Simulink®

Simulation and Model-Based Design

- Modeling
- Simulation
- Implementation

Simulink® Reference

Version 6

How to Contact The MathWorks:

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup

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

508-647-7000 Phone

508-647-7001 Fax

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

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

Simulink® Reference
© COPYRIGHT 2000–2005 by The MathWorks, Inc.

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

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

Trademarks

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Patents

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

Revision History

July 2002	Online only	Revised for Simulink 5 (Release 13)
April 2003	Online only	Revised for Simulink 5.1 (Release 13SP1)
April 2004	Online only	Revised for Simulink 5.1.1 (Release 13SP1+)
June 2004	Online only	Revised for Simulink 6 (Release 14)
October 2004	Online only	Revised for Simulink 6.1 (Release 14SP1)
March 2005	Online only	Revised for Simulink 6.2 (Release 14SP2)
September 2005	Online only	Revised for Simulink 6.3 (Release 14SP3)

Contents

Blocks — Categorical List

1

What Each Block Reference Page Contains **1-2**

Blocks by Category **1-4**
Commonly Used **1-4**
Continuous **1-5**
Discontinuities **1-5**
Discrete **1-6**
Logic and Bit Operations **1-6**
Lookup Tables **1-7**
Math Operations **1-7**
Model Verification **1-8**
Model-Wide Utilities **1-8**
Ports & Subsystems **1-8**
Signal Attributes **1-9**
Signal Routing **1-9**
Sinks **1-10**
Sources **1-10**
User-Defined Functions **1-10**
Additional Math & Discrete **1-11**

Blocks — Alphabetical List

2

Linearization and Trimming Commands

3

Linearization and Trimming Commands — Alphabetical List **3-2**

4 Model Construction Commands

Task-Oriented Command List **4-2**

Model Construction Commands — Alphabetical List . . **4-4**

5 Simulation Commands

Task-Oriented Command List **5-2**

Simulation Commands — Alphabetical List **5-3**

6 Mask Icon Drawing Commands

Command Summary **6-2**

Mask Icon Drawing Commands — Alphabetical List . . **6-3**

7 Simulink Debugger Commands

Command Summary **7-2**

Simulink Debugger Commands — Alphabetical List . . **7-5**

8 Data Type Functions

Data Type Functions — Alphabetical List **8-2**

9 Data Object Classes

Class Summary **9-2**

Classes — Alphabetical List **9-5**

10 Model and Block Parameters

Model Parameters **10-2**
Examples of Setting Model Parameters **10-40**

Common Block Parameters **10-41**
Examples of Setting Block Parameters **10-50**

Block-Specific Parameters **10-51**

Mask Parameters **10-145**
Setting Mask Parameters **10-149**
How Masked Parameters are Stored **10-149**

11 Model File Format

Model File Contents **11-2**

Model Section 11-3
BlockDefaults Section 11-3
AnnotationDefaults Section 11-3
System Section 11-3

12 Embedded MATLAB Function Block Reference

Supported Variable Types in Embedded MATLAB Functions **12-2**

Operators in Embedded MATLAB Functions **12-4**
Control Flow Statements in Embedded MATLAB Functions **12-4**
Arithmetic Operators in Embedded MATLAB Functions . . **12-5**
Relational Operators in Embedded MATLAB Functions . . **12-6**
Logical Operators in Embedded MATLAB Functions **12-6**

Embedded MATLAB Run-Time Function Library **12-8**
Embedded MATLAB Run-Time Function Library – Alphabetical List **12-8**
Embedded MATLAB Run-Time Library – Categorical List **12-20**

Unsupported MATLAB Features and Limitations **12-32**
List of Unsupported Features **12-32**
Limitations on Indexing Operations **12-33**
Limitations with Complex Numbers **12-34**

Index

Blocks — Categorical List

What Each Block Reference Page Contains (p. 1-2)	Describes the structure and content of block reference pages.
Blocks by Category (p. 1-4)	Blocks listed according to the category (library) to which they belong.

What Each Block Reference Page Contains

Blocks appear in alphabetical order and contain some or all of this information:

- The block name, icon, and block library that contains the block
- The purpose of the block
- A description of the block's use
- The data types and numeric type (complex or real) accepted and generated by the block
- The block dialog box and parameters
- The rules for some or all of these topics, as they apply to blocks with fixed-point capabilities:
  - Converting block parameters from double-precision numbers to fixed-point data types
  - Converting the input data type(s) to the output data type
  - Performing block operations between inputs and parameters
- The block characteristics, including some or all of these, as they apply to the block:
  - Direct Feedthrough - Whether the block or any of its ports has direct feedthrough. For more information, see "Algebraic Loops".
  - Sample Time - How the block's sample time is determined, whether by the block itself (as is the case with discrete and continuous blocks) or inherited from the block that drives it or is driven by it. For more information, see "Sample Time Propagation".
  - Scalar Expansion - Whether or not scalar values are expanded to arrays. Some blocks expand scalar inputs and/or parameters as appropriate. For more information, see "Scalar Expansion of Inputs and Parameters".
  - States - The number of discrete and continuous states.
  - Dimensionalized - Whether the block accepts and/or generates multidimensional signal arrays. For more information, see "Signal Basics".

- Zero Crossings - Whether the block detects zero-crossing events. For more information, see “Zero Crossing Detection”.

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

```
showblockdatatypetable
```

Blocks by Category

- "Commonly Used" on page 1-4
- "Continuous" on page 1-5
- "Discontinuities" on page 1-5
- "Discrete" on page 1-6
- "Logic and Bit Operations" on page 1-6
- "Lookup Tables" on page 1-7
- "Math Operations" on page 1-7
- "Model Verification" on page 1-8
- "Model-Wide Utilities" on page 1-8
- "Ports & Subsystems" on page 1-8
- "Signal Attributes" on page 1-9
- "Signal Routing" on page 1-9
- "Sinks" on page 1-10
- "Sources" on page 1-10
- "User-Defined Functions" on page 1-10
- "Additional Math & Discrete" on page 1-11

Commonly Used

```
Bus Creator

Bus Selector
Constant
Data Type Conversion
Demux
Discrete-Time Integrator
Gain
Ground
Inport
Integrator
Logical Operator
```

Mux
Outport
Product
Relational Operator
Saturation
Scope, Floating Scope, Signal Viewer Scope
Subsystem, Atomic Subsystem, CodeReuse Subsystem
Sum
Switch
Terminator
Unit Delay

Continuous

Derivative
Integrator
State-Space
Transfer Fcn
Transport Delay
Variable Time Delay, Variable Transport Delay
Zero-Pole

Discontinuities

Backlash
Coulomb and Viscous Friction
Dead Zone
Dead Zone Dynamic
Hit Crossing
Quantizer
Rate Limiter
Rate Limiter Dynamic
Relay
Saturation
Saturation Dynamic
Wrap To Zero

Discrete

Difference
Discrete Derivative
Discrete Filter
Discrete State-Space
Discrete Transfer Fcn
Discrete Zero-Pole
Discrete-Time Integrator
First-Order Hold
Integer Delay
Memory
Tapped Delay
Transfer Fcn First Order
Transfer Fcn Lead or Lag
Transfer Fcn Real Zero
Unit Delay
Weighted Moving Average
Zero-Order Hold

Logic and Bit Operations

Bit Clear
Bitwise Operator
Combinatorial Logic
Compare To Constant
Compare To Zero
Detect Change
Detect Decrease
Detect Fall Negative
Detect Fall Nonpositive
Detect Increase
Detect Rise Nonnegative
Detect Rise Positive
Extract Bits
Interval Test
Interval Test Dynamic
Logical Operator
Relational Operator
Shift Arithmetic

Lookup Tables

Cosine
Direct Lookup Table (n-D)
Interpolation (n-D) Using PreLookup
Lookup Table
Lookup Table (2-D)
Lookup Table (n-D)
Lookup Table Dynamic
Prelookup Index Search
Sine

Math Operations

Abs
Add
Algebraic Constraint
Assignment
Bias
Complex to Magnitude-Angle
Complex to Real-Imag
Divide
Dot Product
Gain
Magnitude-Angle to Complex
Math Function
Matrix Concatenation
MinMax
MinMax Running Resettable
Polynomial
Product
Real-Imag to Complex
Reshape
Rounding Function
Sign
Sine Wave Function
Slider Gain
Subtract
Sum
Sum of Elements
Trigonometric Function

Unary Minus
Weighted Sample Time Math

Model Verification

Assertion
Check Discrete Gradient
Check Dynamic Gap
Check Dynamic Lower Bound
Check Dynamic Range
Check Dynamic Upper Bound
Check Input Resolution
Check Static Gap
Check Static Lower Bound
Check Static Range
Check Static Upper Bound

Model-Wide Utilities

DocBlock
Model Info
Time-Based Linearization
Trigger-Based Linearization

Ports & Subsystems

Configurable Subsystem
Enable
Enabled and Triggered Subsystem
Enabled Subsystem
For Iterator Subsystem
Function-Call Generator
Function-Call Subsystem

If Action Subsystem
Inport
Model
Outport
Subtract
Switch Case

Switch Case Action Subsystem
Trigger
Triggered Subsystem
While Iterator Subsystem

Signal Attributes

Data Type Conversion
Data Type Conversion Inherited
Data Type Duplicate
Data Type Propagation
Data Type Scaling Strip
IC
Probe
Rate Transition
Signal Conversion
Signal Specification
Weighted Sample Time
Width

Signal Routing

Bus Assignment
Bus Creator
Bus Selector
Data Store Memory
Data Store Read
Data Store Write
Demux
Environment Controller
From
Goto
Goto Tag Visibility
Index Vector
Manual Switch
Merge
Multiport Switch
Mux
Selector
Switch

Sinks

Display
Outport
Scope, Floating Scope, Signal Viewer Scope
Stop Simulation
Terminator
To File
To Workspace
XY Graph

Sources

Band-Limited White Noise
Chirp Signal
Clock
Constant
Counter Free-Running
Counter Limited
Digital Clock
From File
From Workspace
Ground
Inport
Pulse Generator
Ramp
Random Number
Repeating Sequence
Repeating Sequence Interpolated
Repeating Sequence Stair
Signal Builder
Signal Generator
Sine Wave
Step
Uniform Random Number

User-Defined Functions

Embedded MATLAB Function
Fcn
Level-2 M-File S-Function

MATLAB Fcn
MinMax
S-Function
S-Function Builder

Additional Math & Discrete

Additional Discrete

Fixed-Point State-Space
Transfer Fcn Direct Form II
Transfer Fcn Direct Form II Time Varying
Unit Delay Enabled
Unit Delay Enabled External IC
Unit Delay Enabled Resettable
Unit Delay Enabled Resettable External IC
Unit Delay External IC
Unit Delay Resettable
Unit Delay Resettable External IC
Unit Delay With Preview Enabled
Unit Delay With Preview Enabled Resettable
Unit Delay With Preview Enabled Resettable External RV
Unit Delay With Preview Resettable
Unit Delay With Preview Resettable External RV

Additional Math

Decrement Real World
Decrement Stored Integer
Decrement Time To Zero
Decrement To Zero
Increment Real World
Increment Stored Integer

2

Blocks — Alphabetical List

This chapter is a reference for the blocks in Simulink. Blocks are listed alphabetically.

Abs
Action Port
Add
Algebraic Constraint
Assertion
Assignment
Backlash
Band-Limited White Noise
Bias
Bit Clear
Bit Set
Bitwise Operator
Bus Assignment
Bus Creator
Bus Selector
Check Discrete Gradient
Check Dynamic Gap
Check Dynamic Lower Bound
Check Dynamic Range
Check Dynamic Upper Bound
Check Input Resolution
Check Static Gap
Check Static Lower Bound
Check Static Range
Check Static Upper Bound
Chirp Signal

Clock
Combinatorial Logic
Compare To Constant
Compare To Zero
Complex to Magnitude-Angle
Complex to Real-Imag
Configurable Subsystem
Constant
Cosine
Coulomb and Viscous Friction
Counter Free-Running
Counter Limited
Data Store Memory
Data Store Read
Data Store Write
Data Type Conversion
Data Type Conversion Inherited
Data Type Duplicate
Data Type Propagation
Data Type Scaling Strip
Dead Zone
Dead Zone Dynamic
Decrement Real World
Decrement Stored Integer
Decrement Time To Zero
Decrement To Zero
Demux
Derivative
Detect Change
Detect Decrease
Detect Fall Negative
Detect Fall Nonpositive
Detect Increase
Detect Rise Nonnegative
Detect Rise Positive
Difference
Digital Clock

Direct Lookup Table (n-D)
Discrete Derivative
Discrete Filter
Discrete State-Space
Discrete Transfer Fcn
Discrete Zero-Pole
Discrete-Time Integrator
Display
Divide
DocBlock
Dot Product
Embedded MATLAB Function
Enable
Enabled and Triggered Subsystem
Enabled Subsystem
Environment Controller
Extract Bits
Fcn
First-Order Hold
Fixed-Point State-Space
For Iterator
For Iterator Subsystem
From
From File
From Workspace
Function-Call Generator
Function-Call Subsystem
Gain
Goto
Goto Tag Visibility
Ground
Hit Crossing
IC
If
If Action Subsystem
Increment Real World
Increment Stored Integer

Index Vector
Inport
Integer Delay
Integrator
Interpolation (n-D) Using PreLookup
Interval Test
Interval Test Dynamic
Level-2 M-File S-Function
Logical Operator
Lookup Table
Lookup Table (2-D)
Lookup Table (n-D)
Lookup Table Dynamic
Magnitude-Angle to Complex
Manual Switch
Math Function
MATLAB Fcn
Matrix Concatenation
Memory
Merge
MinMax
MinMax Running Resettable
Model
Model Info
Multiport Switch
Mux
Outport
Polynomial
Prelookup Index Search
Probe
Product
Product of Elements
Pulse Generator
Quantizer
Ramp
Random Number
Rate Limiter

Rate Limiter Dynamic
Rate Transition
Real-Imag to Complex
Relational Operator
Relay
Repeating Sequence
Repeating Sequence Interpolated
Repeating Sequence Stair
Reshape
Rounding Function
S-Function
S-Function Builder
Saturation
Saturation Dynamic
Scope, Floating Scope, Signal Viewer Scope
Selector
Shift Arithmetic
Sign
Signal Builder
Signal Conversion
Signal Generator
Signal Specification
Sine
Sine and Cosine
Sine Wave
Sine Wave Function
Slider Gain
State-Space
Step
Stop Simulation
Subsystem, Atomic Subsystem, CodeReuse Subsystem
Subtract
Sum
Sum of Elements
Switch
Switch Case
Switch Case Action Subsystem

Tapped Delay
Terminator
Time-Based Linearization
To File
To Workspace
Transfer Fcn
Transfer Fcn Direct Form II
Transfer Fcn Direct Form II Time Varying
Transfer Fcn First Order
Transfer Fcn Lead or Lag
Transfer Fcn Real Zero
Transport Delay
Trigger
Trigger-Based Linearization
Triggered Subsystem
Trigonometric Function
Unary Minus
Uniform Random Number
Unit Delay
Unit Delay Enabled
Unit Delay Enabled External IC
Unit Delay Enabled Resettable
Unit Delay Enabled Resettable External IC
Unit Delay External IC
Unit Delay Resettable
Unit Delay Resettable External IC
Unit Delay With Preview Enabled
Unit Delay With Preview Enabled Resettable
Unit Delay With Preview Enabled Resettable External RV
Unit Delay With Preview Resettable
Unit Delay With Preview Resettable External RV
Variable Time Delay, Variable Transport Delay
Weighted Moving Average
Weighted Sample Time
Weighted Sample Time Math
While Iterator
While Iterator Subsystem

Width
Wrap To Zero
XY Graph
Zero-Order Hold
Zero-Pole

Abs

Purpose

Output the absolute value of the input

Library

Math Operations

Description

The Abs block outputs the absolute value of the input.

For signed data types, the absolute value of the most negative value is problematic since it is not representable by the data type. In this case, the behavior of the block is controlled by the **Saturate on integer overflow** check box. If checked, the absolute value of the data type saturates to the most positive representable value. If not checked, the absolute value of the most negative value represented by the data type has no effect.

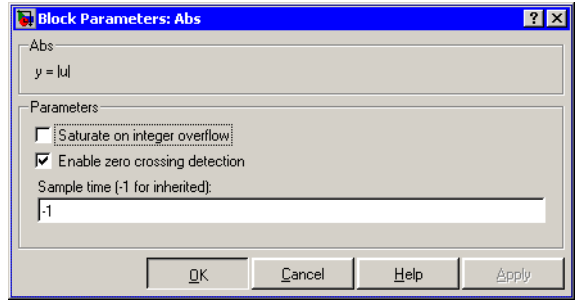
For example, suppose the block input is an 8-bit signed integer. The range of this data type is from -128 to 127, and the absolute value of -128 is not representable. If you select the **Saturate on integer overflow** check box, then the absolute value of -128 is 127. If it is not selected, then the absolute value of -128 remains at -128.

Data Type Support

The Abs block accepts real signals of any data type supported by Simulink, except `Boolean`. The Abs block supports real fixed-point data types. The block also accepts complex `single` and `double` inputs. Outputs are a real value of the same data type as the input.

For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in Using Simulink.

Parameters and Dialog Box



Saturate on integer overflow

When selected, the block maps signed integer input elements corresponding to the most negative value of that data type to the most positive value of that data type:

- For 8-bit integers, -128 maps to to 127.
- For 16-bit integers, -32768 maps to 32767.
- For 32-bit integers, -2147483648 maps to 2147483647.

When not selected, the block does not act on signed integer input elements corresponding to the most negative value of that data type.

- For 8-bit integers, -128 remains -128.
- For 16-bit integers, -32768 remains -32768.
- For 32-bit integers, -2147483648 remains -2147483648.

Enable zero crossing detection

Select to enable zero crossing detection. For more information, see "Zero-Crossing Detection" in Using Simulink.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in Using Simulink for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Dimensionalized	Yes
Zero Crossing	Yes, if enabled

Action Port

Purpose

Implement the Action subsystems used by `if` and `switch` control flow statements in Simulink

Library

Ports & Subsystems

Description

Action Port blocks implement Action subsystems used in `if` and `switch` control flow statements. See the references for the If and Switch Case blocks for examples using Action Port blocks.

Use Action Port blocks to create Action subsystems as follows:

1. Place a subsystem in the system containing the If or Switch Case block.

   You can use an ordinary subsystem or an atomic subsystem. In either case, the resulting Action subsystem is atomic.

2. Add an Action Port to the new subsystem.

   This adds an input port named Action to the subsystem, which is now an Action subsystem.

Action subsystems execute their programming in response to the conditional outputs of an If or Switch Case block. Use Action subsystems as follows:

1. Create an Action subsystem for each output port configured for an If or Switch Case block.

2. Connect each output port (if, else, or elseif ports for the If block; case or default ports for the Switch Case block) to the Action port on an Action subsystem.

   When the connection is made, the icon for the subsystem and the Action Port block it contains are changed to the name of the output port for the If or Switch Case block (i.e., `if{ }`, `else{ }`, `elseif{ }`, `case{ }`, or `default{ }`).

3 Open the new subsystem and add the diagram that you want to execute in response to the condition this subsystem covers.

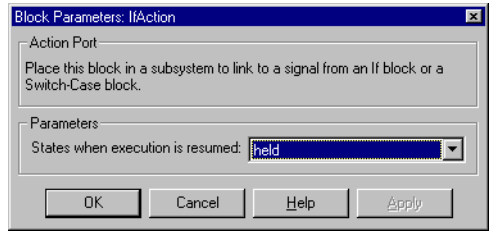
The Action Port block has only the **States when execution is resumed** parameter in its parameters dialog. If you set this field to `held` (the default value) for an Action Port block, the states of its Action subsystem are retained between calls even if other member Action subsystems of an `if-else` or `switch` control flow statement are called. If you set the **States when execution is resumed** field to `reset`, the states of a member Action subsystem are reset to initial values when it is reenabled.

Note All blocks in an Action subsystem driven by an If or Switch Case block must run at the same rate as the driving block.

Data Type Support

There are no data inputs or outputs for Action Port blocks.

Parameters and Dialog Box



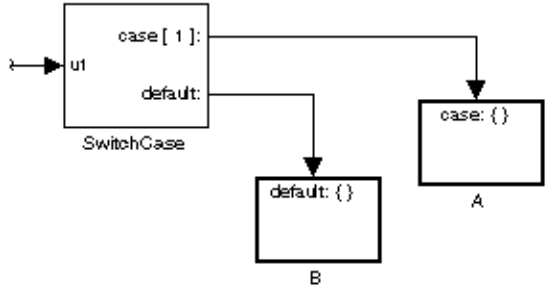
States when execution is resumed

Specifies how to handle internal states when the subsystem of this Action Port block is reenabled.

Set this field to `held` (the default value) to make sure that the Action subsystem states retain their previous values when the subsystem is reenabled. Otherwise, set this field to `reset` if you

want the states of the Action subsystem to be reinitialized when the subsystem is reenabled.

Reenablement of a subsystem occurs when it is called and the condition of the call is true after having been previously false. In the following example, the Action Port blocks for both Action subsystems A and B have the **States when execution is resumed** parameter set to `reset`.



If case[1] is true, Action subsystem A is called. This implies that the default condition is false. When B is later called for the default condition, its states are reset. In the same way, Action subsystem A's states are reset when it is called right after Action subsystem B is called.

Repeated calls to a case's Action subsystem do not reset its states. If A is called again right after a previous call to A, this does not reset A's states because its condition, case[1], was not previously false. The same applies to B.

Characteristics

Sample Time	Inherited from driving If or Switch Case block

Purpose

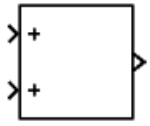
Add or subtract inputs

Library

Math Operations

Description

The Add block is an implementation of the Sum block. See `Sum` for more information.



Algebraic Constraint

Purpose

Constrain the input signal to zero

Library

Math Operations

Description

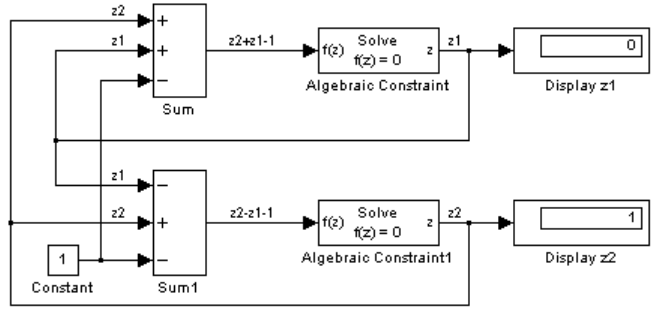
The Algebraic Constraint block constrains the input signal f(*z*) to zero and outputs an algebraic state *z*. The block outputs the value necessary to produce a zero at the input. The output must affect the input through some feedback path. This enables you to specify algebraic equations for index 1 differential/algebraic systems (DAEs).

By default, the **Initial guess** parameter is zero. You can improve the efficiency of the algebraic loop solver by providing an **Initial guess** for the algebraic state *z* that is close to the solution value.

For example, the following model solves these equations.

```
z2 + z1 = 1
z2 - z1 = 1
```

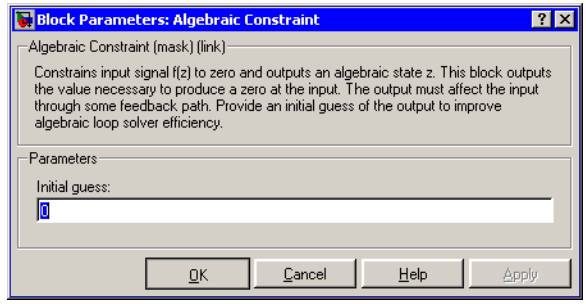
The solution is `z2 = 1`, `z1 = 0`, as the Display blocks show.



Data Type Support

The Algebraic Constraint block accepts and outputs real values of type `double`.

Parameters and Dialog Box



Initial guess

An initial guess for the solution value. The default is `0`.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	Yes
Zero-Crossing	No

Assertion

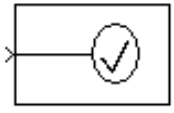
Purpose

Check whether a signal is nonzero

Library

Model Verification

Description



The Assertion block checks whether any of the elements of the signal at its input is nonzero. If any element is nonzero, the block does nothing. If any element is zero, the block halts the simulation, by default, and displays an error message. The block's parameter dialog box allows you to

- Specify that the block should display an error message when the assertion fails but allow the simulation to continue.
- Specify an M-expression to be evaluated when the assertion fails.
- Enable or disable the assertion.

You can also use the **Model Verification block enabling** setting on the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box to enable or disable all Assertion blocks in a model.

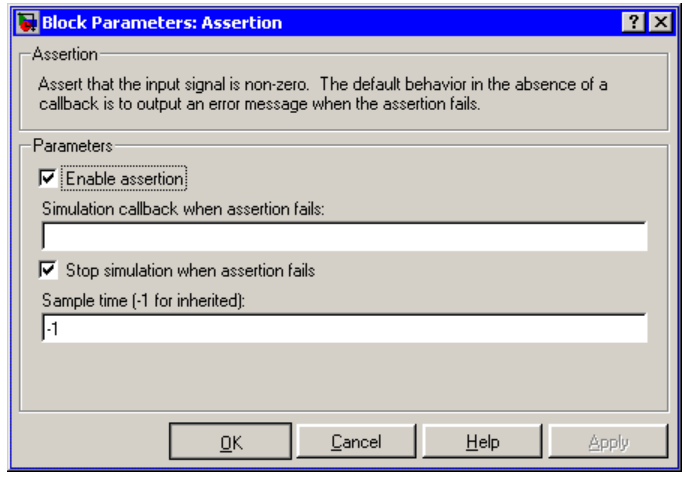
The Assertion block and its companion blocks in the Model Verification library are intended to facilitate creation of self-validating models. For example, you can use model verification blocks to test that signals do not exceed specified limits during simulation. When you are satisfied that a model is correct, you can turn error checking off by disabling the verification blocks. You do not have to physically remove them from the model. If you need to modify a model, you can temporarily turn the verification blocks back on to ensure that your changes do not break the model.

Data Type Support

The Assertion block accepts input signals of any dimensions and any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see "Data Types Supported by Simulink" in Using Simulink.

