchical blocks in the model.

Hierarchy Table

The model hierarchy is displayed in a tree, with each block representing a node in the hierarchy. The hierarchy displays active, inactive, overridden, and invalid variants. You can edit referenced model configurations, variant controls, and variant conditions. Expand nodes to view the underlying blocks.

Note Protected reference models cannot be viewed in the hierarchy.

The hierarchy table consists of these columns:

- **Name** — Name of the model or block.
- **Submodel Configuration** — Configurations used by referenced models. You can only edit the **Submodel Configuration** for rows that display models referenced by the top model.
- **Variant Control** — Variant control parameter of a variant choice. This column is identical to the Variant Control column of the parameter dialog box of variant blocks. You can edit this column for variant choices across the hierarchy.
- **Condition** — Displays and allows you to edit the condition for the `Simulink.Variant` object when it is used as variant control. You can edit this column for variant choices across the variant hierarchy.

Tip Right-click the item on the hierarchy table and use the **Override using this Choice** or the **Open Parent Block Parameters** options on the context menu, as necessary.

In the model hierarchy section, each block is represented with an icon. The following table displays the icons and the corresponding block name.

Icon	Block Name
	Model Block

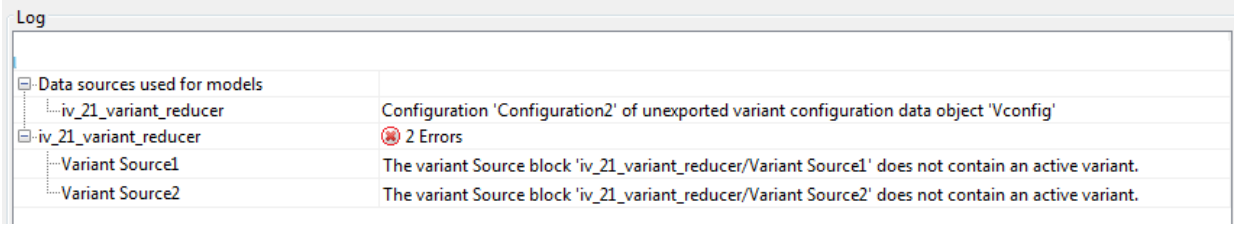
Icon	Block Name
	Inline Variants Block (Variant Source and Variant Sink)
	Variant Subsystem block
	Subsystem block
	Model Variant block
	Simulink Function block
	Trigger Port block
	Variant Sink output port
	Variant Source input port
	Variant Subsystem block with **Propagate conditions outside of variant subsystem** option selected.
	Variant Subsystem block with **Analyze all choices during update diagram and generate preprocessor conditionals** option selected.
	Variant Subsystem block with **Override variant conditions and use the following variant** option selected.
	Variant Subsystem block with **Propagate conditions outside of variant subsystem** and **Analyze all choices during update diagram and generate preprocessor conditionals** options selected.

Icon	Block Name
	Variant Subsystem block with **Propagate conditions outside of variant subsystem** and **Override variant conditions and use the following variant** options selected.
	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 **Override variant conditions and use the following variant** option selected.
	Inline Variants Block (Variant Source and Variant Sink) with **Analyze all choices during update diagram and generate preprocessor conditionals** option selected.
	Inline Variants Block (Variant Source and Variant Sink) with **Allow zero active variant control** and **Analyze all choices during update diagram and generate preprocessor conditionals** options selected.
	Initialize Function block
	Event Listener block of Initialize Function block
	Reset Function block
	Event Listener block of Rest Function block
	Terminate Function block
	Event Listener block of Terminate Function block

Log

The Log pane displays information and validation results of the source of control variables for the models in the hierarchy.

For example, if a variant configuration is used for a referenced model, the referenced model name is displayed in the row along with name of the variant configuration data object and variant configuration. The pane also displays errors encountered during validation of the variant configuration.



See Also

Related Examples

- “Create and Validate Variant Configurations”
- “Import Control Variables to Variant Configuration”
- “Define Constraints”