Parameters and Dialog Box



Enable Assertion

Unchecking this option disables the Assertion block, that is, causes the model to behave as if the Assertion block did not exist. The **Model Verification block enabling** setting on the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box allows you to enable or disable all Assertion blocks in a model regardless of the setting of this option.

Simulation callback when assertion fails

An M-expression to be evaluated when the assertion fails.

Stop simulation when assertion fails

If checked, this option causes the Assertion block to halt the simulation when the block's input is zero and display an error message in the **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB Command Window and continues the simulation.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in Using Simulink for more information.

Characteristics

Direct Feedthrough	No
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

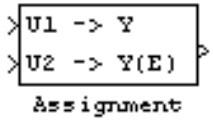
Purpose

Assign values to specified elements of a signal

Library

Math Operations

Description



The Assignment block assigns values to specified elements of the signal. You can specify the indices of the elements to be assigned values either by entering the indices in the block's dialog box or by connecting an external indices source or sources to the block. The signal at the block's data port, labeled U2 in most modes, specifies values to be assigned to Y. The block replaces the specified elements of Y with elements from the data signal, leaving unassigned elements unchanged from their initial values. If the assignment indices source is internal or is external and the `Initialize using input` option is selected, the Assignment block uses the signal at the block's initialization port, labeled U1, to initialize the elements of the output signal before assigning them values from U2.

Note The Assignment block's data port is labeled U2 in all modes except external mode with no initialization, in which case the port is labeled U1 as there is no initialization port. The rest of this section refers to the data port as U2 in order to avoid unnecessarily complicating the explanation of the block's usage.

You can use the block to assign values to vector, or matrix signals.

Assigning Values to a Vector Signal

To assign values to a scalar or vector signal, set the block's **Input Type** parameter to `Vector`. The block's dialog box displays a **Source of element indices** parameter. You can specify the indices source as `Internal` or `External`. If you select `Internal`, the block dialog box displays an **Elements** field. Use this field to enter the element indices. If you specify `External` as the source of element indices, the block displays an input port named E. Connect an external index source to this port. Use **Index mode** to specify whether `0` or `1` indicates the first element of Y.

The index source can specify any of the following values as indices:

- -1 (internal source only)

  Assigns every element of U2 to the corresponding element of Y.

- Index of a single element specified as a nonnegative integer

  If **Use index as starting value** option is not selected, the block assigns U2, which must be a scalar, to the specified element of Y.

  If **Use index as starting value** is selected, the block assigns U2 to a range of elements of Y, starting at the specified index. For example, suppose that U1 is a 5-element vector, U2 is a 3-element vector, the index mode is one-based, and the starting index is 3. In this case, the Assignment block assigns U2(1:3) to Y(3:5).

- A set of indices specified as a vector

  Assigns U2 to a specified set of elements of Y.

  The width of the values signal connected to U2 must be the same as the width of the indices vector. For example, if the indices vector contains two indices, U2 must be a two-element vector of values. The block assigns the first element of U2 to the element of Y specified by the first index, the second element of U2 to the Y element specified by the second index, and so on.

  If U2 is a scalar, it is assigned to the specified elements of the output vector.

Assigning Values to a Matrix Signal

To assign values to a matrix signal, set the **Input Type** parameter to `Matrix`. If you specify the **Input Type** of the Assignment block as `Matrix`, the block's dialog box displays a **Source of row indices** parameter and a **Source of column indices** parameter. You can specify either or both of these parameters as `Internal` or `External`. If you specify the row and/or column index source as internal, the block displays a **Rows** and/or **Columns** field. Enter the row or column indices of the elements of Y to be assigned values into the corresponding

field. If you specify the row and/or column index source as External, the block displays an input port labeled R and/or an input port labeled C. Connect an external source of indices to each indices port.

A row or column indices source can have any of the following values:

- -1 (internal source only)

  Specifies all rows or columns of Y.

- Single row or column index value

  If **Use index as starting value** option is not selected, the block assigns values to the specified row or column. If **Use index as starting value** is selected, the block assigns values from U2 to a range of rows or columns of Y, starting at the specified row or column index. For example, suppose that U1 is a 5 x 5 matrix, U2 is a 3 x 3 matrix, the indexing mode is one-based, and the starting row and column indices are both 3. In this case, the Assignment block assigns U2(1:3, 1:3) to Y(3:5,3:5).

- Vector of row or column indices

  Specifies a set of rows or columns of Y.

  The block assigns values from U2 to the specified elements of Y in column-major order. In particular, the block assigns the first element of the first row of U2 to the first specified element in the first specified row in Y. It assigns the second element of the first row of U2 to the second specified element of the first specified row of Y, and so on.

  To enable all specified elements to be assigned values, U2 must be an `N`-by-`M` matrix where `N` is the width of the row indices vector and `M` is the width of the column indices vector. For example, suppose that you specify a vector of row indices of size 2 and a vector of column indices of size 4. Then U2 must be a 2-by-4 matrix signal.

  When determining the dimensions of U2, count a scalar index as a vector of size 1 and -1 as equivalent to a vector of indices of the same width as the row or dimension size of Y. For example, suppose your

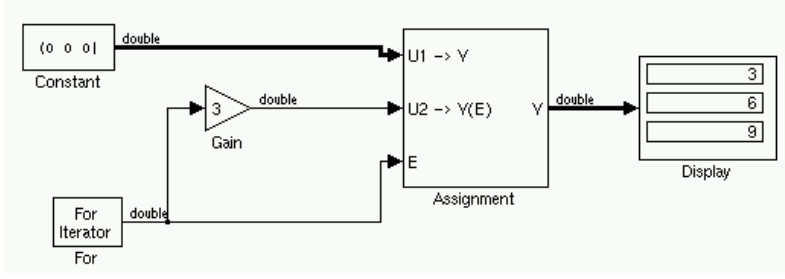
row and column index sources are a scalar and a two-element vector, respectively. Then U2 must by a 1-by-2 matrix.

If U2 is a scalar, the Assignment block assigns the scalar to the specified elements of the output signal.

Note An Assignment block whose **Input type** is `Matrix` accepts only matrix signals at its U1 port and only a matrix signal or a one-element vector signal at its U2 port. Simulink displays an error dialog box if you update or simulate a model that violates this constraint.

Iterated Assignment

You can use the Assignment block to assign values computed in a For or While Iterator loop to successive elements of a vector or matrix signal in a single time step. For example, the following model uses a For Iterator block to create a vector signal each of whose elements equals 3*i where i is the index of the element.



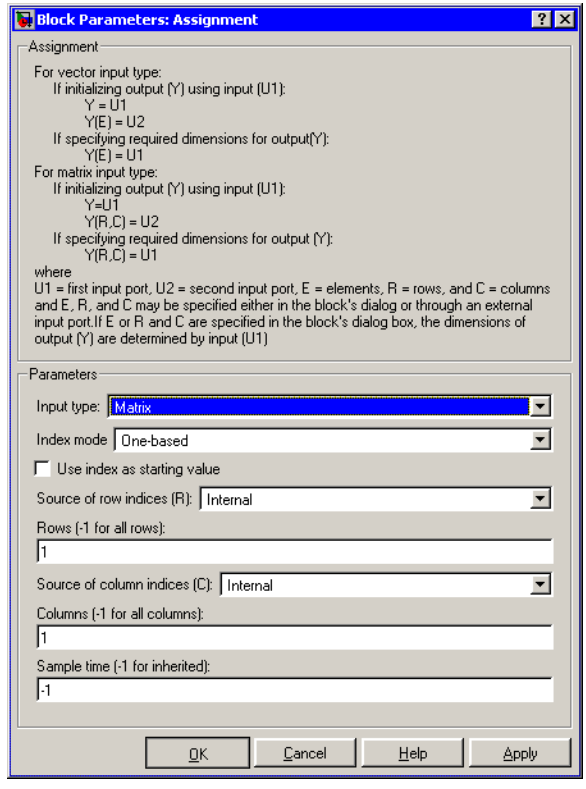
Iterated assignment uses an iterator (For or While) block to generate the indices required by the Assignment block. On the first iteration of an iterated assignment, the Assignment block copies the first input (U1) to the output (Y) and assigns the second input (U2) to the output $Y(E_0)$. On successive iterations, the Assignment block simply assigns the current value of U2 to $Y(E_i)$, i.e., without first copying U1 to Y. All of this occurs in a single time step.

Data Type Support

The data and initialization ports of the Assignment block accept signals of any data type supported by Simulink, including fixed-point data types. The external indices port accepts any data type, except boolean and fixed-point data types.

For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in Using Simulink.

Parameters and Dialog Box



Input Type

You can select either `Vector` or `Matrix` input. If you select `Vector`, the **Source of element indices** field appears. If you select

`Matrix`, the **Source of row indices** and **Source of column indices** fields appear.

Index mode
Specifies whether the index corresponding to the first element of a vector or the first row or column of a matrix is `0` or `1`.

Use index as starting value
Specifies that the value in the Elements (or Row or Column) field is the starting index of a range of elements (or rows or columns).

Source of element indices
You can specify either `Internal` (the default) or `External` as the source for the indices of the elements to be assigned values. If you select Internal, the block dialog box displays an **Elements** field (see following). Use this field to enter the element indices. If you select `External`, the block displays an input port labeled E. Connect the external index source to this port.

Elements
This field appears only if you selected `Internal` for the **Source of element indices** field. It specifies the indices of elements in Y to be assigned values from elements in U2. The value of this parameter can be `-1`, a nonnegative integer specifying a single index, or a vector of nonnegative integers specifying a set of indices (e.g., `[1,3,5,6]`).

Source of row indices
Either `Internal` (the default) or `External`. If you select `Internal`, the **Rows** field appears. Enter the indices of the rows to be assigned values in this field. If you select `External`, the block displays an input port labeled R. Connect an external source of row indices to this port.

Rows
This field appears only if you select `Internal` for the **Source of row indices** field. Valid values are `-1` (all rows), a single row index, or a vector of row indices (e.g., `[1,3,5,6]`).

Source of column indices

Either `Internal` (the default) or `External`. If you select `Internal`, the **Columns** field appears. Enter the indices of the columns to be assigned values in this field. If you select `External`, the block displays an input port labeled C. Connect an external source of column indices to this port.

Columns

This field appears only if you selected `internal` for the **Source of column indices** field. Valid values are `-1` (all columns), a single column index, or a vector of column indices (e.g., `[1,3,5,6]`).

Output (Y)

This control appears only if the source of assignment indices is external or, in the case of matrix assignment, the source of either the row or column indices, or both, is external. The options are `Initialize using input (U1)` (the default) or `Specify required dimensions`. The first option causes the Assignment block to display an initialization port labeled U1 and to use the signal at this port to initialize the output signal (Y) before assigning it values from the data port (U2) as specified by the external indices signal (E). The second option does not initialize Y before assigning values from the block's data input port (labeled U1 in this case) to it. This option requires that the block's U1 and E inputs assign values to every element of Y. Further, it requires that you specify the dimensions of the output signal (see next control).

Output dimensions

This control appears only if you specify the `Specify required dimensions` option of the **Output (Y)** control. It specifies the dimensions of the Assignment block's output signal.

Diagnostic if not all required dimensions are populated

This control appears only if you specify the `Specify required dimensions` option of the **Output (Y)** control. It specifies the diagnostic action that Simulink should take if the block's data (U1) and external indices (E) inputs do not assign a value to every element of the block's output (Y). The options are to display an

error message and halt the simulation (`Error`), display a warning message (`Warning`) and continue the simulation, or continue the simulation (`None`). If you choose `Warning` or `None`, the values of the unassigned elements of the output are undefined.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in Using Simulink for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes
Dimensionalized	Yes
Zero Crossing	No

Purpose

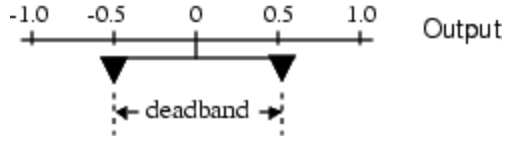
Model the behavior of a system with play

Library

Discontinuities

Description

The Backlash block implements a system in which a change in input causes an equal change in output. However, when the input changes direction, an initial change in input has no effect on the output. The amount of side-to-side play in the system is referred to as the *deadband*. The deadband is centered about the output. This figure shows the block's initial state, with the default deadband width of 1 and initial output of 0.



A system with play can be in one of three modes:

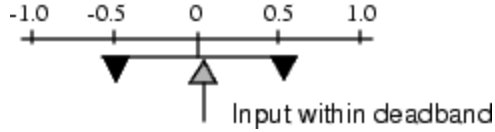
- Disengaged - In this mode, the input does not drive the output and the output remains constant.
- Engaged in a positive direction - In this mode, the input is increasing (has a positive slope) and the output is equal to the input *minus* half the deadband width.
- Engaged in a negative direction - In this mode, the input is decreasing (has a negative slope) and the output is equal to the input *plus* half the deadband width.

If the initial input is outside the deadband, the **Initial output** parameter value determines whether the block is engaged in a positive or negative direction, and the output at the start of the simulation is the input plus or minus half the deadband width.

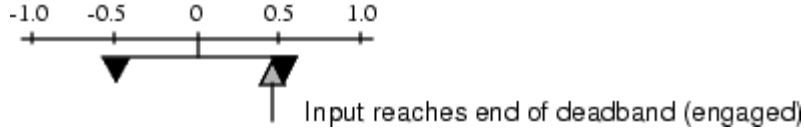
For example, the Backlash block can be used to model the meshing of two gears. The input and output are both shafts with a gear on one end, and the output shaft is driven by the input shaft. Extra space

between the gear teeth introduces *play*. The width of this spacing is the **Deadband width** parameter. If the system is disengaged initially, the output (the position of the driven gear) is defined by the **Initial output** parameter.

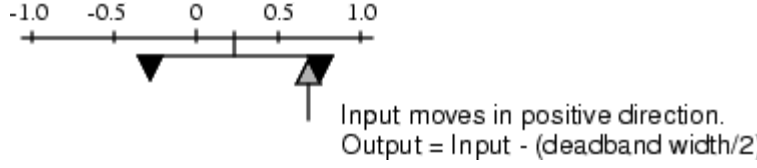
The following figures illustrate the block's operation when the initial input is within the deadband. The first figure shows the relationship between the input and the output while the system is in disengaged mode (and the default parameter values are not changed).



The next figure shows the state of the block when the input has reached the end of the deadband and engaged the output. The output remains at its previous value.



The final figure shows how a change in input affects the output while they are engaged.

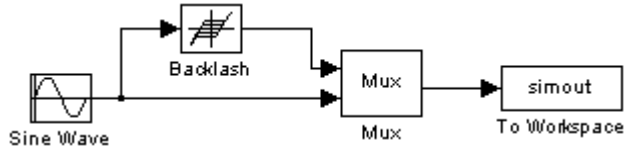


If the input reverses its direction, it disengages from the output. The output remains constant until the input either reaches the opposite end of the deadband or reverses its direction again and engages at the same end of the deadband. Now, as before, movement in the input causes equal movement in the output.

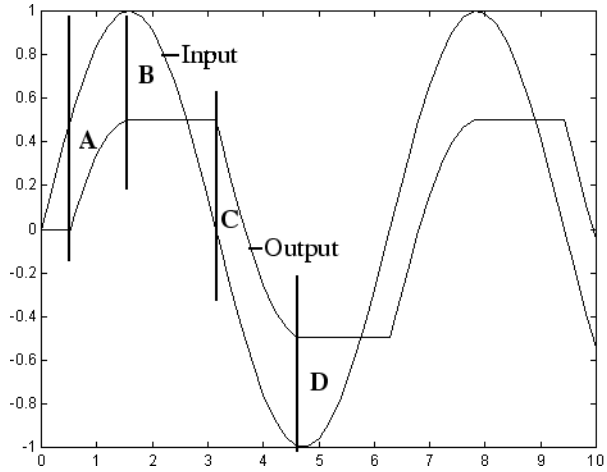
For example, if the deadband width is 2 and the initial output is 5, the output, y, at the start of the simulation is as follows:

- 5 if the input, u, is between 4 and 6
- $u + 1$ if $u < 4$
- $u - 1$ if $u > 6$

This sample model and the plot that follows it show the effect of a sine wave passing through a Backlash block.



The Backlash block parameters are unchanged from their default values (the deadband width is 1 and the initial output is 0). Notice in the plotted output following that the Backlash block output is zero until the input reaches the end of the deadband (at 0.5). Now the input and output are engaged and the output moves as the input does until the input changes direction (at 1.0). When the input reaches 0, it again engages the output at the opposite end of the deadband.



A Input engages in positive direction. Change in input causes equal change in output.

B Input disengages. Change in input does not affect output.

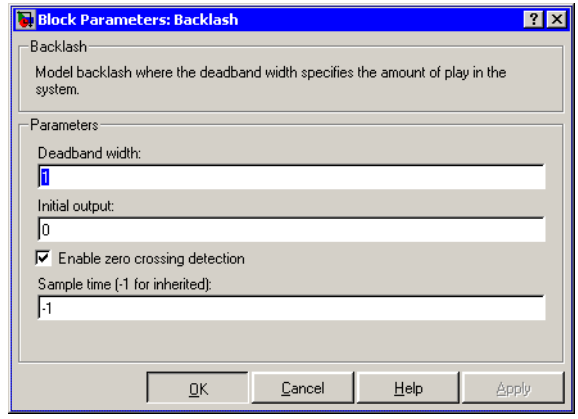
C Input engages in negative direction. Change in input causes equal change in output.

D Input disengages. Change in input does not affect output.

Data Type Support

The Backlash block accepts and outputs real values of `single`, `double`, and built-in integer data types.

Parameters and Dialog Box



Deadband width

Specify the width of the deadband. The default is `1`.

Initial output

Specify the initial output value. The default value is `0`. This parameter is tunable. Simulink does not allow the initial output of this block to be `inf` or `NaN`.

Enable zero-crossing detection

Select to enable use of zero-crossing detection to detect engagement with lower and upper thresholds. For more information, see "Zero-Crossing Detection" in Using Simulink.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in Using Simulink for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes
Dimensionalized	Yes
Zero Crossing	Yes, if you select **Enable zero crossing detection**

Band-Limited White Noise

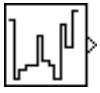
Purpose

Introduce white noise into a continuous system

Library

Sources

Description



The Band-Limited White Noise block generates normally distributed random numbers that are suitable for use in continuous or hybrid systems.

The primary difference between this block and the Random Number block is that the Band-Limited White Noise block produces output at a specific sample rate, which is related to the correlation time of the noise.

Theoretically, continuous white noise has a correlation time of 0, a flat power spectral density (PSD), and a covariance of infinity. In practice, physical systems are never disturbed by white noise, although white noise is a useful theoretical approximation when the noise disturbance has a correlation time that is very small relative to the natural bandwidth of the system.

In Simulink, you can simulate the effect of white noise by using a random sequence with a correlation time much smaller than the shortest time constant of the system. The Band-Limited White Noise block produces such a sequence. The correlation time of the noise is the sample rate of the block. For accurate simulations, use a correlation time much smaller than the fastest dynamics of the system. You can get good results by specifying

$$t_c \approx \frac{1}{100}\frac{2\pi}{f_{max}}$$

where $f_{\max}$ is the bandwidth of the system in rad/sec.

The Algorithm Used in the Block Implementation

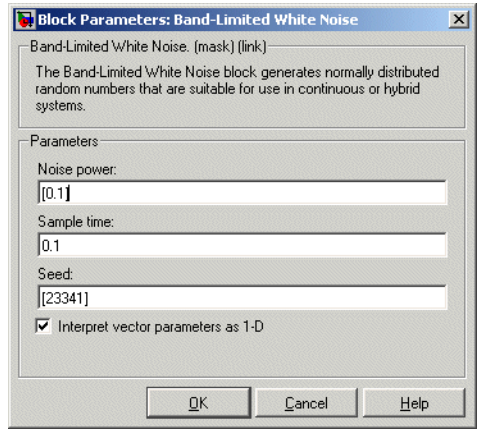
To produce the correct intensity of this noise, the covariance of the noise is scaled to reflect the implicit conversion from a continuous PSD to a discrete noise covariance. The appropriate scale factor is *1*/*tc*, where *tc* is the correlation time of the noise. This scaling ensures that the response of a continuous system to the approximate white noise has the same covariance as the system would have to true white noise. Because

of this scaling, the covariance of the signal from the Band-Limited White Noise block is not the same as the **Noise power** (intensity) dialog box parameter. This parameter is actually the height of the PSD of the white noise. While the covariance of true white noise is infinite, the approximation used in this block has the property that the covariance of the block output is the **Noise Power** divided by *tc*.

Data Type Support

The Band-Limited White Noise block outputs real values of type `double`.

Parameters and Dialog Box



Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in Using Simulink.

Noise power

The height of the PSD of the white noise. The default value is `0.1`.

Sample time

The correlation time of the noise. The default value is `0.1`. See "Specifying Sample Time" in Using Simulink for more information.

Seed

The starting seed for the random number generator. The default value is `23341`.

Interpret vector parameters as 1-D

Output a 1-D array if the block's parameters are vectors. Otherwise, output a 2-D array one of whose dimensions is 1.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of **Noise power** and **Seed** parameters and output
Dimensionalized	Yes
Zero Crossing	No

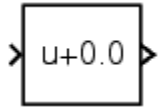
Purpose

Add a bias to the input

Library

Math Operations

Description



The Bias block adds a bias, or offset, to the input signal according to

$$Y = U + Bias$$

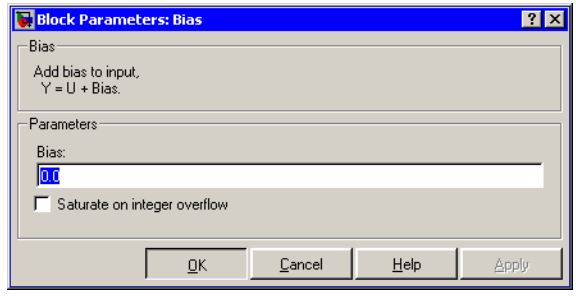
where U is the block input and Y is the output.

Data Type Support

The Bias block accepts and outputs real or complex values of any data type supported by Simulink, except `Boolean`. The Bias block supports fixed-point data types.

For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in Using Simulink.

Parameters and Dialog Box



Bias

Specify the value of the offset to add to the input signal.

Saturate on integer overflow

If the input (and hence the output) is an integer data type (for example, `int8`) and the data type cannot accommodate the

output signal, selecting this option causes the block to output the maximum value allowed by the data type. Otherwise, in this case, the block outputs the result of using twos-complement arithmetic to add the input to the output, i.e., the value is the result of adding the bias to the input modulo the maximum representable value of the data type.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from the driving block
Scalar Expansion	Yes
States	0
Dimensionalized	Yes
Zero Crossing	No

Purpose

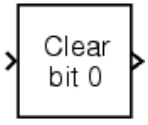
Set the specified bit of the stored integer to zero

Library

Logic and Bit Operations

Description

The Bit Clear block sets the specified bit, given by its index, of the stored integer to zero. Scaling is ignored.

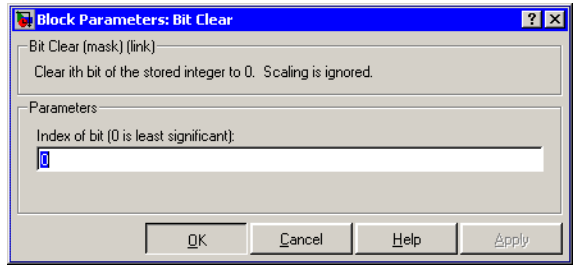


You can specify the bit to be set to zero with the **Index of bit** parameter, where bit zero is the least significant bit.

Data Type Support

The Bit Clear block supports Simulink integer, fixed-point, and Boolean data types. True floating-point data types are not supported.

Parameters and Dialog Box



Index of bit

Index of bit where bit 0 is the least significant bit.

Examples

If the Bit Clear block is turned on for bit 2, bit 2 is set to 0. A vector of constants `2.^[0 1 2 3 4]` is represented in binary as [00001 00010 00100 01000 10000]. With bit 2 set to 0, the result is [00001 00010 00000 01000 10000], which is represented in decimal as `[1 2 0 8 16]`.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also Bit Set

Purpose

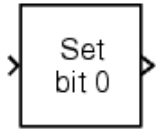
Set the specified bit of the stored integer to one

Library

Logic and Bit Operations

Description

The Bit Set block sets the specified bit of the stored integer to one. Scaling is ignored.

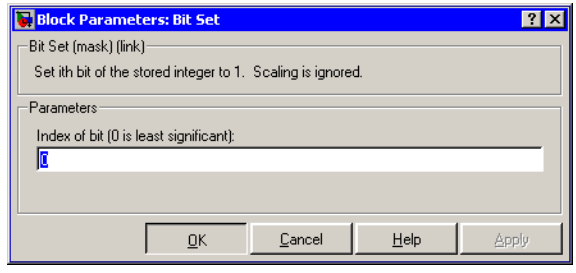


You can specify the bit to be set to one with the **Index of bit** parameter, where bit zero is the least significant bit.

Data Type Support

The Bit Set block supports Simulink integer, fixed-point, and Boolean data types. True floating-point data types are not supported.

Parameters and Dialog Box



Index of bit

Index of bit where bit 0 is the least significant bit.

Examples

If the Bit Set block is turned on for bit 2, bit 2 is set to 1. A vector of constants `2.^[0 1 2 3 4]` is represented in binary as [00001 00010 00100 01000 10000]. With bit 2 set to 1, the result is [00101 00110 00100 01100 10100], which is represented in decimal as `[5 6 4 12 20]`.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also Bit Clear

Purpose

Perform the specified bitwise operation on the inputs

Library

Logic and Bit Operations

Description

The Bitwise Operator block performs the specified bitwise operation on its operands.

Unlike the logic operations performed by the Logical Operator block, bitwise operations treat the operands as a vector of bits rather than a single number. You select the bitwise Boolean operation from the**Operator** parameter list. The supported operations are given below.

Operation	Description
AND	TRUE if the corresponding bits are all TRUE
OR	TRUE if at least one of the corresponding bits is TRUE
NAND	TRUE if at least one of the corresponding bits is FALSE
NOR	TRUE if no corresponding bits are TRUE
XOR	TRUE if an odd number of corresponding bits are TRUE
NOT	TRUE if the input is FALSE (available only for single input)

The Bitwise Operator block does not support shift operations. For shift operations, see the Shift Arithmetic block.

The size of the output of the Bitwise Operator block depends on the number of inputs, their vector size, and the selected operator:

- The NOT operator accepts only one input, which can be a scalar or a vector. If the input is a vector, the output is a vector of the same size containing the bitwise logical complements of the input vector elements.

- For a single vector input, the block applies the operation (except the NOT operator) to all elements of the vector. If a bit mask is not specified, then the output is a scalar. If a bit mask is specified, then the output is a vector.
- For two or more inputs, the block performs the operation between all of the inputs. If the inputs are vectors, the operation is performed between corresponding elements of the vectors to produce a vector output.

When configured as a multi-input XOR gate, this block performs an addition- modulo-two operation as mandated by the IEEE Standard for Logic Elements.

If you do not select the **Use bit mask** check box, then the block can accept multiple inputs. You select the number of input ports from the **Number of input ports** parameter. The input data types must be identical.

If you select the **Use bit mask** check box, then a single input is associated with the bit mask you specify from the **Bit Mask** parameter. You specify the bit mask using any valid MATLAB expression. For example, you can specify the bit mask 00100101 as `2^5+2^2+2^0`. Alternatively, you can use strings to specify a hexadecimal bit mask such as `{'FE73','12AC'}`. If the bit mask is larger than the input signal data type, then it is ignored.

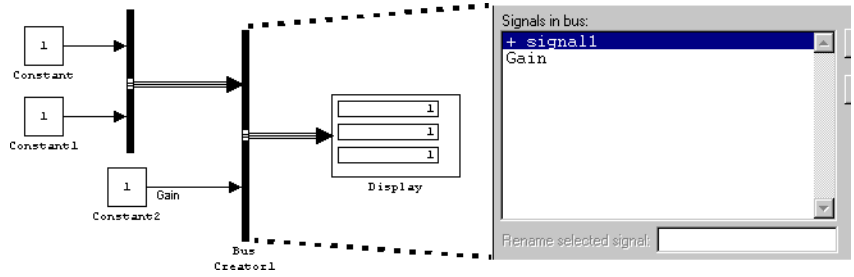
Note The output data type, which is inherited from the driving block, should represent zero exactly. Data types that satisfy this condition include signed and unsigned integers and any floating-point data type.

The **Treat mask as** parameter list controls how the mask is treated. The possible values are `Real World Value` and `Stored Integer`. In terms of the general encoding scheme described in the “Scaling” section of the Simulink Fixed Point documentation, `Real World Value` treats

the mask as $V = SQ + B$ where S is the slope and B is the bias. `Stored Integer` treats the mask as a stored integer, Q.

You can use the bit mask to perform a bit set or a bit clear on the input. To perform a bit set, set the **Operator** parameter list to `OR` and create a bit mask with a 1 for each corresponding input bit that you want to set to 1. To perform a bit clear, set the **Operator** parameter list to `AND` and create a bit mask with a 0 for each corresponding input bit that you want to set to 0.

For example, suppose you want to perform a bit set on the fourth bit of an 8-bit input vector. The bit mask would be 00010000, which you can specify as `2^4` in the **Bit mask** parameter. To perform a



The Bus Creator block generates names for bus signals whose corresponding inputs do not have names. The names are of the form `signaln` where `n` is the number of the port to which the input signal is connected.

You can change the name of any signal by editing its name on the block diagram or in the **Signal Properties** dialog box. If you change a name in this way while the Bus Creator block's dialog box is open, you must close and reopen the dialog box or click the **Refresh** button next to the **Signals in bus** list to update the name in the dialog box.

To specify that the bus inputs must have specific names, select `Require input signal names to match signals below` from the list box on the block's parameter dialog box. The block's parameter dialog box displays the names of the signals currently connected to its inputs or a generated name (for example, signal2) for an anonymous input. You can now use the parameter dialog box to change the required names of the block's inputs. To change the required signal name, select the signal in the **Signals in bus** list. The selected signal's name appears in the **Rename selected signal** field. Edit the name in the field and click the parameter dialog box's **Apply** button to apply your edits or the **OK** button to apply the edits and close the dialog box.

Browsing Bus Signals

The **Signals in bus** list on a Bus Creator block's parameter dialog displays a list of the signals entering the block. A plus sign (+) sign next to a signal indicates that the signal is itself a bus. You can display its contents by clicking the plus sign. If the expanded input includes bus signals, plus signs appear next to the names of those bus signals. You

can expand them as well. In this way, you can view all signals entering the block, including those entering via buses. To find the source of any signal entering the block, select the signal in the **Signals in bus** list and click the adjacent **Find** button. Simulink opens the subsystem containing the signal source, if necessary, and highlights the source's icon.

Note Simulink hides the name of a Bus Creator block when you copy it from the Simulink library to a model.

Data Type Support

The Bus Creator block accepts and outputs real or complex values of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, refer to "Data Types Supported by Simulink" in Using Simulink.

Parameters and Dialog Box

Signal naming options

Select `Inherit bus signal names from input ports` to assign input signal names to the corresponding bus signals. Select `Require input signal names to match signals below` to specify that inputs must have the names listed in the **Signals in bus** list. Selecting this option enables the **Rename selected signal** field.

Number of inputs

Specifies the number of input ports on this block.

Signals in bus

The **Signals in bus list box** shows the signals in the output bus. A plus sign (+) next to a signal name indicates that the signal is itself a bus. Click the plus sign to display the subsidiary bus signals. Click the **Refresh** button to update the list after editing

Signals in the bus

The **Signals in the bus** list shows the signals in the input bus. Use the **Select>>** button to select output signals. To find the source of any signal entering the block, select the signal in the **Signals in the bus** list and click the adjacent **Find** button. Simulink opens the subsystem containing the signal source, if necessary, and highlights the source's icon. To refresh the display (e.g., to reflect modifications to the bus connected to the block), click the adjacent **Refresh** button.

Selected signals

The **Selected signals** list box shows the output signals. You can order the signals by using the **Up**, **Down**, and **Remove** buttons. Port connectivity is maintained when the signal order is changed.

If an output signal listed in the **Selected signals** list box is not an input to the Bus Selector block, the signal name is preceded by three question marks (`???`).

Output as bus

If selected, this option causes the block to output the selected elements as a bus. Otherwise, the block outputs the elements as standalone signals, each from its own output port and labeled with the corresponding element's name.

Check Discrete Gradient

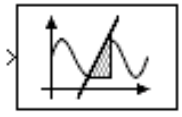
Purpose

Check that the absolute value of the difference between successive samples of a discrete signal is less than an upper bound

Library

Model Verification

Description



The Check Discrete Gradient block checks each signal element at its input to determine whether the absolute value of the difference between successive samples of the element is less than an upper bound. The block's parameter dialog box allows you to specify the value of the upper bound (1 by default). If the verification condition is true, the block does nothing. Otherwise, the block halts the simulation, by default, and displays an error message in the Simulation Diagnostics Viewer.

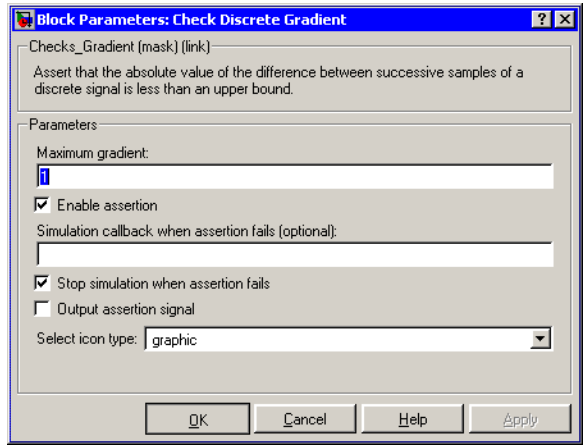
The **Model Verification block enabling** setting on the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box lets you enable or disable all model verification blocks, including Check Discrete Gradient blocks, in a model.

The Check Discrete Gradient block and its companion blocks in the Model Verification library are intended to facilitate creation of self-validating models. For example, you can use model verification blocks to test that signals do not exceed specified limits during simulation. When you are satisfied that a model is correct, you can turn error checking off by disabling the verification blocks. You do not have to physically remove them from the model. If you need to modify a model, you can temporarily turn the verification blocks back on to ensure that your changes do not break the model.

Data Type Support

The Check Discrete Gradient block accepts `single`, `double`, `int8`, `int16`, and `int32` input signals of any dimensions.

Parameters and Dialog Box



Maximum gradient

Upper bound on the gradient of the discrete input signal.

Enable assertion

Unchecking this option disables the Check Discrete Gradient block, that is, causes the model to behave as if the block did not exist. The **Model Verification block enabling** setting on the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box allows you to enable or disable all Check Discrete Gradient blocks in a model regardless of the setting of this option.

Simulation callback when assertion fails

An M-expression to be evaluated when the assertion fails.

Stop simulation when assertion fails

If checked, this option causes the Check Discrete Gradient block to halt the simulation when the block's input is zero and display an error message in Simulink's **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB Command Window and continues the simulation.

Output assertion signal

If checked, this option causes the block to output a Boolean signal that is true (`1`) at each time step if the assertion succeeds and false (`0`) if the assertion fails. The data type of the output signal is `Boolean` if you have selected the `Implement logic signals as boolean data` option on the **Simulation and code generation** optimization pane of Simulink's **Configuration Parameters** dialog box. Otherwise the data type of the output signal is `double`.

Select icon type

Type of icon used to display this block in a block diagram: either `graphic` or `text`. The `graphic` option displays a graphical representation of the assertion condition on the icon. The `text` option displays a mathematical expression that represents the assertion condition. If the icon is too small to display the expression, the text icon displays an exclamation point. To see the expression, enlarge the icon.

Characteristics

Direct Feedthrough	No
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

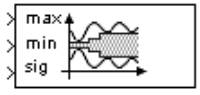
Purpose

Check that a gap of possibly varying width occurs in the range of a signal's amplitudes

Library

Model Verification

Description



The Check Dynamic Gap block checks that a gap of possibly varying width occurs in the range of a signal's amplitudes. The test signal is the signal connected to the input labeled *sig*. The inputs labeled *min* and *max* specify the lower and upper bounds of the dynamic gap, respectively. If the verification condition is true, the block does nothing. If not, the block halts the simulation, by default, and displays an error message.

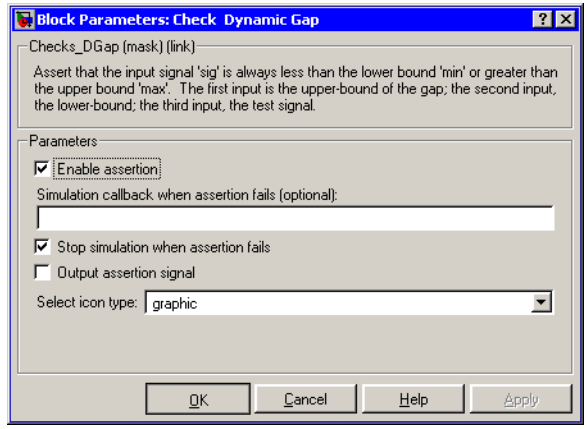
The Check Dynamic Gap block and its companion blocks in the Model Verification library are intended to facilitate creation of self-validating models. For example, you can use model verification blocks to test that signals do not exceed specified limits during simulation. When you are satisfied that a model is correct, you can turn error checking off by disabling the verification blocks. You do not have to physically remove them from the model. If you need to modify a model, you can temporarily turn the verification blocks back on to ensure that your changes do not break the model.

Data Type Support

The Check Dynamic Gap block accepts input signals of any dimensions and of any data type supported by Simulink. All three input signals must have the same dimension and data type. If the inputs are nonscalar, the block checks each element of the input test signal to the corresponding elements of the reference signals.

For a discussion on the data types supported by Simulink, see "Data Types Supported by Simulink" in Using Simulink.

Parameters and Dialog Box



Enable assertion

Unchecking this option disables the Check Dynamic Gap block, that is, causes the model to behave as if the block did not exist. The **Model Verification block enabling** setting on the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box allows you to enable or disable all model verification blocks in a model, including Check Dynamic Gap blocks, regardless of the setting of this option.

Simulation callback when assertion fails

An M-expression to be evaluated when the assertion fails.

Stop simulation when assertion fails

If checked, this option causes the Assertion block to halt the simulation when the block's input is zero and display an error message in the **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB Command Window and continues the simulation.

Output assertion signal

If checked, this option causes the block to output a Boolean signal that is true (`1`) at each time step if the assertion succeeds and false (`0`) if the assertion fails. The data type of the output signal is `Boolean` if you have selected the `Implement logic signals as`

`boolean data` option on the **Simulation and code generation** optimization pane of the **Configuration Parameters** dialog box. Otherwise the data type of the output signal is `double`.

Select Icon Type

Type of icon used to display this block in a block diagram: either `graphic` or `text`. The `graphic` option displays a graphical representation of the assertion condition on the icon. The `text` option displays a mathematical expression that represents the assertion condition. If the icon is too small to display the expression, the text icon displays an exclamation point. To see the expression, enlarge the icon.

Characteristics

Direct Feedthrough	No
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

Check Dynamic Lower Bound

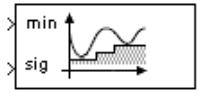
Purpose

Check that one signal is always less than another signal

Library

Model Verification

Description



The Check Dynamic Lower Bound block checks that the amplitude of a test signal is less than the amplitude of a reference signal at the current time step. The test signal is the signal connected to the input labeled *sig*. If the verification condition is true, the block does nothing. If not, the block halts the simulation, by default, and displays an error message.

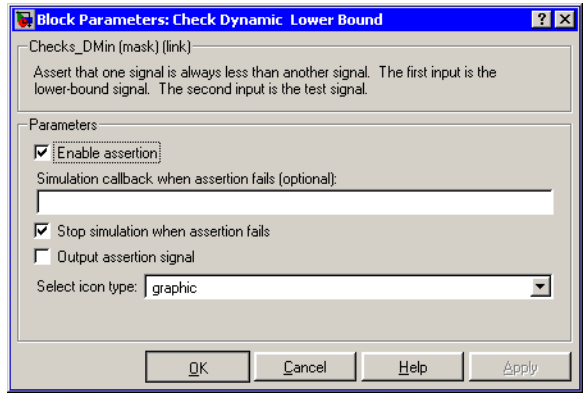
The Check Dynamic Lower Bound block and its companion blocks in the Model Verification library are intended to facilitate creation of self-validating models. For example, you can use model verification blocks to test that signals do not exceed specified limits during simulation. When you are satisfied that a model is correct, you can turn error checking off by disabling the verification blocks. You do not have to physically remove them from the model. If you need to modify a model, you can temporarily turn the verification blocks back on to ensure that your changes do not break the model.

Data Type Support

The Check Dynamic Lower Bound block accepts input signals of any data type supported by Simulink. The test and the reference signals must have the same dimensions and data type. If the inputs are nonscalar, the block checks each element of the input test signal to the corresponding elements of the reference signal.

For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in Using Simulink.

Parameters and Dialog Box



Enable Assertion

Unchecking this option disables the Check Dynamic Lower Bound block, that is, causes the model to behave as if the block did not exist. The **Model Verification block enabling** setting on the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box allows you to enable or disable all model verification blocks, including Check Dynamic Lower Bound blocks, in a model regardless of the setting of this option.

Simulation callback when assertion fails

An M-expression to be evaluated when the assertion fails.

Stop simulation when assertion fails

If checked, this option causes the Check Dynamic Lower Bound block to halt the simulation when the block's input is zero and display an error message in the **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB Command Window and continues the simulation.

Output assertion signal

If checked, this option causes the block to output a Boolean signal that is true (`1`) at each time step if the assertion succeeds and false (`0`) if the assertion fails. The data type of the output signal is `Boolean` if you have selected the `Implement logic signals as boolean data` option on the **Simulation and code generation**

optimization pane of the **Configuration Parameters** dialog box. Otherwise the data type of the output signal is `double`.

Select icon type

Type of icon used to display this block in a block diagram: either `graphic` or `text`. The `graphic` option displays a graphical representation of the assertion condition on the icon. The `text` option displays a mathematical expression that represents the assertion condition. If the icon is too small to display the expression, the text icon displays an exclamation point. To see the expression, enlarge the icon.

Characteristics

Direct Feedthrough	No
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

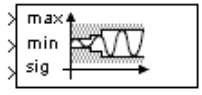
Purpose

Check that a signal falls inside a range of amplitudes that varies from time step to time step

Library

Model Verification

Description



The Check Dynamic Range block checks that a test signal falls inside a range of amplitudes at each time step. The width of the range can vary from time step to time step. The input labeled *sig* is the test signal. The inputs labeled *min* and *max* are the lower and upper bounds of the valid range at the current time step. If the verification condition is true, the block does nothing. If not, the block halts the simulation, by default, and displays an error message.

The Check Dynamic Range block and its companion blocks in the Model Verification library are intended to facilitate creation of self-validating models. For example, you can use model verification blocks to test that signals do not exceed specified limits during simulation. When you are satisfied that a model is correct, you can turn error checking off by disabling the verification blocks. You do not have to physically remove them from the model. If you need to modify a model, you can temporarily turn the verification blocks back on to ensure that your changes do not break the model.

Data Type Support

The Check Dynamic Range block accepts input signals of any dimensions and of any data type supported by Simulink. All three input signals must have the same dimension and data type. If the inputs are nonscalar, the block checks each element of the input test signal to the corresponding elements of the reference signals.

For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in Using Simulink.

Check Dynamic Range

Parameters and Dialog Box

Enable assertion
Unchecking this option disables the Check Dynamic Range block, that is, causes the model to behave as if the block did not exist. The **Model Verification block enabling** setting on the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box allows you to enable or disable all model verification blocks in a model, including Check Dynamic Range blocks, regardless of the setting of this option.

Simulation callback when assertion fails
An M-expression to be evaluated when the assertion fails.

Stop simulation when assertion fails
If checked, this option causes the Assertion block to halt the

blocks, including Check Static Gap blocks, in a model regardless of the setting of this option.

Simulation callback when assertion fails
An M-expression to be evaluated when the assertion fails.

Stop simulation when assertion fails
If checked, this option causes the Check Static Gap block to halt the simulation when the block's input is zero and display an error message in the **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB Command Window and continues the simulation.

Output Assertion Signal
If checked, this option causes the block to output a Boolean signal that is true (`1`) at each time step if the assertion succeeds and false (`0`) if the assertion fails. The data type of the output signal is `Boolean` if you have selected the `Implement logic signals as boolean data` option on the **Simulation and code generation** optimization pane of the **Configuration Parameters** dialog box. Otherwise the data type of the output signal is `double`.

Select Icon Type
Type of icon used to display this block in a block diagram: either `graphic` or `text`. The `graphic` option displays a graphical representation of the assertion condition on the icon. The `text` option displays a mathematical expression that represents the assertion condition. If the icon is too small to display the expression, the text icon displays an exclamation point. To see the expression, enlarge the icon.

Characteristics

Direct Feedthrough	No
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

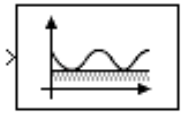
Purpose

Check that a signal is greater than (or optionally equal to) a static lower bound

Library

Model Verification

Description



The Check Static Lower Bound block checks that each element of the input signal is greater than (or optionally equal to) a specified lower bound at the current time step. The block's parameter dialog box allows you to specify the value of the lower bound and whether the lower bound is inclusive. If the verification condition is true, the block does nothing. If not, the block halts the simulation, by default, and displays an error message.

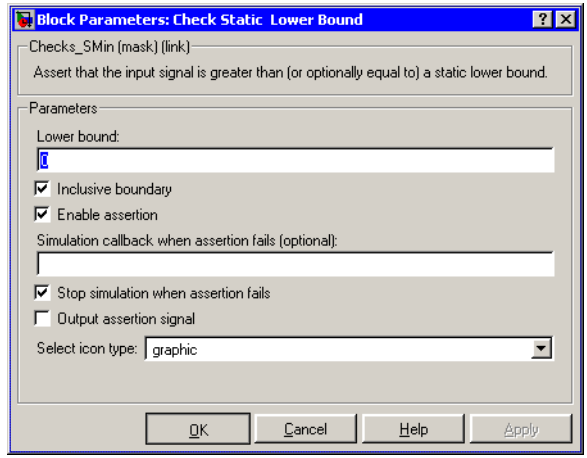
The Check Static Lower Bound block and its companion blocks in the Model Verification library are intended to facilitate creation of self-validating models. For example, you can use model verification blocks to test that signals do not exceed specified limits during simulation. When you are satisfied that a model is correct, you can turn error checking off by disabling the verification blocks. You do not have to physically remove them from the model. If you need to modify a model, you can temporarily turn the verification blocks back on to ensure that your changes do not break the model.

Data Type Support

The Check Static Lower Bound block accepts input signals of any dimensions and of any data type supported by Simulink.

For a discussion on the data types supported by Simulink, see "Data Types Supported by Simulink" in Using Simulink.

Parameters and Dialog Box



Lower bound

Lower bound on the range of amplitudes that the input signal can have.

Inclusive boundary

Checking this option makes the range of valid input amplitudes include the lower bound.

Enable Assertion

Unchecking this option disables the Check Static Lower Bound block, that is, causes the model to behave as if the block did not exist. The **Model Verification block enabling** setting on the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box allows you to enable or disable all model verification blocks, including Check Static Lower Bound blocks, in a model regardless of the setting of this option.

Simulation callback when assertion fails

An M-expression to be evaluated when the assertion fails.

Stop simulation when assertion fails

If checked, this option causes the Check Static Lower Bound block to halt the simulation when the block's input is zero and

display an error message in the **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB Command Window and continues the simulation.

Output Assertion Signal

If checked, this option causes the block to output a Boolean signal that is true (`1`) at each time step if the assertion succeeds and false (`0`) if the assertion fails. The data type of the output signal is `Boolean` if you have selected the `Implement logic signals as boolean data` option on the **Simulation and code generation** optimization pane of the **Configuration Parameters** dialog box. Otherwise the data type of the output signal is `double`.

Select Icon Type

Type of icon used to display this block in a block diagram: either `graphic` or `text`. The `graphic` option displays a graphical representation of the assertion condition on the icon. The `text` option displays a mathematical expression that represents the assertion condition. If the icon is too small to display the expression, the text icon displays an exclamation point. To see the expression, enlarge the icon.

Characteristics

Direct Feedthrough	No
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

Check Static Range

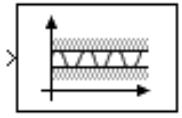
Purpose

Check that a signal falls inside a fixed range of amplitudes

Library

Model Verification

Description



The Check Static Range block checks that each element of the input signal falls inside the same range of amplitudes at each time step. The block's parameter dialog box allows you to specify the upper and lower bounds of the valid amplitude range and whether the range includes the bounds. If the verification condition is true, the block does nothing. If not, the block halts the simulation, by default, and displays an error message.

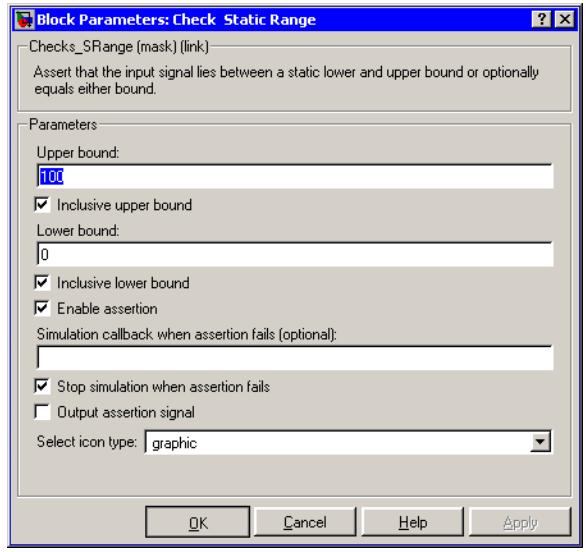
The Check Static Range block and its companion blocks in the Model Verification library are intended to facilitate creation of self-validating models. For example, you can use model verification blocks to test that signals do not exceed specified limits during simulation. When you are satisfied that a model is correct, you can turn error checking off by disabling the verification blocks. You do not have to physically remove them from the model. If you need to modify a model, you can temporarily turn the verification blocks back on to ensure that your changes do not break the model.

Data Type Support

The Check Static Range block accepts input signals of any dimensions and of any data type supported by Simulink.

For a discussion on the data types supported by Simulink, see "Data Types Supported by Simulink" in Using Simulink.

Parameters and Dialog Box



Upper bound

Upper bound of the range of valid input signal amplitudes.

Inclusive upper bound

Checking this option specifies that the valid signal range includes the upper bound.

Lower bound

Lower bound of the range of valid input signal amplitudes.

Inclusive lower bound

Checking this option specifies that the valid signal range includes the lower bound.

Enable Assertion

Unchecking this option disables the Check Static Range block, that is, causes the model to behave as if the block did not exist. The **Model Verification block enabling** setting on the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box allows you to enable or disable all model verification

blocks, including Check Static Range blocks, in a model regardless of the setting of this option.

Simulation callback when assertion fails
An M-expression to be evaluated when the assertion fails.

Stop simulation when assertion fails
If checked, this option causes the Check Static Range block to halt the simulation when the block's input is zero and display an error message in the **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB Command Window and continues the simulation.

Output Assertion Signal
If checked, this option causes the block to output a Boolean signal that is true (`1`) at each time step if the assertion succeeds and false (`0`) if the assertion fails. The data type of the output signal is `Boolean` if you have selected the `Implement logic signals as boolean data` option on the **Simulation and code generation** optimization pane of the **Configuration Parameters** dialog box. Otherwise the data type of the output signal is `double`.

Select Icon Type
Type of icon used to display this block in a block diagram: either `graphic` or `text`. The `graphic` option displays a graphical representation of the assertion condition on the icon. The `text` option displays a mathematical expression that represents the assertion condition. If the icon is too small to display the expression, the text icon displays an exclamation point. To see the expression, enlarge the icon.

Characteristics

Direct Feedthrough	No
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

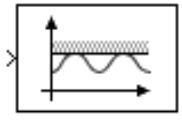
Purpose

Check that a signal is greater than (or optionally equal to) a static lower bound

Library

Model Verification

Description



The Check Static Upper Bound block checks that each element of the input signal is less than (or optionally equal to) a specified lower bound at the current time step. The block's parameter dialog box allows you to specify the value of the upper bound and whether the bound is inclusive. If the verification condition is true, the block does nothing. If not, the block halts the simulation, by default, and displays an error message.

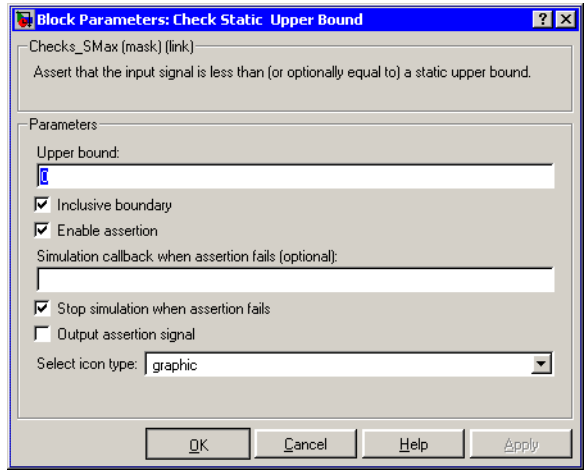
The Check Static Upper Bound block and its companion blocks in the Model Verification library are intended to facilitate creation of self-validating models. For example, you can use model verification blocks to test that signals do not exceed specified limits during simulation. When you are satisfied that a model is correct, you can turn error checking off by disabling the verification blocks. You do not have to physically remove them from the model. If you need to modify a model, you can temporarily turn the verification blocks back on to ensure that your changes do not break the model.

Data Type Support

The Check Static Upper Bound block accepts input signals of any dimensions and of any data type supported by Simulink.

For a discussion on the data types supported by Simulink, see "Data Types Supported by Simulink" in Using Simulink.

Parameters and Dialog Box



Upper bound
Upper bound on the range of amplitudes that the input signal can have.

Inclusive boundary
Checking this option makes the range of valid input amplitudes include the upper bound.

Enable Assertion
Unchecking this option disables the Check Static Upper Bound block, that is, causes the model to behave as if the block did not exist. The **Model Verification block enabling** setting on the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box allows you to enable or disable all model verification blocks, including Check Static Lower Bound blocks, in a model regardless of the setting of this option.

Simulation callback when assertion fails
An M-expression to be evaluated when the assertion fails.

Stop simulation when assertion fails
If checked, this option causes the Check Static Upper Bound block to halt the simulation when the block's input is zero and

display an error message in the **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB Command Window and continues the simulation.

Output Assertion Signal

If checked, this option causes the block to output a Boolean signal that is true (`1`) at each time step if the assertion succeeds and false (`0`) if the assertion fails. The data type of the output signal is `Boolean` if you have selected the `Implement logic signals as boolean data` option on the **Simulation and code generation** optimization pane of the **Configuration Parameters** dialog box. Otherwise the data type of the output signal is `double`.

Select Icon Type

Type of icon used to display this block in a block diagram: either `graphic` or `text`. The `graphic` option displays a graphical representation of the assertion condition on the icon. The `text` option displays a mathematical expression that represents the assertion condition. If the icon is too small to display the expression, the text icon displays an exclamation point. To see the expression, enlarge the icon.

Characteristics

Direct Feedthrough	No
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

Chirp Signal

Purpose

Generate a sine wave with increasing frequency

Library

Sources

Description

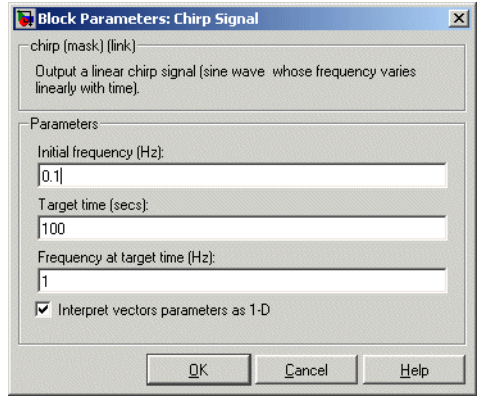
The Chirp Signal block generates a sine wave whose frequency increases at a linear rate with time. You can use this block for spectral analysis of nonlinear systems. The block generates a scalar or vector output.

The parameters, **Initial frequency**, **Target time**, and **Frequency at target time**, determine the block's output. You can specify any or all of these variables as scalars or arrays. All the parameters specified as arrays must have the same dimensions. The block expands scalar parameters to have the same dimensions as the array parameters. The block output has the same dimensions as the parameters unless you select the **Interpret vector parameters as 1-D** option. If you select this option and the parameters are row or column vectors, the block outputs a vector (1-D array) signal.

Data Type Support

The Chirp Signal block outputs a real-valued signal of type `double`.

Parameters and Dialog Box



Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in Using Simulink for details.

Initial frequency
The initial frequency of the signal, specified as a scalar or matrix value. The default is `0.1` Hz.

Target time
The time at which the frequency reaches the **Frequency at target time** parameter value, a scalar or matrix value. The frequency continues to change at the same rate after this time. The default is `100` seconds.

Frequency at target time
The frequency of the signal at the target time, a scalar or matrix value. The default is `1` Hz.

Interpret vector parameters as 1-D
If selected, column or row matrix values for the **Initial frequency**, **Target time**, and **Frequency at target time** parameters result in a vector output whose elements are the elements of the row or column.

Characteristics

Sample Time	Continuous
Scalar Expansion	Yes, of parameters
Dimensionalized	Yes
Zero Crossing	No

Clock

Purpose

Display and provide the simulation time

Library

Sources

Description

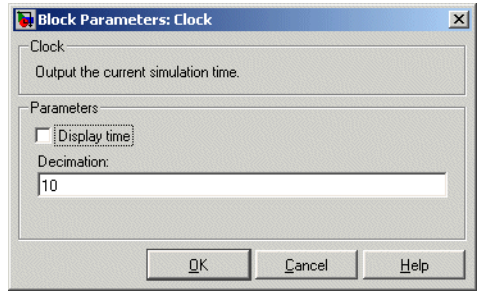
The Clock block outputs the current simulation time at each simulation step. This block is useful for other blocks that need the simulation time.

When you need the current time within a discrete system, use the Digital Clock block.

Data Type Support

The Clock block outputs a real-valued signal of type `double`.

Parameters and Dialog Box



Display time

Use the **Display time** check box to display the current simulation time inside the Clock icon.

Decimation

The **Decimation** parameter value is the increment at which the clock is updated; it can be any positive integer. For example, if the decimation is 1000, then, for a fixed integration step of 1 millisecond, the clock updates at 1 second, 2 seconds, and so on.

Characteristics

Sample Time	Continuous
Scalar Expansion	N/A

Dimensionalized	No
Zero Crossing	No

Purpose

Implement a truth table

Library

Logic and Bit Operations

Description

The Combinatorial Logic block implements a standard truth table for modeling programmable logic arrays (PLAs), logic circuits, decision tables, and other Boolean expressions. You can use this block in conjunction with Memory blocks to implement finite-state machines or flip-flops.

You specify a matrix that defines all possible block outputs as the **Truth table** parameter. Each row of the matrix contains the output for a different combination of input elements. You must specify outputs for every combination of inputs. The number of columns is the number of block outputs.

The relationship between the number of inputs and the number of rows is

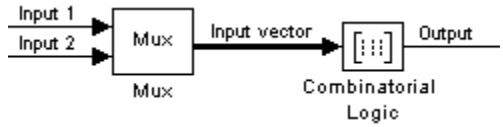
```
number of rows = 2 ^ (number of inputs)
```

Simulink returns a row of the matrix by computing the row's index from the input vector elements. Simulink computes the index by building a binary number where input vector elements having zero values are 0 and elements having nonzero values are 1, then adding 1 to the result. For an input vector, `u`, of `m` elements,

$$\texttt{row index} = 1 + u(m)*2^0 + u(m-1)*2^1 + \ldots + u(1)*2^{m-1}$$

Example of Two-Input AND Function

This example builds a two-input AND function, which returns 1 when both input elements are 1, and 0 otherwise. To implement this function, specify the **Truth table** parameter value as `[0; 0; 0; 1]`. The portion of the model that provides the inputs to and the output from the Combinatorial Logic block might look like this.



The following table indicates the combination of inputs that generate each output. The input signal labeled "Input 1" corresponds to the column in the table labeled Input 1. Similarly, the input signal "Input 2" corresponds to the column with the same name. The combination of these values determines the row of the Output column of the table that is passed as block output.

For example, if the input vector is `[1 0]`, the input references the third row:

```
(21*1 + 1)
```

The output value is 0.

Row	Input 1	Input 2	Output
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	1

Example of Circuit

This sample circuit has three inputs: the two bits (**a** and **b**) to be summed and a carry-in bit (**c**). It has two outputs: the carry-out bit (**c'**) and the sum bit (**s**). Here are the truth table and the outputs associated with each combination of input values for this circuit.

Inputs			Outputs	
a	b	c	c'	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

To implement this adder with the Combinatorial Logic block, you enter the 8-by-2 matrix formed by columns **c'** and **s** as the **Truth table** parameter.

You can also implement sequential circuits (that is, circuits with states) with the Combinatorial Logic block by including an additional input for the state of the block and feeding the output of the block back into this state input.

Data Type Support

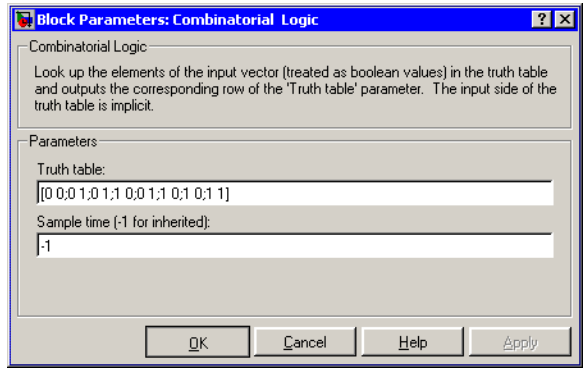
The type of signals accepted by a Combinatorial Logic block depends on whether you selected the Boolean logic signals option (see "Enabling Strict Boolean Type Checking" in Using Simulink). If this option is enabled, the block accepts real signals of type `Boolean` or `double`. The truth table can have Boolean values (0 or 1) of any data type. If the table contains non-Boolean values, the table's data type must be `double`.

The type of the output is the same as that of the input except that the block outputs `double` if the input is `Boolean` and the truth table contains non-Boolean values.

If Boolean compatibility mode is disabled, the Combinatorial Logic block accepts only signals of type `Boolean`. The block outputs `double` if

the truth table contains non-Boolean values of type `double`. Otherwise, the output is `Boolean`.

Parameters and Dialog Box



Truth table

The matrix of outputs. Each column corresponds to an element of the output vector and each row corresponds to a row of the truth table.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See “Specifying Sample Time” in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	Yes; the output width is the number of columns of the **Truth table** parameter
Zero Crossing	No

Compare To Constant

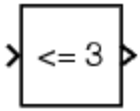
Purpose

Determine how a signal compares to the specified constant

Library

Logic and Bit Operations

Description



The Compare To Constant block compares an input signal to a constant. Specify the constant in the **Constant value** parameter. Specify how the input is compared to the constant value with the **Operator** parameter. The **Operator** parameter can have the following values:

- == — Determine whether the input is equal to the specified constant.
- ~= — Determine whether the input is not equal to the specified constant.
- < — Determine whether the input is less than the specified constant.
- <= — Determine whether the input is less than or equal to the specified constant.
- > — Determine whether the input is greater than the specified constant.
- >= — Determine whether the input is greater than or equal to the specified constant.

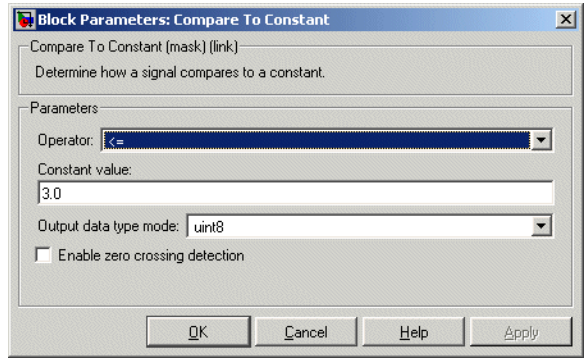
The output is `0` if the comparison is false, and `1` if it is true.

Data Type Support

The Compare To Constant block accepts inputs of any data type supported by Simulink, including fixed-point data types. The block output is `uint8` or `Boolean` as specified by the **Output data type mode** parameter.

For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in Using Simulink.

Parameters and Dialog Box



Operator

Specify how the input is compared to the constant value, as discussed in Description.

Constant value

Specify the constant value to which the input is compared.

Output data type mode

Specify the data type of the output, `uint8` or `Boolean`.

Enable zero crossing detection

Select to enable zero-crossing detection. For more information, see "Zero-Crossing Detection" in Using Simulink.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes
Zero Crossing	Yes, if enabled.

See Also

Compare To Zero

Compare To Zero

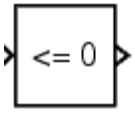
Purpose

Determine how a signal compares to zero

Library

Logic and Bit Operations

Description



The Compare To Zero block compares an input signal to zero. Specify how the input is compared to zero with the **Operator** parameter. The **Operator** parameter can have the following values:

- == — Determine whether the input is equal to zero.
- ~= — Determine whether the input is not equal to zero.
- < — Determine whether the input is less than zero.
- <= — Determine whether the input is less than or equal to zero.
- > — Determine whether the input is greater than zero.
- >= — Determine whether the input is greater than or equal to zero.

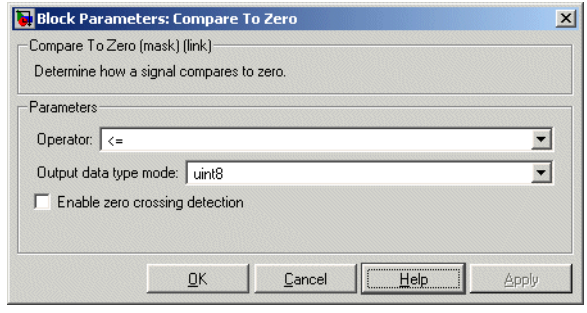
The output is `0` if the comparison is false, and `1` if it is true.

Data Type Support

The Compare To Zero block accepts inputs of any data type supported by Simulink, including fixed-point data types. The block output is `uint8` or `Boolean` as specified by the **Output data type mode** parameter.

For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in Using Simulink.

Parameters and Dialog Box



Operator
Specify how the input is compared to zero, as discussed in Description.

Output data type mode
Specify the data type of the output, `uint8` or `Boolean`.

Enable zero crossing detection
Select to enable zero-crossing detection. For more information, see "Zero-Crossing Detection" in Using Simulink.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes
Zero Crossing	Yes, if enabled.

See Also

Compare To Constant

Complex to Magnitude-Angle

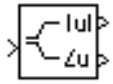
Purpose

Compute the magnitude and/or phase angle of a complex signal

Library

Math Operations

Description

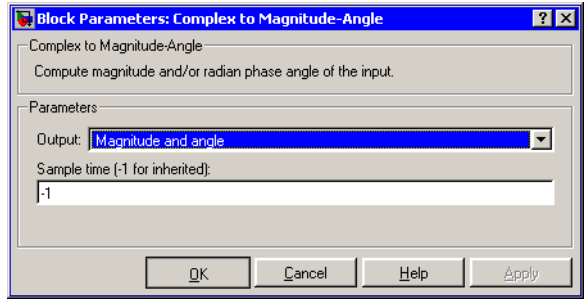


The Complex to Magnitude-Angle block accepts a complex-valued signal of type `double`. It outputs the magnitude and/or phase angle of the input signal, depending on the setting of the **Output** parameter. The outputs are real values of type `double`. The input can be an array of complex signals, in which case the output signals are also arrays. The magnitude signal array contains the magnitudes of the corresponding complex input elements. The angle output similarly contains the angles of the input elements.

Data Type Support

See the preceding description.

Parameters and Dialog Box



Output

Determines the output of this block. Choose from the following values: `MagnitudeAndAngle` (outputs the input signal's magnitude and phase angle in radians), `Magnitude` (outputs the input's magnitude), `Angle` (outputs the input's phase angle in radians).

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in Using Simulink for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

Complex to Real-Imag

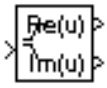
Purpose

Output the real and imaginary parts of a complex input signal

Library

Math Operations

Description

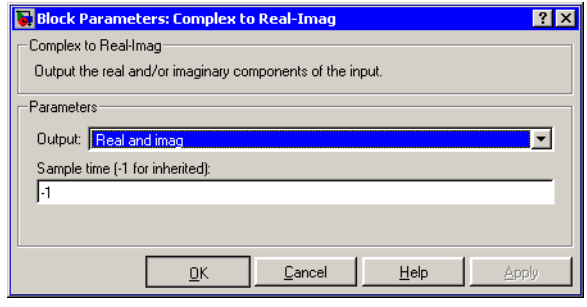


The Complex to Real-Imag block accepts a complex-valued signal of any data type supported by Simulink, including fixed-point data types. It outputs the real and/or imaginary part of the input signal, depending on the setting of the **Output** parameter. The real outputs are of the same data type as the complex input. The input can be an array (vector or matrix) of complex signals, in which case the output signals are arrays of the same dimensions. The real array contains the real parts of the corresponding complex input elements. The imaginary output similarly contains the imaginary parts of the input elements.

Data Type Support

See the preceding description. For a discussion on the data types supported by Simulink, see "Data Types Supported by Simulink" in Using Simulink.

Parameters and Dialog Box



Output

Determines the output of this block. Choose from the following values: RealAndImag (outputs the input signal's real and imaginary parts), Real (outputs the input's real part), Imag (outputs the input's imaginary part).

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See “Specifying Sample Time” in Using Simulink for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

Configurable Subsystem

Purpose

Represent any block selected from a user-specified library of blocks

Library

Ports & Subsystems

Description

The Configurable Subsystem block represents one of a set of blocks contained in a specified library of blocks. The block's context menu lets you choose which block the configurable subsystem represents.

Configurable Subsystem blocks simplify creation of models that represent families of designs. For example, suppose that you want to model an automobile that offers a choice of engines. To model such a design, you would first create a library of models of the engine types available with the car. You would then use a Configurable Subsystem block in your car model to represent the choice of engines. To model a particular variant of the basic car design, a user need only choose the engine type, using the configurable engine block's dialog.

To create a configurable subsystem in a model, you must first create a library containing a master configurable subsystem and the blocks that it represents. You can then create configurable instances of the master subsystem by dragging copies of the master subsystem from the library and dropping them into models.

You can add any type of block to a master configurable subsystem library. Simulink derives the port names for the configurable subsystem by making a unique list from the port names of all the choices. Note that Simulink uses default port names for non-subsystem block choices.

Note that Simulink does not allow you to break library links in a configurable subsystem because Simulink needs the links to reconfigure the subsystem when you choose a new configuration. Breaking links would be useful only if you never intended to reconfigure the subsystem, in which case you could simply replace the configurable subsystem with a nonconfigurable subsystem that implements the permanent configuration.

Creating a Master Configurable Subsystem

To create a master configurable subsystem:

1. Create a library of blocks representing the various configurations of the configurable subsystem.
2. Save the library.
3. Create an instance of the Configurable Subsystem block in the library.

   To do this, drag a copy of the Configurable Subsystem block from the Simulink Signals and Systems library into the library you created in the preceding step.
4. Display the Configurable Subsystem block's dialog by double-clicking it. The dialog displays a list of the other blocks in the library.
5. Select the blocks that represent the various configurations of the configurable subsystems you are creating.
6. Select Block Choice from the subsystem's context menu.

   The context menu displays a submenu listing the blocks that the subsystem can represent.
7. Select the block that you want the subsystem to represent by default.
8. Close the dialog.
9. Save the library.

Note If you add or remove blocks from a library, you must recreate any Configurable Subsystem blocks that use the library.

If you modify a library block that is the default block choice for a configurable subsystem, the change does not immediately propagate to the configurable subsystem. To propagate this change, do one of the following:

- Change the default block choice to another block in the subsystem, then change the default block choice back to the original block.
- Recreate the configurable subsystem block, including the selection of the updated block as the default block choice.

Creating an Instance of a Configurable Subsystem

To create an instance of a configurable subsystem in a model,

1. Open the library containing the master configurable subsystem.
2. Drag a copy of the master into the model.
3. Select **Block Choice** from the copy's context menu.
4. Select the block that you want the configurable subsystem to represent.

The instance of the configurable system displays the icon and parameter dialog box of the block that it represents.

Setting Instance Block Parameters

As with other blocks, you can use the parameter dialog box of a configurable subsystem instance to set the instance's parameters interactively and the `set_param` command to set the parameters from the MATLAB command line or in an M-file program. If you use `set_param`, you must specify the full path name of the configurable subsystem's current block choice as the first argument of `set_param`, e.g.,

```
curr_choice = get_param('mymod/myconfigsys', 'BlockChoice');
curr_choice = ['mymod/myconfigsys/' curr_choice];
set_param(curr_choice, 'MaskValues', ...);
```

Mapping I/O Ports

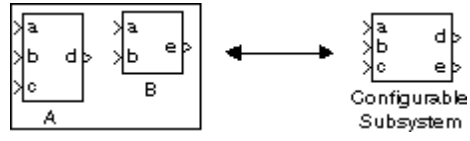
A configurable subsystem displays a set of input and output ports corresponding to input and output ports in the selected library.

Simulink uses the following rules to map library ports to Configurable Subsystem block ports:

- Map each uniquely named input/output port in the library to a separate input/output port of the same name on the Configurable Subsystem block.
- Map all identically named input/output ports in the library to the same input/output ports on the Configurable Subsystem block.
- Terminate any input/output port not used by the currently selected library block with a Terminator/Ground block.

This mapping allows a user to change the library block represented by a Configurable Subsystem block without having to rewire connections to the Configurable Subsystem block.

For example, suppose that a library contains two blocks A and B and that block A has input ports labeled `a`, `b`, and `c` and an output port labeled `d` and that block B has input ports labeled `a` and `b` and an output port labeled `e`. A Configurable Subsystem block based on this library would have three input ports labeled `a`, `b`, and `c`, respectively, and two output ports labeled `d` and `e`, respectively, as illustrated in the following figure.



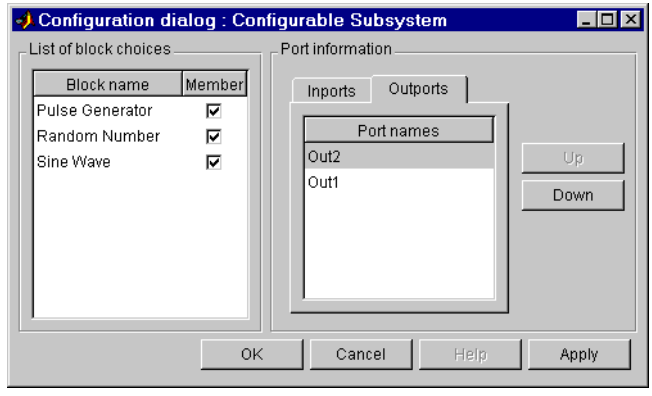
In this example, port `a` on the Configurable Subsystem block connects to port `a` of the selected library block no matter which block is selected. On the other hand, port `c` on the Configurable Subsystem block functions only if library block A is selected. Otherwise, it simply terminates.

Note A Configurable Subsystem block does not provide ports that correspond to non-I/O ports, such as the trigger and enable ports on triggered and enabled subsystems. Thus, you cannot use a Configurable Subsystem block directly to represent blocks that have such ports. You can do so indirectly, however, by wrapping such blocks in subsystem blocks that have input or output ports connected to the non-I/O ports.

Data Type Support

The Configurable Subsystem block accepts and outputs signals of the same types as are accepted or output by the block that it currently represents. The data types may be any supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



List of block choices

Select the blocks you want to include as members of the configurable subsystem. You can include user-defined subsystems as blocks.

Port information

Lists of input and output ports of member blocks. In the case of multiports, you can rearrange selected port positions by clicking the **Up** and **Down** buttons.

Characteristics

A Configurable Subsystem block has the characteristics of the block that it currently represents. Double-clicking the block opens the dialog box for the block that it currently represents.

Constant

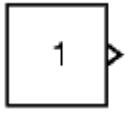
Purpose

Generate a constant value

Library

Sources

Description



The Constant block generates a real or complex constant value. The block generates scalar (`1x1` 2-D array), vector (1-D array), or matrix (2-D array) output, depending on the dimensionality of the **Constant value** parameter and the setting of the **Interpret vector parameters as 1-D** parameter.

The output of the block has the same dimensions and elements as the **Constant value** parameter. If you specify a vector for this parameter, and you want the block to interpret it as a vector (i.e., a 1-D array), select the **Interpret vector parameters as 1-D** parameter; otherwise, the block treats the **Constant value** parameter as a matrix (i.e., a 2-D array).

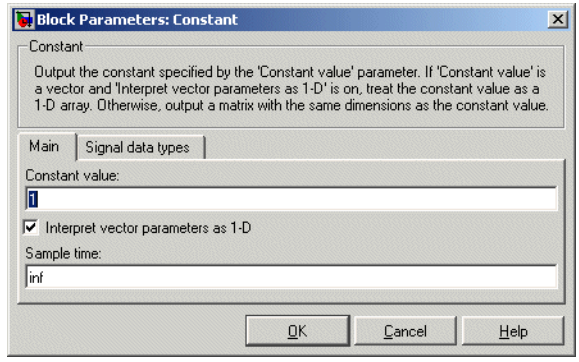
Data Type Support

By default, the Constant block outputs a signal whose data type and complexity are the same as that of the block's **Constant value** parameter. However, you can specify the output to be any supported data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see "Data Types Supported by Simulink" in Using Simulink.

Parameters and Dialog Box

The **Main** pane of the Constant block dialog appears as follows:



Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in Using Simulink for details.

Constant value

Specify the constant value output by the block. You can enter any MATLAB expression in this field, including the Boolean keywords, `true` or `false`, that evaluates to a matrix value. The **Constant value** parameter is converted from its data type to the specified output data type offline using round-to-nearest and saturation.

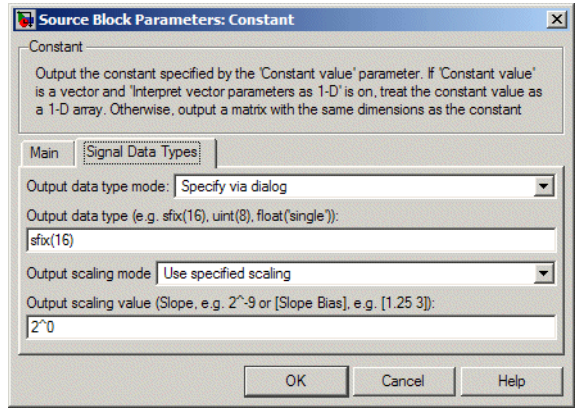
Interpret vector parameters as 1-D

If you select this check box, the Constant block outputs a vector of length `N` if the **Constant value** parameter evaluates to an `N`-element row or column vector, i.e., a matrix of dimension `1xN` or `Nx1`.

Sample time

Specify the interval between times that the Constant block's output can change during simulation (e.g., as a result of tuning its **Constant value** parameter). The default sample time is `inf`, i.e., the block's output can never change. This setting speeds simulation and generated code by avoiding the need to recompute the block's output. See "Specifying Sample Time" in Using Simulink for more information.

The **Signal data types** pane of the Constant block dialog appears as follows:



Output data type mode

Specify how the data type of the output is designated. The data type can be inherited through backpropagation, or can be designated in the **Constant value** parameter, for example `int8(29)`. You can also choose a built-in data type from the list. If you choose `Specify via dialog`, the following parameters become visible.

Output data type

Specify any data type, including fixed-point data types. This parameter is only visible you select `Specify via dialog` for the **Output data type mode** parameter.

Output scaling mode

Specify how the scaling of the output is designated. The output can be automatically scaled to maintain best vector-wise precision without overflow, or you can choose to specify the scaling in the dialog via the **Output scaling value** parameter. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Output scaling value

Set the output scaling using binary point-only or [Slope Bias] scaling. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter and `Use specified scaling` for the **Output Scaling Mode** parameter.

Characteristics

Direct Feedthrough	N/A
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

Cosine

Purpose

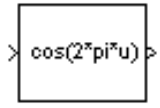
Implement a cosine function in fixed-point using a lookup table approach that exploits quarter wave symmetry

Library

Lookup Tables

Description

The Cosine block is an implementation of the Sine and Cosine block.



Purpose

Model discontinuity at zero, with linear gain elsewhere

Library

Discontinuities

Description

The Coulomb and Viscous Friction block models Coulomb (static) and viscous (dynamic) friction. The block models a discontinuity at zero and a linear gain otherwise. The offset corresponds to the Coulombic friction; the gain corresponds to the viscous friction. The block is implemented as

```
y = sign(u) * (Gain * abs(u) + Offset)
```

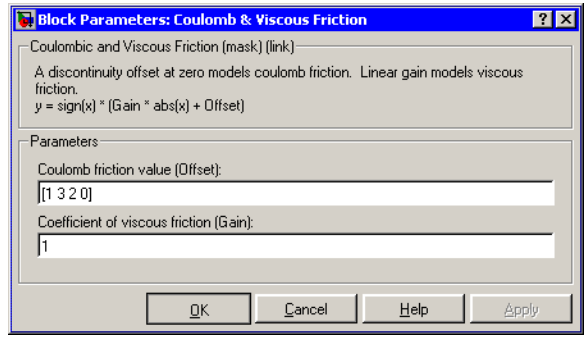
where `y` is the output, `u` is the input, and `Gain` and `Offset` are block parameters.

The block accepts one input and generates one output.

Data Type Support

The Coulomb and Viscous Friction block accepts and outputs real signals of type `double`.

Parameters and Dialog Box



Coulomb friction value

The offset, applied to all input values. The default is `[1 3 2 0]`.

Coefficient of viscous friction

The signal gain at nonzero input points. The default is `1`.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	Yes, at the point where the static friction is overcome

Purpose

Count up and overflow back to zero after the maximum value possible is reached for the specified number of bits

Library

Sources

Description

The Counter Free-Running block counts up until the maximum possible value, $2^{Nbits} - 1$, is reached, where Nbits is the number of bits. Then the counter overflows to zero, and restarts counting up. The counter is always initialized to zero.

You can specify the number of bits with the **Number of Bits** parameter.

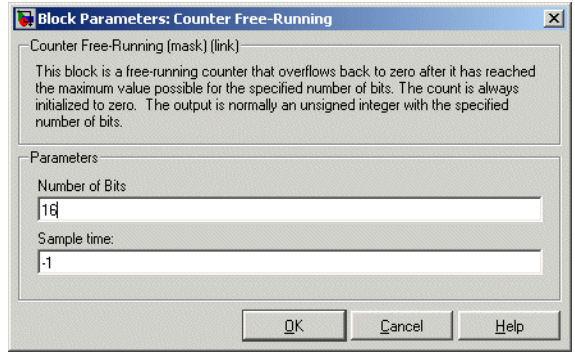
You can specify the sample time with the **Sample time** parameter.

The output is an unsigned integer. If you select the global doubles override, the Counter Free-Running block does not wrap back to zero.

Data Type Support

The Counter Free-Running block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Number of Bits

Specified number of bits.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See “Specifying Sample Time” in Using Simulink for more information.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No

See Also

Counter Limited

Purpose

Count up and wrap back to zero after outputting the specified upper limit

Library

Sources

Description

The Counter Limited block counts up until the specified upper limit is reached. Then the counter wraps back to zero, and restarts counting up. The counter is always initialized to zero.

You can specify the upper limit with the **Upper limit** parameter.

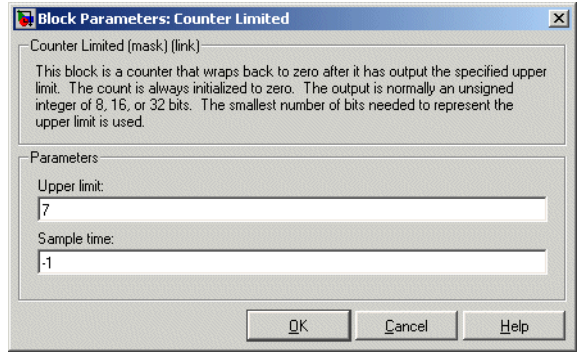
You can specify the sample time with the **Sample time** parameter. A **Sample time** of `-1` means that the sample time is inherited.

The output is an unsigned integer of 8, 16, or 32 bits, with the smallest number of bits needed to represent the upper limit.

Data Type Support

The Counter Limited block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Upper limit
Upper limit.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See “Specifying Sample Time” in Using Simulink for more information.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No

See Also

Counter Free-Running

Purpose

Define a data store

Library

Signal Routing

Description

The Data Store Memory block defines and initializes a named shared data store, which is a memory region usable by Data Store Read and Data Store Write blocks with the same data store name.

The location of the Data Store Memory block that defines a data store determines the Data Store Read and Data Store Write blocks that can access the data store:

- If the Data Store Memory block is in the *top-level system*, the data store can be accessed by Data Store Read and Data Store Write blocks located anywhere in the model.
- If the Data Store Memory block is in a *subsystem*, the data store can be accessed by Data Store Read and Data Store Write blocks located in the same subsystem or in any subsystem below it in the model hierarchy.

Note You can use signal objects in addition to or instead of Data Store Memory blocks to define data stores. See "Working with Data Stores" for more information.

You initialize the data store by specifying a scalar value or an array of values in the **Initial value** parameter. The dimensions of the array determine the dimensionality of the data store. Any data written to the data store must have the dimensions designated by the **Initial value** parameter. Otherwise, an error occurs.

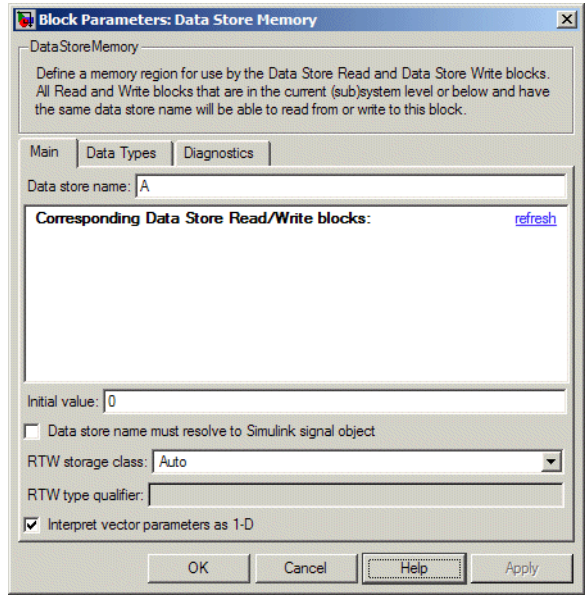
Data Type Support

The Data Store Memory block stores real or complex signals of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see "Data Types Supported by Simulink" in Using Simulink.

Parameters and Dialog Box

The **Main** pane of the Data Store Memory block dialog appears as follows:



Data store name

Specify a name for the data store you are defining with this block. Data Store Read and Data Store Write blocks with the same name will be able to read from and write to the data store initialized by this block.

Corresponding Data Store Read blocks

This parameter lists all the Data Store Read and Data Store Write blocks that have the same data store name as the current block, and that are in the current (sub)system or in any subsystem below it in the model hierarchy. Double-click any entry on this list to highlight the block and bring it to the foreground.

Initial value
Specify the initial value or values of the data store. The dimensions of this value determine the dimensions of data that may be written to the data store.

Data store must resolve to Simulink signal object
Causes Simulink, when compiling the model, to search the model and base workspace for a `Simulink.Signal` object having the same name. If such an object is not found, Simulink halts the compilation and displays an error. Otherwise Simulink compares the attributes of the signal object with the corresponding attributes of the data store memory block. If the block and the object attributes are inconsistent, Simulink halts model compilation and displays an error.

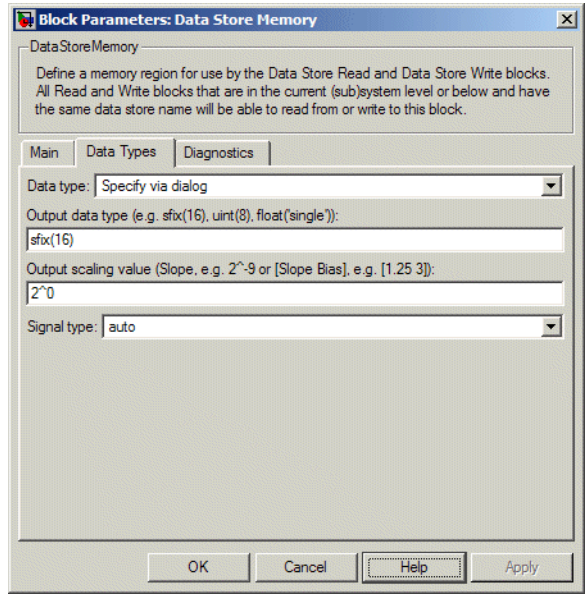
These following parameters pertain to code generation and have no effect during model simulation:

- **Data store name must resolve to Simulink signal object**
- **RTW storage class**
- **RTW type qualifier**

See "Block States: Storing and Interfacing" in the Real-Time Workshop® documentation for more information.

Interpret vector parameters as 1-D
If selected and the **Initial value** parameter is specified as a column or row matrix, the data store is initialized to a 1-D array whose elements are equal to the elements of the row or column vector.

The **Data types** pane of the Data Store Memory block dialog appears as follows:



Data type

Select the data type of the values stored in the data store from the drop-down menu. If you select `auto`, Simulink sets the data type of the data store to the data type of the data store read and write blocks that access it. If you select `Specify via dialog`, the dialog box displays the **Output data type** and **Output scaling value** fields, which enable you to specify fixed-point and other data types not listed in the drop-down menu.

Output data type

Specify any data type for the data store, including fixed-point data types. This parameter is only visible if you select `Specify via dialog` for the **Data type** parameter.

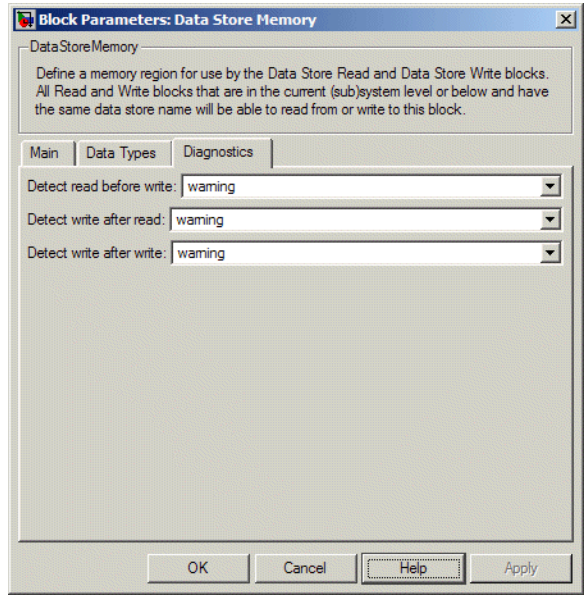
Output scaling value

Set the output scaling using either binary point-only or [Slope Bias] scaling. This parameter is only visible if you select `Specify via dialog` for the **Data type** parameter.

Signal type

Specify the numeric type, real or complex, of the values stored in the data store.

The **Diagnostics** pane of the Data Store Memory block dialog appears as follows:



Detect read before write

The model is attempting to read data from this data store without having previously written data into the store in the current time step.

Detect write after write

The model is attempting to store data in this data store twice in succession in the current time step.

Detect write after read
The model is attempting to store data in this data store after previously reading data from it in the current time step.

Characteristics

Sample Time	N/A
Dimensionalized	Yes

See Also

Data Store Read, Data Store Write

Purpose

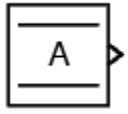
Read data from a data store

Library

Signal Routing

Description

The Data Store Read block copies data from the named data store to its output.



The data store from which the data is read is determined by the location of the Data Store Memory block or signal object that defines the data store. For more information, see "Working with Data Stores"and `Data Store Memory`.

More than one Data Store Read block can read from the same data store.

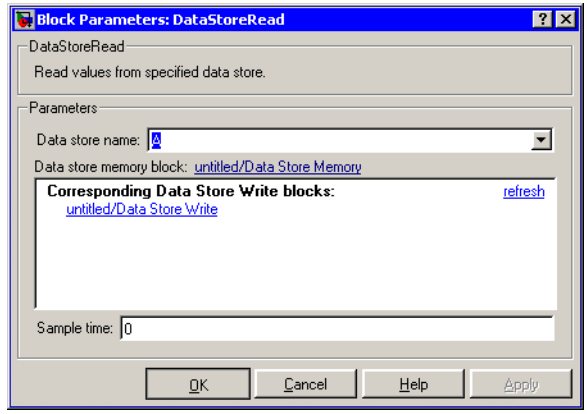
Note Be careful when setting an execution priority on a Data Store Read block. Make sure that the block reads from the data store after the store is updated by any Data Store Write blocks that write to the store in the same time step.

Data Type Support

The Data Store Read block can output a real or complex signal of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see "Data Types Supported by Simulink" in Using Simulink.

Parameters and Dialog Box



Data store name

Specifies the name of the data store from which this block reads data. The adjacent pull-down list lists the names of Data Store Memory blocks that exist at the same level in the model as the Data Store Read block or at higher levels. To change the name, select a name from the pull-down list or enter the name directly in the edit field.

When Simulink compiles the model containing this block, Simulink searches the model upwards from this block's level for a Data Store Memory block having the specified data store name. If Simulink does not find such a block, it searches the model workspace and the MATLAB workspace for a `Simulink.Signal` object having the same name. If Simulink finds the signal object, it creates a hidden Data Store Memory block at the model's root level having the properties specified by the signal object and an initial value of `0`. If Simulink finds neither the Data Store Memory block nor the signal object, it halts the compilation and displays an error.

Data store memory block

This field lists the Data Store Memory block that initialized the store from which this block reads.

Data store write blocks

This parameter lists all the Data Store Write blocks with the same data store name as this block that are in the same (sub)system or in any subsystem below it in the model hierarchy. Double-click any entry on this list to highlight the block and bring it to the foreground.

Sample time

The sample time, which controls when the block reads from the data store. A value of `-1` indicates that the sample time is inherited. See Specifying Sample Time in the online documentation for more information.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Dimensionalized	Yes

See Also

Data Store Memory, Data Store Write

Data Store Write

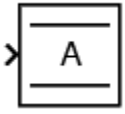
Purpose

Write data to a data store

Library

Signal Routing

Description



The Data Store Write block copies the value at its input to the named data store.

Each write operation performed by a Data Store Write block writes over the data store, replacing the previous contents.

The data store to which this block writes is determined by the location of the Data Store Memory or signal object that defines the data store. For more information, see "Working with Data Stores" and `Data Store Memory`. The size of the data store is set by the signal object or the Data Store Memory block that defines and initializes the data store. Each Data Store Write block that writes to that data store must write the same amount of data.

More than one Data Store Write block can write to the same data store. However, if two Data Store Write blocks attempt to write to the same data store during the same simulation step, results are unpredictable.

Data Type Support

The Data Store Write block accepts a real or complex signal of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see "Data Types Supported by Simulink" in Using Simulink.

Parameters and Dialog Box

Data store name

Specifies the name of the data store to which this block writes data. The adjacent pull-down list lists the names of Data Store Memory blocks that exist at the same level in the model as the Data Store Write block or at higher levels. To change the name, select a name from the pull-down list or enter the name directly in the edit field.

When Simulink compiles the model containing this block, Simulink searches the model upwards from this block's level for a

Data Store Memory block having the specified data store name. If Simulink does not find such a block, it searches the model workspace and the MATLAB workspace for a `Simulink.Signal` object having the same name. If Simulink finds the signal object, it creates a hidden Data Store Memory block at the model's root level having the properties specified by the signal object and an initial value of `0`. If Simulink finds neither the Data Store Memory block nor the signal object, it halts the compilation and displays an error.

Data store memory block
This field lists the Data Store Memory block that initialized the store to which this block writes.

Data store read blocks
This parameter lists all the Data Store Read blocks with the same data store name as this block that are in the same (sub)system or in any subsystem below it in the model hierarchy. Double-click any entry on this list to highlight the block and bring it to the foreground.

Sample time
Specify the sample time that controls when the block writes to the data store. A value of `-1` indicates that the sample time is inherited. See Specifying Sample Time in the online documentation for more information.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Dimensionalized	Yes

See Also

Data Store Memory, Data Store Read

Data Type Conversion

Purpose

Convert an input signal to a specified data type

Library

Signal Attributes

Description

The Data Type Conversion block converts an input signal of any Simulink data type to the data type and scaling specified by the block's **Output data type mode**, **Output data type**, and/or **Output scaling** parameters. The input can be any real- or complex-valued signal. If the input is real, the output is real. If the input is complex, the output is complex.

Note This block requires that you specify the data type and/or scaling for the conversion. If you want to inherit this information from an input signal, you should use the Data Type Conversion Inherited block.

The **Input and output to have equal** parameter controls how the input is processed. The possible values are `Real World Value (RWV)` and `Stored Integer (SI)`:

- Select `Real World Value (RWV)` to treat the input as $V = SQ + B$ where S is the slope and B is the bias. V is used to produce $Q = (V - B)/S$, which is stored in the output. This is the default value.
- Select `Stored Integer (SI)` to treat the input as a stored integer, Q. The value of Q is directly used to produce the output. In this mode, the input and output are identical except that the input is a raw integer lacking proper scaling information. Selecting `Stored Integer` may be useful in these circumstances:
  - If you are generating code for a fixed-point processor, the resulting code only uses integers and does not use floating-point operations.
  - If you want to partition your model based on hardware characteristics. For example, part of your model may involve simulating hardware that produces integers as output.

Working with Fixed-Point Values Greater than 32 Bits

The MATLAB built-in integer data types are limited to 32 bits. If you want to output fixed-point numbers that range between 33 and 53 bits without loss of precision or range, you should break the number into pieces using the Gain block, and then output the pieces using the Data Type Conversion block to store the value inside a `double`.

For example, suppose the original signal is an unsigned 128-bit value with default scaling. You can break this signal into four pieces using four parallel Gain blocks configured with the gain and output settings shown below.

Piece	Gain	Output Data Type
1	`2^0`	`uint(32)` - Least significant 32 bits
2	`2^-32`	`uint(32)`
3	`2^-64`	`uint(32)`
4	`2^-96`	`uint(32)` - Most significant 32 bits

For each Gain block, you must also configure the **Round integer calculations toward** parameter to `Floor`, and the **Saturate on integer overflow** check box must be cleared.

Data Type Support

The Data Type Conversion block handles any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in Using Simulink.

Parameters and Dialog Box

Output data type mode

You can set the output signal to a built-in data type from this drop-down list, or you can choose to inherit the output data type and scaling by backpropagation. Lastly, if you choose `Specify via dialog`, the **Output data type**, **Output scaling value**, and

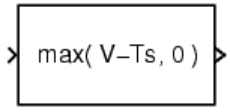
Purpose

Decrease the real-world value of the signal by the sample time, but only to zero.

Library

Additional Math & Discrete / Additional Math: Increment - Decrement

Description

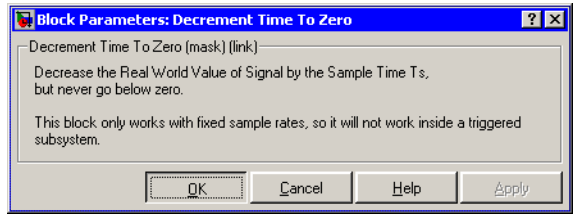


The Decrement Time To Zero block decreases the real-world value of the signal by the sample time, Ts. The output will never go below zero. This block only works with fixed sample rates.

Data Type Support

The Decrement Time To Zero block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Characteristics

Direct Feedthrough	Yes
Scalar Expansion	No

See Also

Decrement Real World, Decrement Stored Integer, Decrement To Zero

Decrement To Zero

Purpose

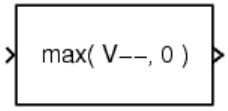
Decreases the real-world value of a signal by one, but only to zero.

Library

Additional Math & Discrete / Additional Math: Increment - Decrement

Description

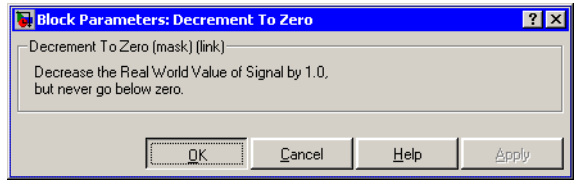
The Decrement To Zero block decreases the real-world value of the signal by one. The output will never go below zero.



Data Type Support

The Decrement To Zero block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Characteristics

Direct Feedthrough	Yes
Scalar Expansion	No

See Also

Decrement Real World, Decrement Stored Integer, Decrement Time To Zero

Purpose

Extract and output the elements of a bus or vector signal

Library

Signal Routing

Description

The Demux block extracts the components of an input signal and outputs the components as separate signals. The block accepts either vector (1-D array) signals or bus signals (see “Signal Buses” in Using Simulink for more information). The **Number of outputs** parameter allows you to specify the number and, optionally, the dimensionality of each output port. If you do not specify the dimensionality of the outputs, the block determines the dimensionality of the outputs for you.

The Demux block operates in either vector or bus selection mode, depending on whether you selected the **Bus selection mode** parameter. The two modes differ in the types of signals they accept. Vector mode accepts only a vector-like signal, that is, either a scalar (one-element array), vector (1-D array), or a column or row vector (one row or one column 2-D array). Bus selection mode accepts only the output of a Mux block or another Demux block.

The Demux block’s **Number of outputs** parameter determines the number and dimensionality of the block’s outputs, depending on the mode in which the block operates.

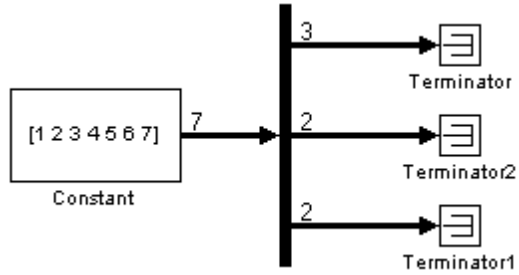
Specifying the Number of Outputs in Vector Mode

In vector mode, the value of the parameter can be a scalar specifying the number of outputs or a vector whose elements specify the widths of the block’s output ports. The block determines the size of its outputs from the size of the input signal and the value of the **Number of outputs** parameter.

The following table summarizes how the block determines the outputs for an input vector of width `n`.

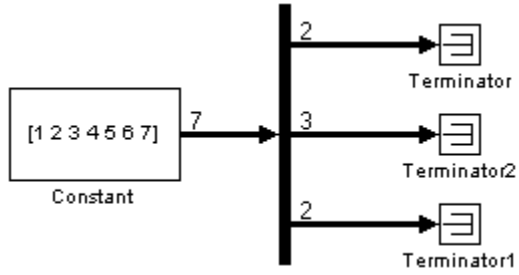
Parameter Value	Block outputs...	Comments
`p = n`	`p` scalar signals	For example, if the input is a three-element vector and you specify three outputs, the block outputs three scalar signals.
`p > n`	Error	
`p < n` `n mod p = 0`	`p` vector signals each having `n/p` elements	If the input is a six-element vector and you specify three outputs, the block outputs three two-element vectors.
`p < n` `n mod p = m`	`m` vector signals each having `(n/p)+1` elements and `p-m` signals having `n/p` elements	If the input is a five-element vector and you specify three outputs, the block outputs two two-element vector signals and one scalar signal.
$[p_1\ p_2\ \ldots\ p_m]$ $p_1+p_2+\ldots+p_m=n$ $p_i > 0$	`m` vector signals having widths p_1, p_2, ... p_m	If the input is a five-element vector and you specify `[3, 2]` as the output, the block outputs three of the input elements on one port and the other two elements on the other port.
$[p_1\ p_2\ \ldots\ p_m]$ $p_1+p_2+\ldots+p_m=n$ some or all $p_i = -1$	`m` vector signals	If `pi` is greater than zero, the corresponding output has width p_i. If p_i is -1, the width of the corresponding output is dynamically sized.
$[p_1\ p_2\ \ldots\ p_m]$ $p_1+p_2+\ldots+p_m!=n$ $p_i = > 0$	Error	

Note that you can specify the number of outputs as fewer than the number of input elements, in which case the block distributes the elements as evenly as possible over the outputs as illustrated in the following example.



You can use -1 in a vector expression to indicate that the block should dynamically size the corresponding port. For example, the expression `[-1, 3 -1]` causes the block to output three signals in which the second signal always has three elements while the sizes of the first and third signals depend on the size of the input signal.

If a vector expression comprises positive values and -1 values, the block assigns as many elements as needed to the ports with positive values and distributes the remain elements as evenly as possible over the ports with -1 values. For example, suppose that the block input is seven elements wide and you specify the output as `[-1, 3 -1]`. In this case, the block outputs two elements on the first port, three elements on the second, and two elements on the third.

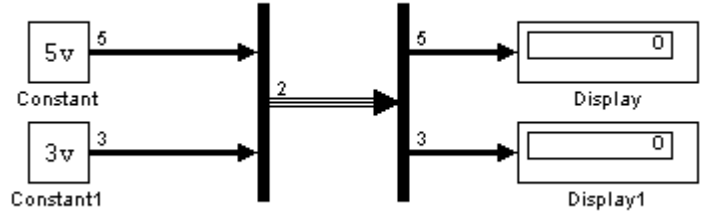


Specifying the Number of Outputs in Bus Selection Mode

In bus selection mode, the value of the **Number of outputs** parameter can be a

- Scalar specifying the number of output ports

  The specified value must equal the number of input signals. For example, if the input bus comprises two signals and the value of this parameter is a scalar, the value must equal `2`.

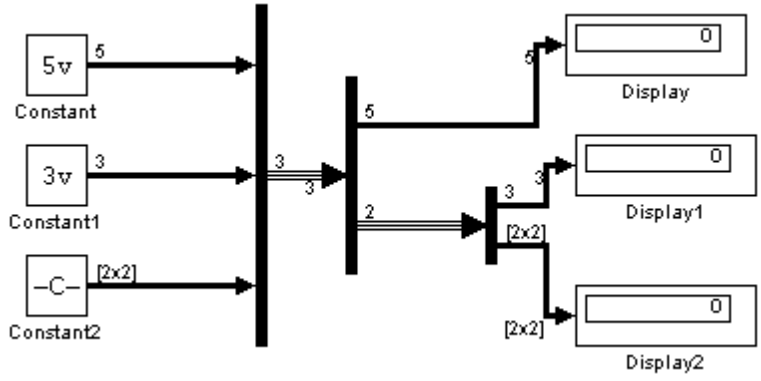


- Vector each of whose elements specifies the number of signals to output on the corresponding port

  For example, if the input bus contains five signals, you can specify the output as `[3, 2]`, in which case the block outputs three of the input signals on one port and the other two signals on a second port.

- Cell array each of whose elements is a cell array of vectors specifying the dimensions of the signals output by the corresponding port

The cell array format constrains the Demux block to accept only signals of specified dimensions. For example, the cell array {{[2 2], 3} {1}} tells the block to accept only a bus signal comprising a 2-by-2 matrix, a three-element vector, and a scalar signal. You can use the value -1 in a cell array expression to let the block determine the dimensionality of a particular output based on the input. For example, the following diagram uses the cell array expression {{-1}, {-1,-1}} to specify the output of the leftmost Demux block.



In bus selection mode, if you specify the dimensionality of an output port, i.e., if you specify any value other than -1, the corresponding input element must match the specified dimensionality.

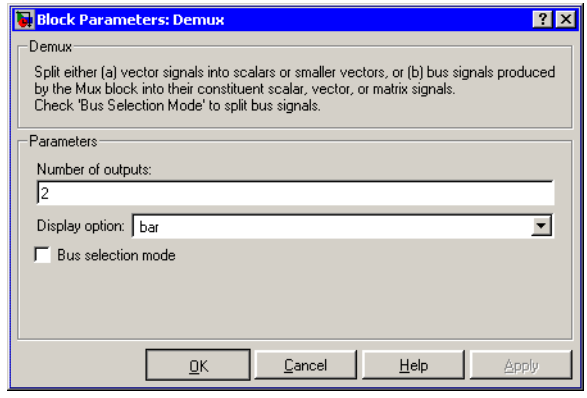
Note Simulink hides the name of a Demux block when you copy it from the Simulink library to a model.

Data Type Support

The Demux block accepts and outputs complex or real signals of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in Using Simulink.

Parameters and Dialog Box



Number of outputs

The number and dimensions of outputs.

Display option

Options for displaying the Demux block. The options are

Option	Description	Example
bar	Display the icon as a solid bar of the block's foreground color.	
none	Display the icon as a box containing the block's type name.	

Bus selection mode

Enable bus selection mode.

Derivative

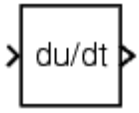
Purpose

Output the time derivative of the input

Library

Continuous

Description



The Derivative block approximates the derivative of its input by computing

$$\frac{du}{dt}$$

where *du* is the change in input value and *dt* is the change in time since the previous simulation time step. The block accepts one input and generates one output. The initial output for the block is zero.

The accuracy of the results depends on the size of the time steps taken in the simulation. Smaller steps allow a smoother and more accurate output curve from this block. Unlike blocks that have continuous states, the solver does not take smaller steps when the input changes rapidly.

When the input is a discrete signal, the continuous derivative of the input is an impulse when the value of the input changes, otherwise it is 0. You can obtain the discrete derivative of a discrete signal using

$$y(k) = \frac{1}{\Delta t}(u(k) - u(k-1))$$

and taking the *z*-transform

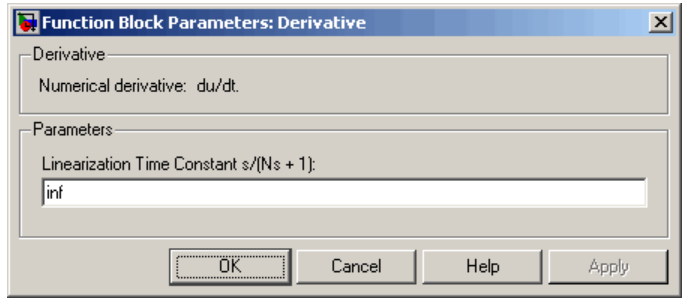
$$\frac{Y(z)}{u(z)} = \frac{1 - z^{-1}}{\Delta t} = \frac{z - 1}{\Delta t \cdot z}$$

Using `linmod` to linearize a model that contains a Derivative block can be troublesome. To improve the accuracy of linearizations of this block, use the optional linearization parameter within the block dialog box. Additionally, for more information about how to avoid problems linearizing Derivative blocks, see Linearizing Models in Using Simulink.

Data Type Support

The Derivative block accepts and outputs a real signal of type `double`.

Parameters and Dialog Box



The exact linearization of the Derivative block is difficult due to the fact that the block cannot be represented as a state space system since the dynamic equation for the block is $y = \dot{u}$. However, it is possible to approximate the linearization by adding a pole to the Derivative to create a proper transfer function. The addition of the pole has the effect of filtering the signal before differentiating it, to remove the effect of noise. The approximated linearization of the Derivative block is then $\frac{s}{Ns+1}$. You can change the **Linearization Time Constant**, N, to more accurately approximate the linearization for your system. Its default value is `Inf`, corresponding to a linearization of 0, but it is common practice to change it to $\frac{1}{f_b}$, where f_b is the break frequency for the filter.

Characteristics

Direct Feedthrough	Yes
Sample Time	Continuous
Scalar Expansion	N/A
States	2*[1+(number of input elements)]
Dimensionalized	Yes
Zero Crossing	No

See Also

Discrete Derivative

Detect Change

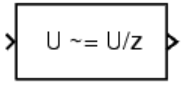
Purpose

Detect a change in a signal's value

Library

Logic and Bit Operations

Description



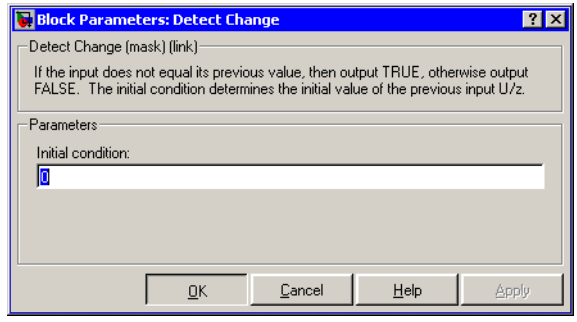
The Detect Change block determines if an input does not equal its previous value where

- The output is true (not 0), when the input signal does not equal its previous value.
- The output is false (equal to 0), when the input signal equals its previous value.

Data Type Support

The Detect Change block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Initial condition

Set the initial condition for the previous input U/z.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also

Detect Decrease, Detect Fall Negative, Detect Fall Nonpositive, Detect Increase, Detect Rise Nonnegative, Detect Rise Positive

Detect Decrease

Purpose

Detect a decrease in a signal's value

Library

Logic and Bit Operations

Description

The Detect Decrease block determines if an input is

Purpose

Detect a rising edge when a signal's value increases to a nonnegative value, and its previous value was strictly negative

Library

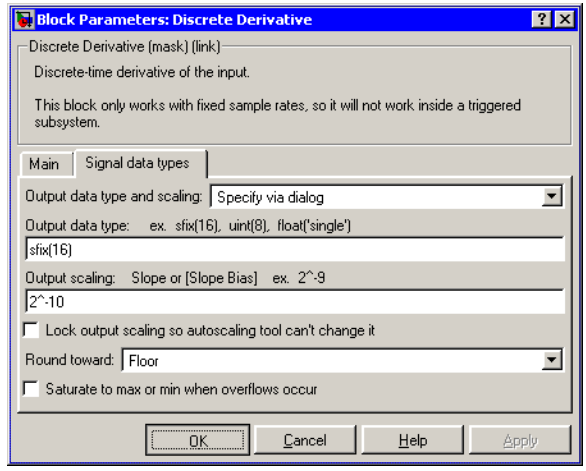
Logic and Bit Operations

Description

The Detect Rise Nonnegative block determines if the input is greater than or equal to zero, and its previous value was less than zero where

- The output isL0-300.3(a)-311.2(at).3s2ie

The **Signal data types** pane of the Discrete Derivative block dialog box appears as follows:



Output data type and scaling

Specify the output data type and scaling via the dialog box, or inherit the data type and scaling from the driving block or by backpropagation. If you choose `Specify via dialog`, the **Output data type** and **Output scaling** parameters appear.

Output data type

Set the output data type. This parameter is only visible if you select `Specify via dialog` for the **Output data type and scaling** parameter.

Output scaling

Set the output scaling using either binary point-only or [Slope Bias] scaling. This parameter is only visible if you select `Specify via dialog` for the **Output data type and scaling** parameter.

Lock output scaling so autoscaling tool can't change it

If you select this check box, the output scaling is locked.

Round toward
Select the rounding mode for fixed-point operations.

Saturate to max or min when overflows occur
If selected, fixed-point overflows saturate.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of inputs and gain

See Also

Derivative

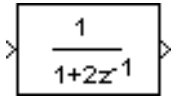
Purpose

Implement IIR and FIR filters

Library

Discrete

Description



The Discrete Filter block implements Infinite Impulse Response (IIR) and Finite Impulse Response (FIR) filters. You specify the filter as a ratio of polynomials in z^{-1}. You can specify that the block have a single output or multiple outputs where the outputs correspond to a set of filters that have the same denominator polynomial but different numerator polynomials.

Use the **Numerator coefficients** parameter to specify the coefficients of the discrete filter's numerator polynomial or polynomials. Use a vector to specify the coefficients for a single numerator polynomial. Use a matrix to specify the coefficients of multiple numerator polynomials where each row contains the coefficients of one of the polynomials. Use the **Denominator coefficients** parameter to specify the coefficients of the function's denominator polynomial. The value of the **Denominator coefficients** parameter must be a vector of coefficients.

You must specify the coefficients of the numerator and denominator polynomials in ascending powers of z^{-1}. The order of the denominator must be greater than or equal to the order of the numerator.

If you specify a single numerator polynomial, i.e., a vector as the value of the **Numerator coefficients** parameter, the block's output is a scalar signal. If you specify multiple numerator polynomials, i.e., a matrix as the value of the **Numerator coefficients** parameter, the block's output is a vector signal whose width equals the number of matrix rows, i.e., the number or numerator polynomials.

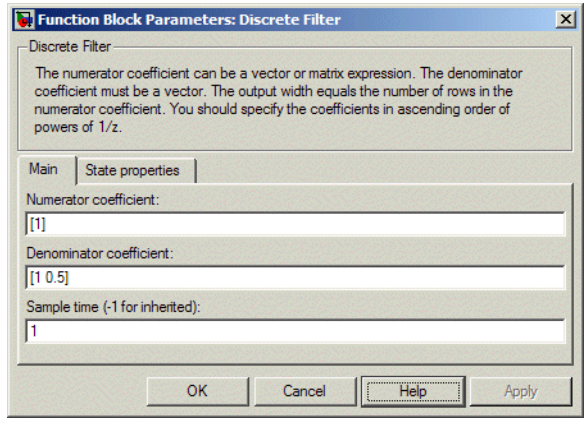
The Discrete Filter block lets you use polynomials in z^{-1} (the delay operator) to represent a discrete system, a method typically used by signal processing engineers. By contrast, the Discrete Transfer Fcn block lets you use polynomials in z to represent a discrete system, the method typically used by control engineers. The two methods are identical when the numerator and denominator polynomials have the same length.

The block displays the numerator and denominator according to how they are specified. For a discussion of how Simulink displays the icon, see `Transfer Fcn`.

Data Type Support

The Discrete Filter block accepts and outputs a real signal of type `double`.

Parameters and Dialog Box



Numerator coefficient

A vector of polynomial coefficients or a matrix of coefficients where each row of coefficients corresponds to a distinct numerator polynomial. You must specify the polynomial coefficients in ascending powers of z^{-1}. If you specify a vector of coefficients, i.e., a single numerator polynomial, the output of the block is a scalar signal. If you specify a matrix of coefficients, i.e., multiple polynomials, the block's output is a vector of signals, each corresponding to the filter consisting off the corresponding numerator polynomial and the denominator polynomial specified by the**Denominator coefficients** parameter. The default is `[1]`.

Denominator coefficient

The vector of denominator coefficients. The default is `[1 2]`. The width of the vector, i.e., the order of the denominator, must

be greater than or equal to the width of the numerator vector or matrix rows, i.e., the order of the numerator.

Sample time
The time interval between samples. See Specifying Sample Time in Using Simulink for more information.

The **State properties** pane of this block pertains to code generation and has no effect on model simulation. See "Block States: Storing and Interfacing" in the Real-Time Workshop User's Guide for more information.

Characteristics

Direct Feedthrough	Only if the lengths of the **Numerator** and **Denominator** parameters are equal
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
States	Length of **Denominator** parameter -1
Dimensionalized	No
Zero Crossing	No

Discrete State-Space

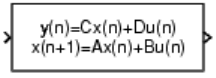
Purpose

Implement a discrete state-space system

Library

Discrete

Description



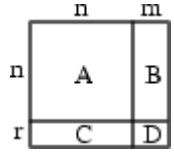
The Discrete State-Space block implements the system described by

$$x(n+1) = Ax(n) + Bu(n)$$

$$y(n) = Cx(n) + Du(n)$$

where u is the input, x is the state, and y is the output. The matrix coefficients must have these characteristics, as illustrated in the following diagram:

- **A** must be an n-by-n matrix, where n is the number of states.
- **B** must be an n-by-m matrix, where m is the number of inputs.
- **C** must be an r-by-n matrix, where r is the number of outputs.
- **D** must be an r-by-m matrix.



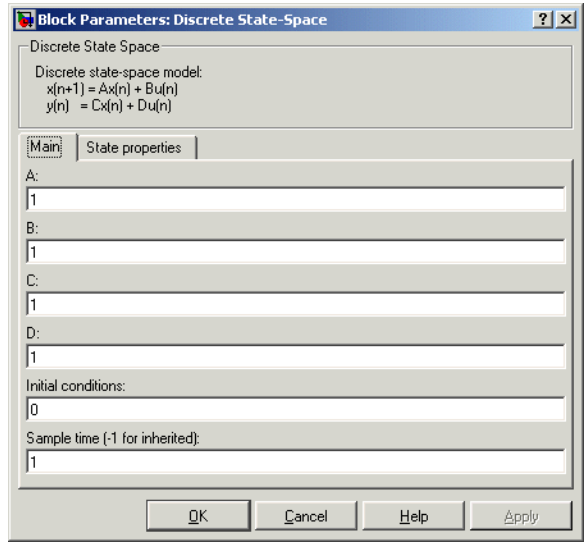
The block accepts one input and generates one output. The input vector width is determined by the number of columns in the B and D matrices. The output vector width is determined by the number of rows in the C and D matrices.

Simulink converts a matrix containing zeros to a sparse matrix for efficient multiplication.

Data Type Support

The Discrete State Space block accepts and outputs a real signal of type `double`.

Parameters and Dialog Box



A, B, C, D
The matrix coefficients, as defined in the preceding equations.

Initial conditions
The initial state vector. The default is `0`. Simulink does not allow the initial states of this block to be `inf` or `NaN`.

Sample time
The time interval between samples. See Specifying Sample Time in Using Simulink for more information.

The **State properties** pane of this block pertains to code generation and has no effect on model simulation. See "Block States: Storing and Interfacing" in the Real-Time Workshop User's Guide for more information.

Characteristics

Direct Feedthrough	Only if $D \neq 0$
Sample Time	Specified in the **Sample time** parameter

Scalar Expansion	Yes, of the initial conditions
States	Determined by the size of *A*
Dimensionalized	Yes
Zero Crossing	No

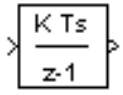
Purpose

Perform discrete-time integration or accumulation of a signal

Library

Discrete

Description



You can use the Discrete-Time Integrator block in place of the Integrator block to create a purely discrete system.

The Discrete-Time Integrator block allows you to

- Define initial conditions on the block dialog box or as input to the block.
- Define an input gain (K) value.
- Output the block state.
- Define upper and lower limits on the integral.
- Reset the state depending on an additional reset input.

These features are described below.

Integration and Accumulation Methods

The block can integrate or accumulate using the Forward Euler, Backward Euler, and Trapezoidal methods. For a given step `n`, Simulink updates `y(n)` and `x(n+1)`. In integration mode, `T` is the block's sample time (delta `T` in the case of triggered sample time). In accumulation mode, `T = 1`; the block's sample time determines when the block's output is computed but not the output's value. `K` is the gain value. Values are clipped according to upper or lower limits.

- Forward Euler method (the default), also known as Forward Rectangular, or left-hand approximation.

  For this method, `1/s` is approximated by `T/(z-1)`. The resulting expression for the output of the block at step `n` is

  ```
  y(n) = y(n-1) + K*T*u(n-1)
  ```

Let `x(n+1) = x(n) + K*T*u(n)`. The block uses the following steps to compute its output:

```
Step 0:    y(0)   = x(0) = IC (clip if necessary)
   x(1)   = y(0) + K*T*u(0)

Step 1:   y(1)   = x(1)
   x(2)   = x(1) + K*T*u(1)

Step n:   y(n)   = x(n)
   x(n+1) = x(n) + K*T*u(n) (clip if necessary)
```

With this method, input port 1 does not have direct feedthrough.

- Backward Euler method, also known as Backward Rectangular or right-hand approximation.

  For this method, `1/s` is approximated by `T*z/(z-1)`. The resulting expression for the output of the block at step `n` is

  ```
  y(n) = y(n-1) + K*T*u(n)
  ```

  Let `x(n) = y(n-1)`. The block uses the following steps to compute its output

  ```
  Step 0:   y(0)   = x(0) = IC (clipped if necessary)
     x(1)   = y(0)
  ```

  or, depending on **Use initial condition as initial and reset value for** parameter:

  ```
  Step 0:   y(0)   = x(0) = IC (clipped if necessary)
     x(1)   = y(0) = x(0) + K*T*u(0)

  Step 1:   y(1)   = x(1) + K*T*u(1)
     x(2)   = y(1)
  ```

```
Step n:   y(n)   = x(n) + K*T*u(n)
   x(n+1) = y(n)
```

With this method, input port 1 has direct feedthrough.

- Trapezoidal method. For this method, `1/s` is approximated by

  ```
  T/2*(z+1)/(z-1)
  ```

  When `T` is fixed (equal to the sampling period), let

  ```
  x(n) = y(n-1) + K*T/2 * u(n-1)
  ```

  The block uses the following steps to compute its output

  ```
  Step 0:   y(0)   = x(0) = IC (clipped if necessary)
     x(1)   = y(0) + K*T/2 * u(0)
  ```

  or, depending on **Use initial condition as initial and reset value for** parameter:

  ```
  Step 0:   y(0)   = x(0) = IC (clipped if necessary)
     x(1)   = y(0) = x(0) + K*T/2*u(0)

  Step 1:   y(1)   = x(1) + K*T/2 * u(1)
     x(2)   = y(1) + K*T/2 * u(1)

  Step n:   y(n)   = x(n) + K*T/2 * u(n)
     x(n+1) = y(n) + K*T/2 * u(n)
  ```

  Here, `x(n+1)` is the best estimate of the next output. It isn't quite the state, in the sense that `x(n) != y(n)`.

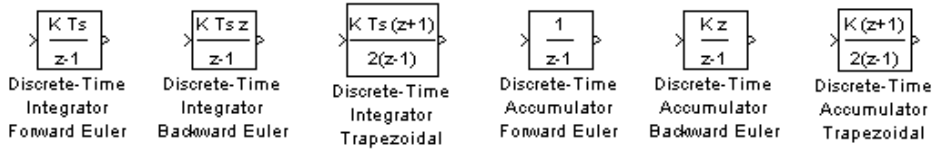
  If `T` is variable (i.e. obtained from the triggering times), the block uses the following algorithm to compute its outputs

  ```
  Step 0:   y(0)   = x(0) = IC (clipped if necessary)
     x(1)   = y(0)
  ```

```
or, depending on Use initial condition as initial and
reset value for
parameter:

Step 0:   y(0)   = x(0) = IC (clipped if necessary)
   x(1)   = y(0) = x(0) + K*T/2*u(0)

Step 1:   y(1)   = x(1) + T/2 * (u(1) + u(0))
   x(2)   = y(1)

Step n:   y(n)   = x(n) + T/2 * (u(n) + u(n-1))
   x(n+1) = y(n)
```

With this method, input port 1 has direct feedthrough.

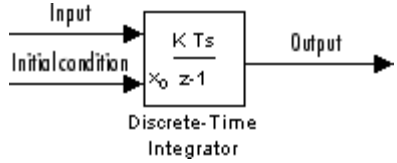
The block reflects the selected integration or accumulation method, as this figure shows.



Defining Initial Conditions

You can define the initial conditions as a parameter on the block dialog box or input them from an external signal:

- To define the initial conditions as a block parameter, specify the **Initial condition source** parameter as `internal` and enter the value in the **Initial condition** parameter field.
- To provide the initial conditions from an external source, specify the **Initial condition source** parameter as `external`. An additional input port appears under the block input, as shown in this figure.



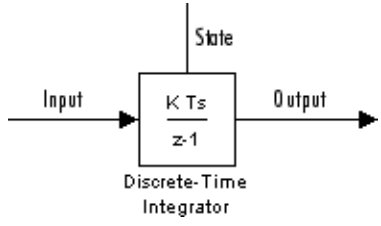
Using the State Port

In two situations, you must use the state port instead of the output port:

- When the output of the block is fed back into the block through the reset port or the initial condition port, causing an algebraic loop. For an example of this situation, see the `bounce` model.
- When you want to pass the state from one conditionally executed subsystem to another, which can cause timing problems. For an example of this situation, see the `clutch` model.

You can correct these problems by passing the state through the state port rather than the output port. Although the values are the same, Simulink generates them at slightly different times, which protects your model from these problems. You output the block state by selecting the **Show state port** check box.

By default, the state port appears on the top of the block, as shown in this figure.



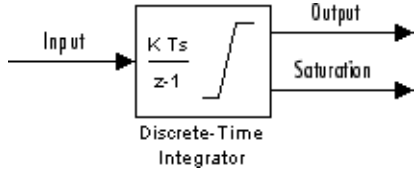
Limiting the Integral

To prevent the output from exceeding specifiable levels, select the **Limit output** check box and enter the limits in the appropriate parameter

fields. Doing so causes the block to function as a limited integrator. When the output reaches the limits, the integral action is turned off to prevent integral wind up. During a simulation, you can change the limits but you cannot change whether the output is limited. The output is determined as follows:

- When the integral is less than or equal to the **Lower saturation limit** and the input is negative, the output is held at the **Lower saturation limit**.
- When the integral is between the **Lower saturation limit** and the **Upper saturation limit**, the output is the integral.
- When the integral is greater than or equal to the **Upper saturation limit** and the input is positive, the output is held at the **Upper saturation limit**.

To generate a signal that indicates when the state is being limited, select the **Show saturation port** check box. A saturation port appears below the block output port, as shown in this figure.



The signal has one of three values:

- 1 indicates that the upper limit is being applied.
- 0 indicates that the integral is not limited.
- -1 indicates that the lower limit is being applied.

Resetting the State

The block can reset its state to the specified initial condition, based on an external signal. To cause the block to reset its state, select one of the

External reset parameter choices. A trigger port appears below the block's input port and indicates the trigger type, as shown in this figure.

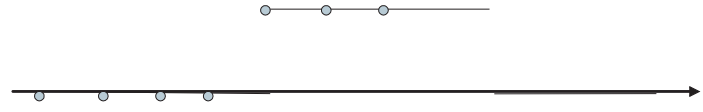
The reset port has direct feedthrough. If the block output is fed back into this port, either directly or through a series of blocks with direct feedthrough, an algebraic loop results. To resolve this loop, feed the output of the block's state port into the reset port instead. To access the block's state, select the **Show state port** check box.

Reset Trigger Types

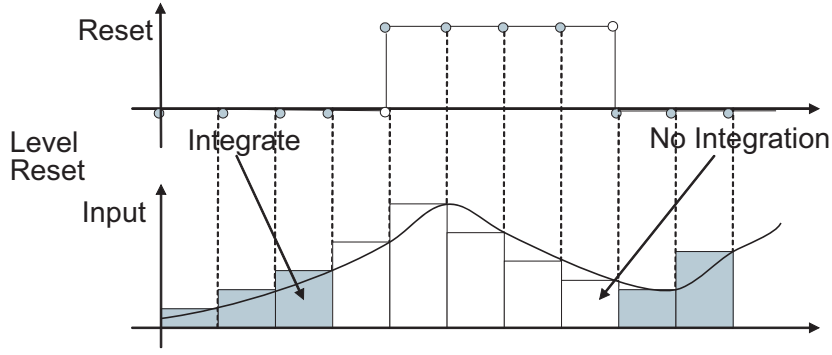
The **External reset** parameter lets you determine the attribute of the reset signal that triggers the reset. The trigger options include:

- `rising`

  Resets the state when the reset signal has a rising edge. For example, the following figure shows the effect that a rising reset trigger has on backward Euler integration.

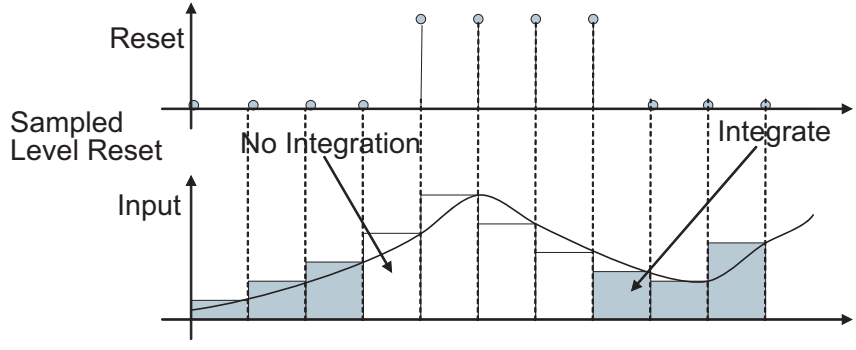


Resets and holds the output to the initial condition while the reset signal is nonzero. For example, the following figure shows the effect that a level reset trigger has on backward Euler integration.



- `sampled level`

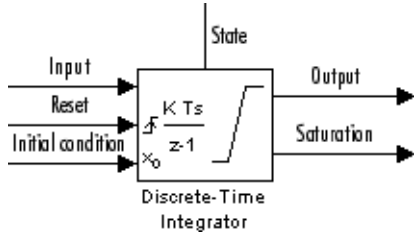
  Resets the output to the initial condition when the reset signal is nonzero. For example, the following figure shows the effect that a sampled level reset trigger has on backward Euler integration.



Note The `sampled level` reset option requires fewer computations and hence is more efficient than the `level` reset option. However, the `level` reset option, but may introduces a discontinuity when integration resumes.

Choosing All Options

When all options are selected, the icon looks like this.

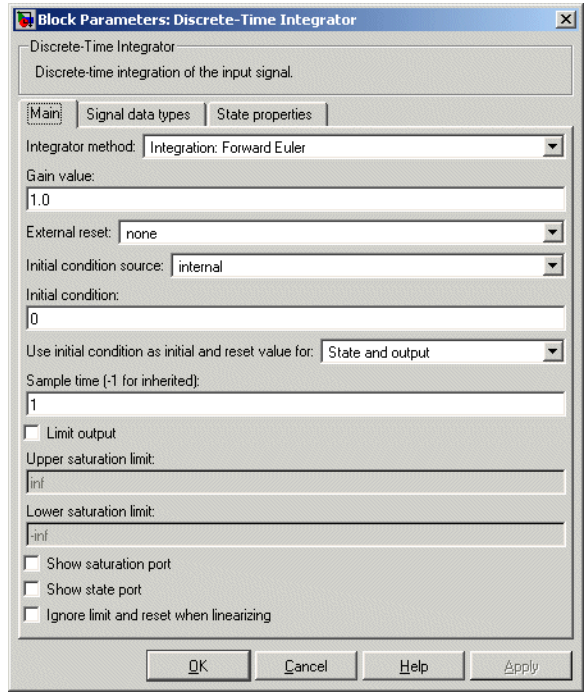


Data Type Support

The Discrete-Time Integrator block accepts real signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box

The **Main** pane of the Discrete-Time Integrator block dialog appears as follows:

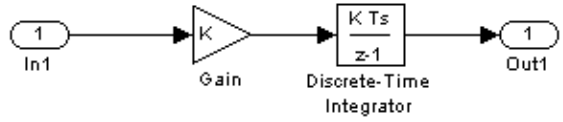


Integrator method

Specify the integration or accumulation method.

Gain value

Specify a value by which to multiply the integrator input. Specifying a value other than 1.0 (the default) is semantically equivalent to connecting a signal to the input of the integrator via a Gain block, i.e., to



Using this parameter to specify the input gain eliminates a multiplication operation in the generated code. Realizing this benefit, however, requires that this parameter be nontunable. Accordingly, the Real-Time Workshop generates a warning during code generation if the **Model Parameter Configuration** dialog box for this model declares this parameter to be tunable. If you want to tune the input gain, set this parameter to 1.0 and use an external Gain block to specify the input gain.

External reset

Resets the states to their initial conditions when a trigger event occurs in the reset signal. See “Resetting the State” on page 2-206 for more information.

Initial condition source

Gets the states' initial conditions from the **Initial condition** parameter (`internal`) or from an external block (`external`). Simulink does not allow the initial condition of this block to be `inf` or `NaN`.

Initial condition

The states' initial conditions. This parameter is only available if the **Initial condition source** parameter is set to `internal`. Simulink does not allow the initial condition of this block to be `inf` or `NaN`.

Use initial condition as initial and reset value for

When you set this parameter to `State and output`,

```
y(0) = IC
```

```
x(0) = IC
```

or at reset

```
y(n) = IC
```

```
x(n) = IC
```

When you set this parameter to `State only (most efficient)`,

```
x(0) = IC
```

or at reset

```
x(n) = IC
```

Sample time
The time interval between samples. The default is `1`. In accumulation mode, the sample time specifies when the block's output is computed. See Specifying Sample Time in Using Simulink for information on how to specify the sample time.

Limit output
If selected, limits the block's output to a value between the **Lower saturation limit** and **Upper saturation limit** parameters.

Upper saturation limit
The upper limit for the integral. This parameter is only available if you select the **Limit output** parameter.

Lower saturation limit
The lower limit for the integral. This parameter is only available if you select the **Limit output** parameter.

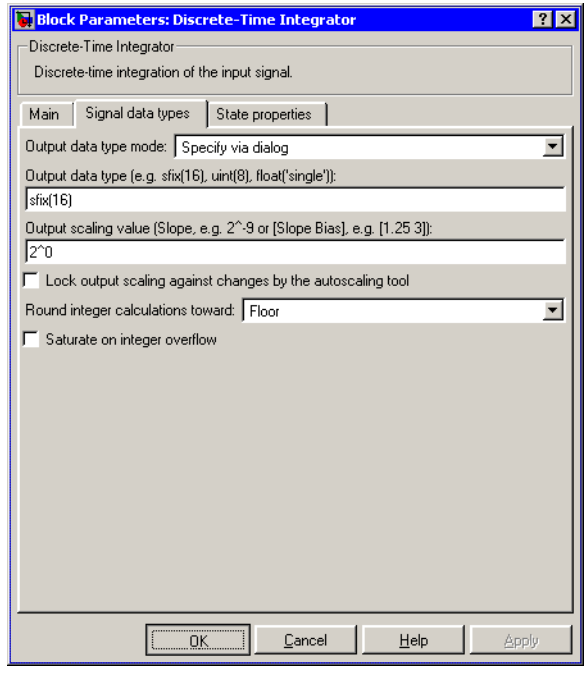
Show saturation port
If selected, adds a saturation output port to the block.

Show state port
If selected, adds an output port to the block for the block's state.

Ignore limit and reset when linearizing

Select this option to cause Simulink linearization commands to treat this block as unresettable and as having no limits on its output, regardless of the settings of the block's reset and output limitation options. This allows you to linearize a model around an operating point that causes the integrator to reset or saturate.

The **Signal data types** pane of the Discrete-Time Integrator block dialog appears as follows:



Output data type mode

Specify the output data type and scaling via the dialog box, or inherit the data type and scaling from the driving block or by backpropagation.

Output data type
Specify any data type, including fixed-point data types. This parameter is only visible if you select `Specify via dialog` for the **Output data type and scaling** parameter.

Output scaling value
Set the output scaling using either binary point-only or [Slope Bias] scaling. This parameter is only visible if you select `Specify via dialog` for the **Output data type and scaling** parameter.

Lock output scaling against changes by the autoscaling tool
Select to lock scaling of outputs. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Round integer calculations toward
Select the rounding mode for fixed-point operations.

Saturate on integer overflow
Select to have overflows saturate.

The **State properties** pane of this block pertains to code generation and has no effect on model simulation. See "Block States: Storing and Interfacing" in the Real-Time Workshop User's Guide for more information.

Characteristics

Direct Feedthrough	Yes, of the reset and external initial condition source ports. The input has direct feedthrough for every integration method except forward Euler and accumulation forward Euler.
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of parameters
States	Inherited from driving block and parameter

Dimensionalized	Yes
Zero Crossing	No

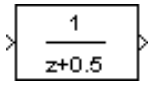
Purpose

Implement a discrete transfer function

Library

Discrete

Description



The Discrete Transfer Fcn block implements the *z*-transform transfer function described by the following equations:

$$H(z) = \frac{num(z)}{den(z)} = \frac{num_0 z^n + num_1 z^{n-1} + \ldots + num_m z^{n-m}}{den_0 z^n + den_1 z^{n-1} + \ldots + den_n}$$

where *m+1* and *n+1* are the number of numerator and denominator coefficients, respectively. *num* and *den* contain the coefficients of the numerator and denominator in descending powers of *z*. *num* can be a vector or matrix, *den* must be a vector, and both are specified as parameters on the block dialog box. The order of the denominator must be greater than or equal to the order of the numerator.

Block input is scalar; output width is equal to the number of rows in the numerator.

The Discrete Transfer Fcn block represents the method typically used by control engineers, representing discrete systems as polynomials in *z*. The Discrete Filter block represents the method typically used by signal processing engineers, who describe digital filters using polynomials in z^{-1} (the delay operator). The two methods are identical when the numerator is the same length as the denominator.

The Discrete Transfer Fcn block displays the numerator and denominator within its icon depending on how they are specified. See `Transfer Fcn` for more information.

Data Type Support

The Discrete Transfer Function block accepts and outputs real signals of type `double`.

Discrete Transfer Fcn

Parameters and Dialog Box

Numerator
The row vector of numerator coefficients. A matrix with multiple rows can be specified to generate multiple output. The default is `[1]`.

Denominator
The row vector of denominator coefficients. The default is `[1 0.5]`.

Sample time
The time interval between samples. The default is `1`. See Specifying Sample Time in Using Simulink for more information.

The **State properties** pane of this block pertains to code generation and has no effect on model simulation. See "Block States: Storing and Interfacing" in the Real-Time Workshop User's Guide for more information.

Characteristics

Direct Feedthrough	Only if the lengths of the **Numerator** and **Denominator** parameters are equal

Lock output scaling against changes by the autoscaling tool
Select to lock scaling of outputs. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Round integer calculations toward
Select the rounding mode for fixed-point operations.

Saturate on integer overflow
Select to have overflows saturate.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	No
States	0
Dimensionalized	Yes
Zero Crossing	No

Embedded MATLAB Function

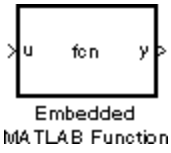
Purpose

Include MATLAB code in models that generate embeddable C code.

Library

User-Defined Functions

Description



An Embedded MATLAB Function block lets you compose a MATLAB function in Simulink like the following example:

The MATLAB function you create executes for simulation and generates code for a Real-Time Workshop target. If you are new to Simulink and MATLAB, see "Using the Embedded MATLAB Function Block" in Simulink documentation for a comprehensive overview including a step-by-step example.

You create the MATLAB function in the **Embedded MATLAB Editor**. To learn about this editor's capabilities see "Using the Embedded MATLAB Editor."

You specify input and output data to the Embedded MATLAB Function block in the function header as arguments and return values. Notice that the argument and return values of the preceding example function correspond to the inputs and outputs of the block in Simulink.

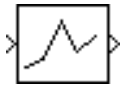
Purpose

Implement a first-order sample-and-hold

Library

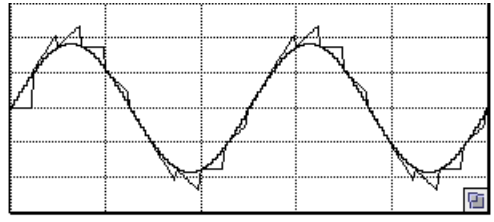
Discrete

Description



The First-Order Hold block implements a first-order sample-and-hold that operates at the specified sampling interval. This block has little value in practical applications and is included primarily for academic purposes.

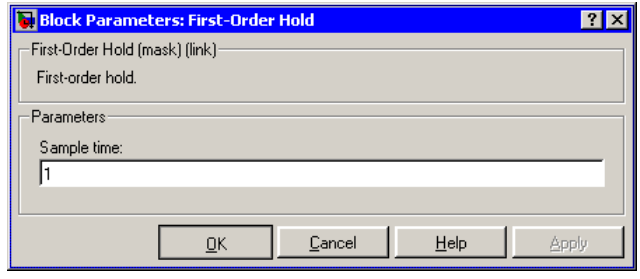
You can see the difference between the Zero-Order Hold and First-Order Hold blocks by running the demo program `fohdemo`. This figure compares the output from a Sine Wave block and a First-Order Hold block.



Data Type Support

The First-Order Hold block accepts and outputs signals of type `double`.

Parameters and Dialog Box



Sample time

The time interval between samples. See “Specifying Sample Time” in the online documentation for more information.

Characteristics

Direct Feedthrough	No
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
States	1 continuous and 1 discrete per input element
Dimensionalized	Yes
Zero Crossing	No

Purpose

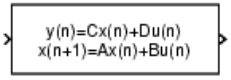
Implement discrete-time state space

Library

Additional Math & Discrete / Additional Discrete

Description

The Fixed-Point State-Space block implements the system described by



$$y(n) = Cx(n) + Du(n)$$

$$x(n+1) = Ax(n) + Bu(n)$$

where u is the input, x is the state, and y is the output. Both equations have the same data type.

The matrices A, B, C and D have the following characteristics:

- A must be an n-by-n matrix, where n is the number of states.
- B must be an n-by-m matrix, where m is the number of inputs.
- C must be an r-by-n matrix, where r is the number of outputs.
- D must be an r-by-m matrix.

In addition:

- The state x must be a n-by-1 vector
- The input u must be a m-by-1 vector
- The output y must be a r-by-1 vector

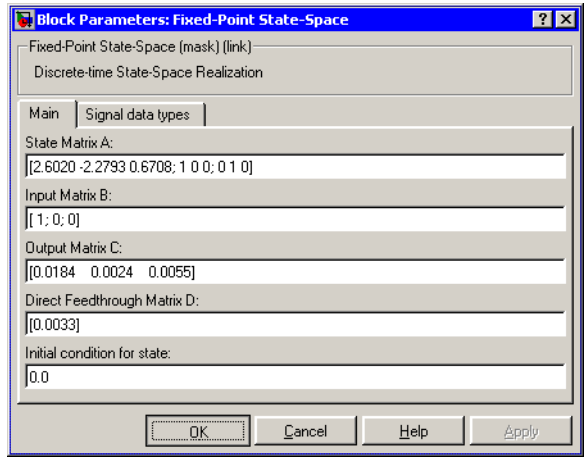
The block accepts one input and generates one output. The input vector width is determined by the number of columns in the B and D matrices. The output vector width is determined by the number of rows in the C and D matrices.

Data Type Support

The Fixed-Point State-Space block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box

The **Main** pane of the Fixed-Point State-Space block dialog appears as follows:



State Matrix A
Specify the matrix of states.

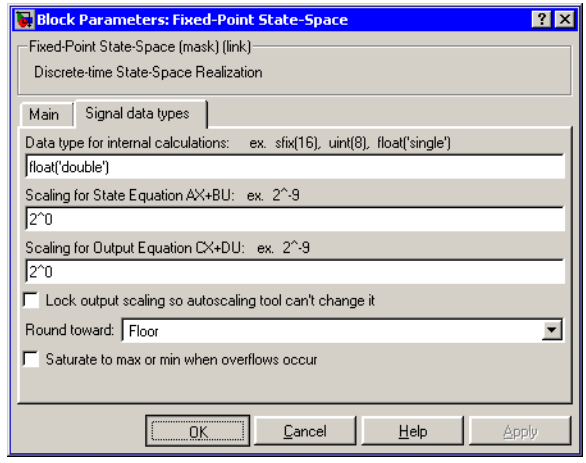
Input Matrix B
Specify the column vector of inputs.

Output Matrix C
Specify the column vector of outputs.

Direct Feedthrough Matrix D
Specify the matrix for direct feedthrough.

Initial condition for state
Specify the initial condition for the state.

The **Signal data types** pane of the Fixed-Point State-Space block dialog appears as follows:



Data type for internal calculations
Specify the data type for internal calculations.

Scaling for State Equation AX+BU
Specify the scaling for state equations.

Scaling for Output Equation CX+DU
Specify the scaling for output equations.

Lock output scaling so autoscaling tool can't change it
If you select this check box, the output scaling is locked.

Round toward
Select the rounding mode for fixed-point operations.

Saturate to max or min when overflows occur
If selected, fixed-point overflows saturate.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

For Iterator

Purpose

Repeatedly execute the contents of a subsystem at the current time step until an iteration variable exceeds a specified iteration limit.

Library

Ports & Subsystems/For Iterator Subsystem

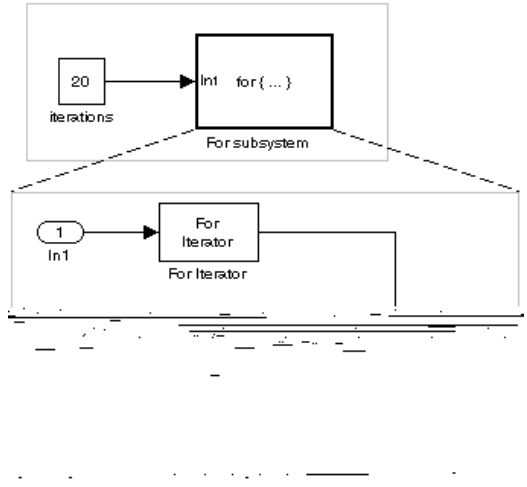
Description

The For Iterator block, when placed in a subsystem, repeatedly executes the contents of the subsystem at the current time step until an iteration variable exceeds a specified iteration limit. You can use this block to implement the block diagram equivalent of a `for` loop in the C programming language.

The block's parameter dialog allows you to specify the maximum value of the iteration variable or an external source for the maximum value and an optional external source for the next value of the iteration variable. If you do not specify an external source for the next value of the iteration variable, the next value is determined by incrementing the current value:

$$i_{n+1} = i_n + 1$$

The model in the following figure uses a For Iterator block to increment an initial value of zero by 10 over 20 iterations at every time step.



The following figure shows the result.

The For Iterator subsystem in this example is equivalent to the following C code.

If the matrix includes two or more columns at the same time value, the output is the data point for the first column encountered. For example, for a matrix that has this data:

```
time values:    0 1 2 2
data points:    2 3 4 5
```

At time 2, the output is 4, the data point for the first column encountered at that time value.

Simulink reads the MAT file into memory at the start of the simulation. As a result, you cannot read data from the same MAT file named in a To File block in the same model.

Using Data Saved by a To File or a To Workspace Block

The From File block can read data written by a To File block without any modifications.

To read data written by a To Workspace block and saved to a MAT file:

- The data must include the simulation times. The easiest way to include time data in the simulation output is to specify a variable for time on the **Data Import/Export** pane of the **Configuration Parameters** dialog box. See The Data Import/Export Pane for more information.
- Before saving the data from the To Workspace block, transpose it to the form expected by the From File block.

Data Type Support

The From File block outputs real signals of type `double`.

Parameters and Dialog Box

Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in the online Simulink documentation for details.

File name

The fully qualified pathname or file name of the MAT file that contains the data used as input. On UNIX, the pathname can start with a tilde (~) character signifying your home directory. The default file name is `untitled.mat`. If you specify an unqualified file name, Simulink assumes that the MAT file resides in the MATLAB working directory. (To determine the working directory, enter `pwd` at the MATLAB command line.) If Simulink cannot find the specified file name in the working directory, it displays an error message.

Sample time

The sample period and offset of the data read from the file. See “Specifying Sample Time”in the online documentation for more information.

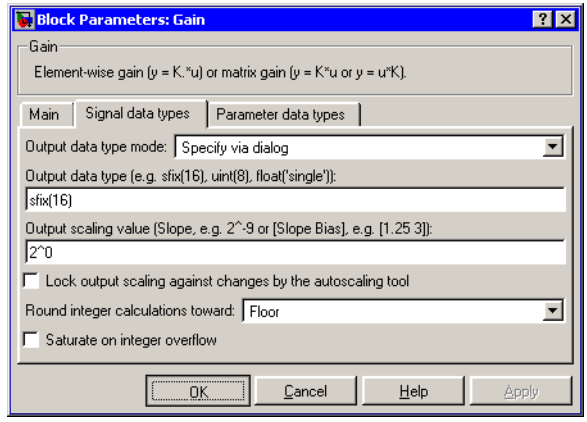
Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See “Specifying Sample Time” in the online documentation for more information.

The **Signal data types** pane of the Gain block dialog appears as follows:



Output data type mode

Set the data type and scaling of the output to be the same as that of the input, or to be inherited via an internal rule or by back propagation. Alternatively, choose to specify the data type and scaling of the output through the **Output data type** and **Output scaling value** parameters in the dialog.

If you select `Inherit via internal rule` for this parameter, Simulink chooses a combination of output scaling and data type that requires the smallest amount of memory consistent with accommodating the output range and maintaining the output precision of the block and with the word size of the targeted hardware implementation specified for the model. If the **Device type** parameter on the **Hardware implementation** configuration parameters pane is set to `ASIC/FPGA`, Simulink chooses the output data type without regard to hardware

constraints. Otherwise, Simulink chooses the smallest available hardware data type capable of meeting the range and precision constraints. For example, if the block multiplies an input of type `int8` by a gain of `int16` and `ASIC/FPGA` is specified as the targeted hardware type, the output data type is `sfix24`. If `Unspecified (assume 32-bit Generic)`, i.e., a generic 32-bit microprocessor, is specified as the target hardware, the output data type is `int32`. If none of the word lengths provided by the target microprocessor can accommodate the output range, Simulink displays an error message in the Simulation Diagnostics Viewer.

Output data type
Set the output data type. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Output scaling value
Set the output scaling using either binary point-only or [Slope Bias] scaling. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Lock output scaling against changes by the autoscaling tool
Select to lock scaling of outputs. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Round integer calculations toward
Select the rounding mode for fixed-point operations.

Saturate on integer overflow
Select to have overflows saturate.

The **Parameter data types** pane of the Gain block dialog appears as follows:

Parameter data type mode
Set the data type and scaling of the gain to be the same as that of the input, or to be inherited via an internal rule. Alternatively, choose to specify the data type and scaling of the gain through the **Parameter data type**, **Parameter scaling mode**, and **Parameter scaling** parameters in the dialog.

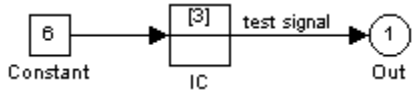
Parameter data type
Specifies the data type of the **Gain** parameter. This parameter is visible only if you select `Specify via dialog` for the **Parameter data type mode** parameter.

Parameter scaling mode
Set the mode to determine the scaling of the gain.

- `Use specified scaling`—This mode allows you to set the scaling of the gain in the **Parameter scaling** parameter.
- `Best Precision: Element-wise`—This mode sets binary points for the elements of the gain such that the precision of each element is maximized.
- `Best Precision: Row-wise`—This mode sets a common binary point within each row of the gain such that the largest element of each row has the best possible precision.

Examples

The following diagram illustrates how the IC block initializes a signal labeled "test signal."



At t = 0, the signal value is 3. Afterwards, the signal value is 6.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of parameter only
Dimensionalized	Yes
Zero Crossing	No

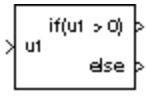
Purpose

Implement a C-like `if-else` control flow statement in Simulink

Library

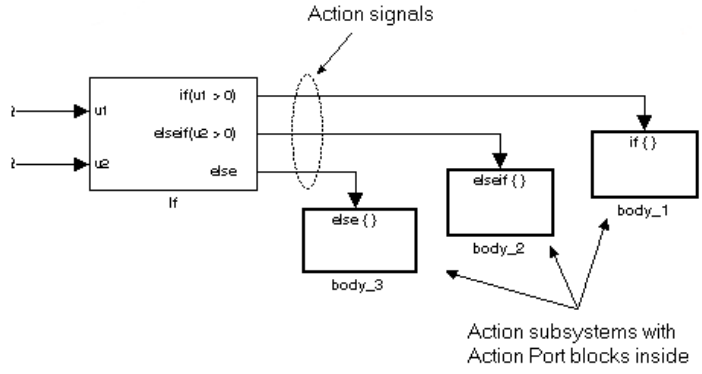
Ports & Subsystems

Description



The If block, along with If Action subsystems containing Action Port blocks, implements standard C-like `if-else` logic.

The following shows a completed `if-else` control flow statement.



In this example, the inputs to the If block determine the values of conditions represented as output ports. Each output port is attached to an If Action subsystem. The conditions are evaluated top down starting with the `if` condition. If a condition is true, its If Action subsystem is executed and the If block does not evaluate any remaining conditions.

The preceding `if-else` control flow statement can be represented by the following pseudocode.

```
if (u1 > 0) {
 body_1;
}
else if (u2 > 0){
 body_2;
}
else {
 body_3;
```

```
}
```

You construct a Simulink `if-else` control flow statement like the preceding example as follows:

1. Place an If block in the current system.
2. Open the **Block Parameters** dialog of the If block and enter as follows:
   - Enter the **Number of inputs** field with the required number of inputs necessary to define conditions for the `if-else` control flow statement.

     Elements of vector inputs can be accessed for conditions using (row, column) arguments. For example, you can specify the fifth element of the vector u2 in the condition u2(5) > 0 in an **If expression** or **Elseif expressions** field.
   - Enter the expression for the if condition of the `if-else` control flow statement in the **If expression** field.

     This creates an if output port for the If block with a label of the form `if(condition)`. This is the only required If Action signal output for an If block.
   - Enter expressions for any elseif conditions of the `if-else` control flow statement in the **Elseif expressions** field.

     Use a comma to separate one condition from another. Entering these conditions creates an output port for the If block for each condition, with a label of the form `elseif(condition)`. elseif ports are optional and not required for operation of the If block.
   - Check the **Show else condition** check box to create an else output port.

     The else port is optional and not required for the operation of the If block.

3 Create If Action subsystems to connect to each of the if, else, and elseif ports.

   These consist of a subsystem with an Action Port block. When you place an Action Port block inside each subsystem, an input port named Action is added to the subsystem.

4 Connect each if, else, and elseif port of the If block to the Action port of an If Action subsystem.

   When you make the connection, the icon for the If Action block is renamed to the type of the condition that it attaches to.

Note During simulation of an `if-else` control flow statement, the Action signal lines from the If block to the If Action subsystems turn from solid to dashed.

---

5 In each If Action subsystem, enter the Simulink blocks appropriate to the body to be executed for the condition it handles.

   In the preceding example, this is shown as `body_1`, `body_2`, and `body_3`.

Data Type Support

Inputs `u1,u2,...,un` can be scalar or vector of any built-in Simulink data type. For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Outputs from the if, else, and elseif ports are Action signals to If Action subsystems that are created with Action Port blocks and subsystems. See `Action Port`.

Parameters and Dialog Box

Number of inputs

The number of inputs to the If block. These appear as data input ports labeled with a `'u'` character followed by a number, `1,2,...,n`, where `n` equals the number of inputs that you specify.

Port number

Specify the port number of the Inport block.

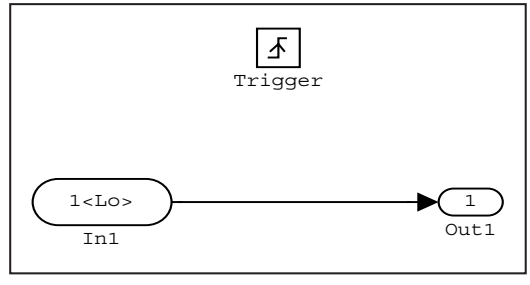
Icon display

Specifies the information to be displayed on the icon of this input port. The options are:

`Port number`	Displays port number of this port.
`Signal name`	Displays the name of the signal connected to this port (or signals if the input is a bus).
`Port name and signal name`	Displays both the port number and the names of the signals connected to this port.

Latch input by delaying outside signal

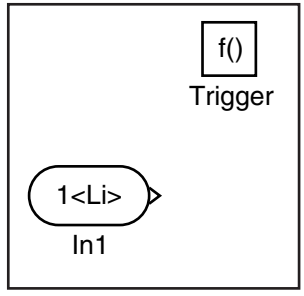
This option applies only to triggered subsystems and is enabled only if the Inport block resides in a triggered subsystem. If selected, the block outputs the value of the input signal at the previous time step. The Inport block indicates that this option is selected by displaying <Lo>.



Latch input by copying inside signal

This option applies only to function-call subsystems and hence is enabled only if the Inport block resides in a function-call subsystem. Selecting this option causes Simulink to copy the signal output by the block into a buffer before executing the

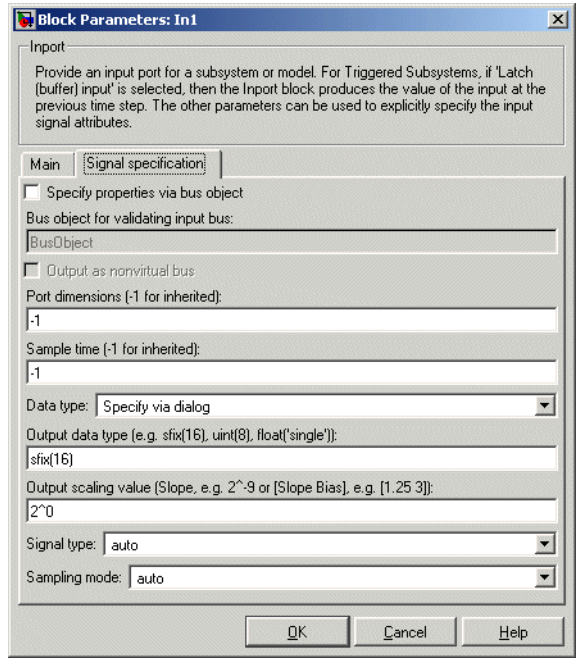
contents of the subsystem and to use this copy as the block's output during execution of the subsystem. This ensures that the subsystem's inputs, including those generated by the subsystem's context, will not change during execution of the subsystem. The Inport block displays <Li> to indicate that this option is selected.



Interpolate data

Select this parameter to cause the block to interpolate or extrapolate output at time steps for which no corresponding workspace data exists when loading data from the workspace. See "Importing Data from the MATLAB Workspace" for more information.

The **Signal specification** pane of the Inport block dialog appears as follows:



Specify properties via bus object

Select this option to use a bus object to define the structure of the bus created by this block (see "Working with Data Objects" and `Simulink.Bus` class to learn how to create bus objects).

Bus object for validating input bus

This option is enabled only if you select the **Specify properties via bus object option**. It specifies the name of the bus object that defines the structure of that a bus must have to be connected to this input port. At the beginning of a simulation or when you update the model's diagram, Simulink checks whether the bus connected to this input port has the specified structure. If not, Simulink displays an error message.

Output as nonvirtual bus

This option is enabled only if you select the **Specify properties via bus object option**. If this option is selected, this block outputs a nonvirtual bus; otherwise, it outputs a virtual bus (see "Virtual Versus Nonvirtual Buses"). Select this option if you want code generated from this model to use a C structure to define the structure of the bus signal output by this block.

Port dimensions

Specify the dimensions of the input signal to the Inport block. Valid values are:

`-1`	Dimensions are inherited from input signal
`n`	Vector signal of width `n` accepted
`[m n]`	Matrix signal having `m` rows and `n` columns accepted

Sample time

Specify the sample time of the input signal. See "Specifying Sample Time".

Data type

Specify the data type of the external input. To accept any data type, set this parameter to `auto`.

Output data type

Specify any data type, including fixed-point data types. This parameter is only visible if you select `Specify via dialog` for the **Data type** parameter.

Output scaling value

Set the output scaling using binary point-only or [Slope Bias] scaling. This parameter is only visible if you select `Specify via dialog` for the **Data type** parameter.

Signal type

Specify the numeric type (`real` or `complex`) of the external input. To accept either type, set this parameter to `auto`.

Sampling mode

Specify the sampling mode (`Sample based` or `Frame based`) that the input signal must match. To accept any sampling mode, set this parameter to `auto`. This parameter is intended to support signal processing applications based on Simulink. See the documentation for the `buffer` function provided by the Signal Processing Toolbox or "“Frame-Based Signals” in the documentation for the Signal Processing Blockset for information about frame-based signals.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Dimensionalized	Yes

Integer Delay

Purpose

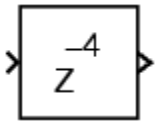
Delay a signal N sample periods

Library

Discrete

Description

The Integer Delay block delays its input by N sample periods.

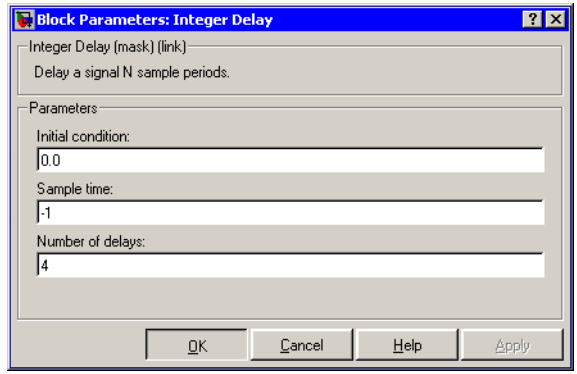


The block accepts one input and generates one output, both of which can be scalar or vector. If the input is a vector, all elements of the vector are delayed by the same sample period.

Data Type Support

The Integer Delay block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Initial condition

The initial output of the simulation. The **Initial condition** parameter is converted from a double to the input data type offline using round-to-nearest and saturation.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See “Specifying Sample Time” in the online documentation for more information.

Number of delays
The number of periods to delay the input signal.

Characteristics

Direct Feedthrough	No
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of input or initial conditions

Integrator

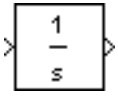
Purpose

Integrate a signal

Library

Continuous

Description



The Integrator block outputs the integral of its input at the current time step. The following equation represents the output of the block y as a function of its input u and an initial condition y_0, where y and u are vector functions of the current simulation time t.

$$y(t) = \int_{t_0}^{t} u(t)dt + y_0$$

Simulink can use a number of different numerical integration methods to compute the Integrator block's output, each with advantages in particular applications. The **Solver** pane of the **Configuration parameters** dialog box (see The Solver Pane) allows you to select the technique best suited to your application.

Simulink treats the Integrator block as a dynamic system with one state, its output. The Integrator block's input is the state's time derivative.

$$x = y(t)$$

$$x_0 = y_0$$

$$\dot{x} = u(t)$$

The currently selected solver computes the output of the Integrator block at the current time step, using the current input value and the value of the state at the previous time step. To support this computational model, the Integrator block saves its output at the current time step for use by the solver to compute its output at the next time step. The block also provides the solver with an initial condition for use in computing the block's initial state at the beginning of a simulation run. The default value of the initial condition is 0. The block's parameter dialog box allows you to specify another value for the initial condition or create an initial value input port on the block.

The parameter dialog box also allows you to

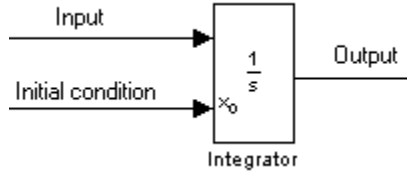
- Define upper and lower limits on the integral
- Create an input that resets the block's output (state) to its initial value, depending on how the input changes
- Create an optional state output that allows you to use the value of the block's output to trigger a block reset

Use the Discrete-Time Integrator block to create a purely discrete system.

Defining Initial Conditions

You can define the initial conditions as a parameter on the block dialog box or input them from an external signal:

- To define the initial conditions as a block parameter, specify the **Initial condition source** parameter as `internal` and enter the value in the **Initial condition** parameter field.
- To provide the initial conditions from an external source, specify the **Initial condition source** parameter as `external`. An additional input port appears under the block input, as shown in this figure.



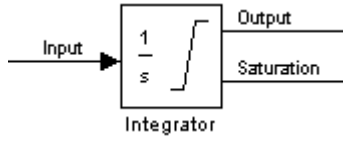
Note If the integrator limits its output (see **"Limiting the Integral" on page 2-320**), the initial condition must fall inside the integrator's saturation limits. If the initial condition is outside the block's saturation limits, the block displays an error message.

Limiting the Integral

To prevent the output from exceeding specifiable levels, select the **Limit output** check box and enter the limits in the appropriate parameter fields. Doing so causes the block to function as a limited integrator. When the output reac-2 hes the limits, the integral action is turned off to prevent integral wind up. During a simulation, you can change the limits but you cannot change whether the output is limited. The output is determined as follows:

- When the integral is less than or equal to the **Lower saturation limit** and the input is negative, the output is held at the **Lower saturation limit**.
- When the integral is between the **Lower saturation limit** and the **Upper saturation limit**, the output is the integral.
- When the integral is greater than or equal to the **Upper saturation limit** and the input is positive, the output is held at the **Upper saturation limit**.

To generate a signal that indicates when the state is being limited, select the **Show saturation port** check box. A saturation port appears below the block output port, as shown on this figure.



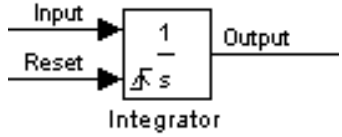
The signal has one of three values:

- 1 indicates that the upper limit is being applied.
- 0 indicates that the integral is not limited.
- -1 indicates that the lower limit is being applied.

When you select this option, the block has three zero crossings: one to detect when it enters the upper saturation limit, one to detect when it enters the lower saturation limit, and one to detect when it leaves saturation.

Resetting the State

The block can reset its state to the specified initial condition based on an external signal. To cause the block to reset its state, select one of the **External reset** choices. A trigger port appears below the block's input port and indicates the trigger type, as shown in this figure.

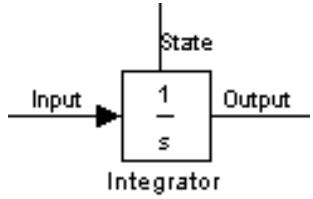


- Select `rising` to trigger the state reset when the reset signal has a rising edge.
- Select `falling` to trigger the state reset when the reset signal has a falling edge.
- Select `either` to trigger the reset when either a rising or falling signal occurs.
- Select `level` to trigger the reset and hold the output to the initial condition while the reset signal is nonzero.

The reset port has direct feedthrough. If the block output is fed back into this port, either directly or through a series of blocks with direct feedthrough, an algebraic loop results (see "Algebraic Loops"). The Integrator block's state port allows you to feed back the block's output without creating an algebraic loop.

About the State Port

Selecting the **Show state port** option on the Integrator block's parameter dialog box causes an additional output port, the state port, to appear atop the Integrator block.



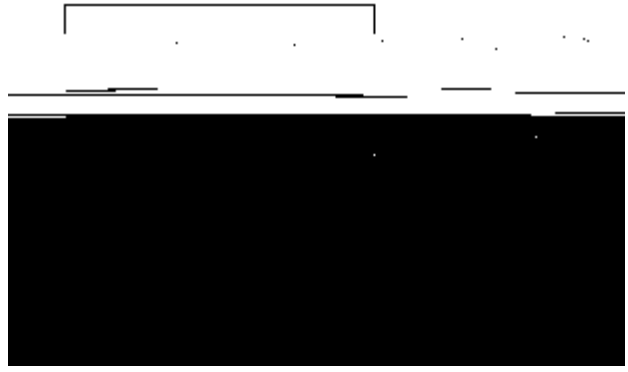
The output of the state port is the same as the output of the block's standard output port except for the following case. If the block is reset in the current time step, the output of the state port is the value that would have appeared at the block's standard output if the block had not been reset. The state port's output appears earlier in the time step than the output of the Integrator block's output port. This allows you to avoid creating algebraic loops in the following modeling scenarios:

- Self-resetting integrators (see "Creating Self-Resetting Integrators" on page 2-322)
- Handing off a state from one enabled subsystem to another (see "Handing Off States Between Enabled Subsystems" on page 2-324)

Note The state port is intended to be used specifically in these two scenarios. When updating a model, Simulink checks to ensure that the state port is being used in one of these two scenarios. If not, Simulink signals an error. Also, Simulink does not allow you to log the output of this port in a referenced model. If logging is enabled for the port, Simulink generates a "signal not found" warning during simulation of the referenced model.

Creating Self-Resetting Integrators

The Integrator block's state port allows you to avoid creating algebraic loops when creating an integrator that resets itself based on the value of its output. Consider, for example, the following model.



This model tries to create a self-resetting integrator by feeding the integrator's output, subtracted from 1, back into the integrator's reset port. In so doing, however, the model creates an algebraic loop. To compute the integrator block's output, Simulink needs to know the value of the block's reset signal, and vice versa. Because the two values are mutually dependent, Simulink cannot determine either. It therefore signals an error if you try to simulate or update this model.

The following model uses the integrator's state port to avoid the algebraic loop.

Upper saturation limit
The upper limit for the integral. The default is `inf`.

Lower saturation limit
The lower limit for the integral. The default is `-inf`.

Show saturation port
If selected, adds a saturation output port to the block.

Show state port
If selected, adds an output port to the block for the block's state.

Absolute tolerance
Absolute tolerance used to compute the block's outputs. You can enter `auto` or a numeric value. If you enter `auto`, Simulink determines the absolute tolerance (see "Error Tolerances"). If you enter a numeric value, Simulink uses the specified value to compute the block's outputs. Note that a numeric value overrides the setting for the absolute tolerance in the **Configuration Parameters** dialog box.

Ignore limit and reset when linearizing
Select this option to cause Simulink linearization commands to treat this block as unresettable and as having no limits on its output, regardless of the settings of the block's reset and output limitation options. This allows you to linearize a model around an operating point that causes the integrator to reset or saturate.

Enable zero crossing detection
If this option, **Limit output**, and zero-crossing detection for the model as a whole are selected, the Integrator block uses zero-crossings to detect and take a time step at any of the following events: reset, entering or leaving an upper saturation state, entering or leaving a lower saturation state. For more information, see "Zero Crossing Detection" in the Using Simulink documentation.

Characteristics

Direct Feedthrough	Yes, of the reset and external initial condition source ports
Sample Time	Continuous
Scalar Expansion	Yes, of parameters
States	Inherited from driving block or parameter
Dimensionalized	Yes
Zero Crossing	Yes, if enabled and you select the **Limit output** option, one for detecting reset, one each to detect upper and lower saturation limits, one when leaving saturation

Interpolation (n-D) Using PreLookup

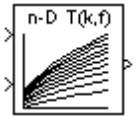
Purpose

Perform high-performance constant or linear interpolation, mapping N input values to a sampled representation of a function in N variables via output from PreLookup Index Search block

Library

Lookup Tables

Description



The Interpolation (n-D) Using PreLookup block uses the precalculated indices and interval fractions from the PreLookup Index Search block to perform the equivalent operation that the Lookup Table (n-D) performs. This combination of blocks allows multiple Interpolation (n-D) blocks to feed a set of PreLookup Index Search blocks. In models that have many interpolation blocks, simulation performance can be greatly increased.

This block supports two interpolation methods: flat (constant) interval lookup and linear interpolation. These operations can be applied to 1-D, 2-D, 3-D, 4-D and higher dimensioned tables.

You define a set of output values as the **Table data** parameter. These table values must correspond to the breakpoint data sets that are in the PreLookup Index Search block. The block generates its output by interpolating the table values based on the (index, fraction) pairs fed into the block by each PreLookup Index Search block.

The block generates output based on the input values:

- If the inputs match breakpoint parameter values, the output is the table value at the intersection of the row, column, and higher dimensions' breakpoints.
- If the inputs do not match row and column parameter values, the block generates output by interpolating between the appropriate table values. If either or both block inputs are less than the first or greater than the last row or column parameter values, the block extrapolates from the first two or last two points in each corresponding dimension.

The block can perform interpolation on portion of the table. For more information, see "Lookups: Prelookup and Sub-Table Interpolation

Blocks" in the "Block Diagramming Features" section of the "Simulink" section on the **Demos** pane of the MATLAB Help browser.

Data Type Support

The Interpolation (n-D) Using PreLookup block accepts signals of types `double` or `single`, but for any given block, the inputs must all be of the same type. The **Table data** parameter must be of the same type as the inputs. The output data type is set to the **Table data** data type.

Parameters and Dialog Box

Number of table dimensions

The number of dimensions that the **Table data** parameter must have. This determines the number of independent variables for the table and hence the number of inputs to the block (see descriptions for "Explicit Number of dimensions" and "Use one (vector) input port instead of N ports," below).

Table data

The table of output values. The matrix size must match the dimensions defined by the N **breakpoint set** parameter or by the

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of inputs
Dimensionalized	Yes
Zero Crossing	No

Lookup Table

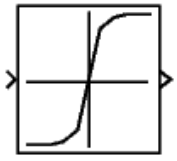
Purpose

Approximate a one-dimensional function using the specified lookup method

Library

Lookup Tables

Description



The Lookup Table block computes an approximation to some function `y=f(x)` given data vectors `x` and `y`.

Note To map two inputs to an output, use the Lookup Table (2-D) block.

The length of the `x` and `y` data vectors provided to this block must match. Also, the `x` data vector must be strictly monotonically increasing after conversion to the input's fixed-point data type, except in the following case. If the input `x` and the output signal are both either `single` or `double`, and if the lookup method is `Interpolation-Extrapolation`, then `x` may be monotonically increasing rather than strictly monotonically increasing. Note that due to quantization, the `x` data vector may be strictly monotonic in doubles format, but not so after conversion to a fixed-point data type.

You define the table by specifying the **Vector of input values** parameter as a 1-by-n vector and the **Vector of output values** parameter as a 1-by-n vector. The block generates output based on the input values using one of these methods selected from the **Look-up method** parameter list:

- `Interpolation-Extrapolation`—This is the default method; it performs linear interpolation and extrapolation of the inputs.
  - If a value matches the block's input, the output is the corresponding element in the output vector.
  - If no value matches the block's input, then the block performs linear interpolation between the two appropriate elements of the table to determine an output value. If the block input is less than

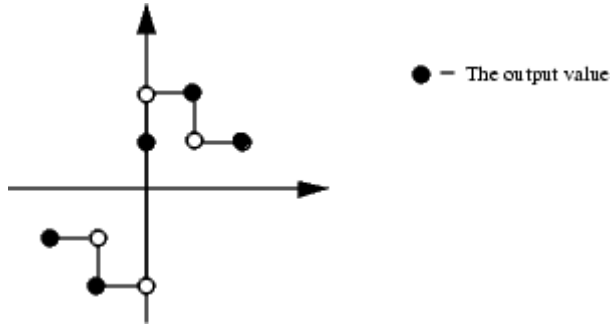
the first or greater than the last input vector element, then the block extrapolates using the first two or last two points.

- `Interpolation-Use End Values`—This method performs linear interpolation as described above but does not extrapolate outside the end points of the input vector. Instead, the end-point values are used.
- `Use Input Nearest`—This method does not interpolate or extrapolate. Instead, the element in `x` nearest the current input is found. The corresponding element in `y` is then used as the output.
- `Use Input Below`—This method does not interpolate or extrapolate. Instead, the element in `x` nearest and below the current input is found. The corresponding element in `y` is then used as the output. If there is no element in `x` below the current input, then the nearest element is found.
- `Use Input Above`—This method does not interpolate or extrapolate. Instead, the element in `x` nearest and above the current input is found. The corresponding element in `y` is then used as the output. If there is no element in `x` above the current input, then the nearest element is found.

Note Note that there is no difference among the `Use Input Nearest`, `Use Input Below`, and `Use Input Above` methods when the input `x` corresponds exactly to table breakpoints.

To create a table with step transitions, repeat an input value with different output values. For example, these input and output parameter values create the input/output relationship described by the plot that follows:

```
Vector of input values:   [-2 -1 -1  0 0 0 1 1 2]
Vector of output values:  [-1 -1 -2 -2 1 2 2 1 1]
```



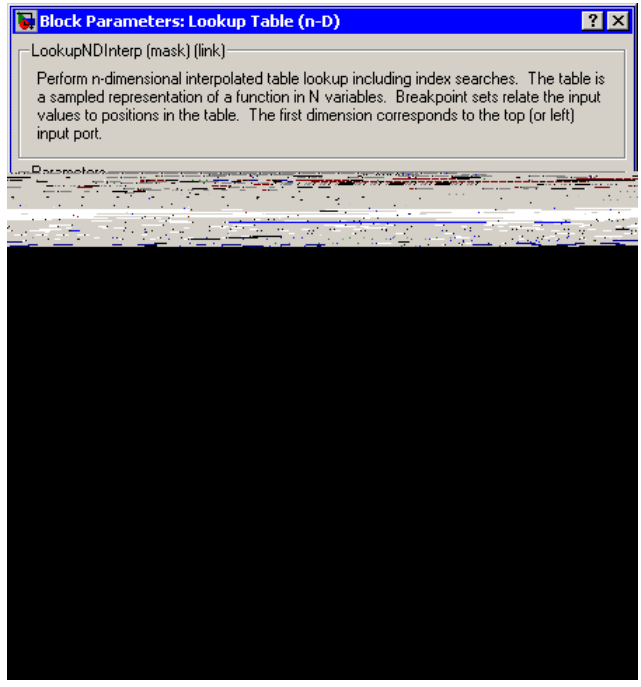
This example has three step discontinuities: at u = -1, 0, and +1.

When there are two points at a given input value, the block generates output according to these rules:

- When the input signal u is less than zero, the output is the value connected with the point first encountered when moving away from the origin in a negative direction. In this example, when u is -1, y is -2, marked with a solid circle.
- When u is greater than zero, the output is the value connected with the point first encountered when moving away from the origin in a positive direction. In this example, when u is 1, y is 2, marked with a solid circle.
- When u is at the origin and there are two output values specified for zero input, the actual output is their average. In this example, if there were no point at u = 0 and y = 1, the output would be 0, the average of the two points at u = 0. If there are three points at zero, the block generates the output associated with the middle point. In this example, the output at the origin is 1.

The Lookup Table icon displays a graph of the input vector versus the output vector. When a parameter is changed on the block's dialog box, the graph is automatically redrawn when you click the **OK** or **Apply** button.

Parameters and Dialog Box



Number of table dimensions

The number of dimensions that the **Table data** parameter is to have. This determines the number of independent variables for the table and hence the number of inputs to the block (see descriptions for **Explicit Number of dimensions** and **Use one (vector) input port instead of N ports** following).

First input (row) breakpoint set

The row values represented in the table, entered as a vector. The vector values must increase monotonically. This field is always visible.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See “Specifying Sample Time” in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	N/A
Dimensionalized	Yes
Zero Crossing	No

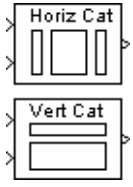
Purpose

Concatenate inputs horizontally or vertically

Library

Math Operations

Description



The Matrix Concatenation block concatenates input matrices u_1, u_2, ..., u_n along rows or columns, where n is specified by the **Number of inputs** parameter. The block accepts inputs with any combination of built-in Simulink data types and/or fixed-point data types. If all inputs are sample-based, the output is sample-based. Otherwise, the output is frame-based.

Horizontal Matrix Concatenation

When the **Concatenation method** parameter is set to **Horizontal**, the block concatenates the input matrices along *rows*.

```
y = [u1 u2 u3 ... un]        % Equivalent MATLAB code
```

For horizontal concatenation, inputs must all have the same row dimension, M, but can have different column dimensions. The output matrix has dimension M-by-ΣN_i, where N_i is the number of columns in input u_i (i = 1, 2, ..., n).

When some of the inputs are length-M 1-D vectors while others are M-by-N_i matrices, the vector inputs are treated as M-by-1 matrices.

Vertical Matrix Concatenation

When the **Concatenation method** parameter is set to **Vertical**, the block concatenates the input matrices along *columns*.

```
y = [u1;u2;u3;...;un]        % Equivalent MATLAB code
```

For vertical concatenation, inputs must all have the same column dimension, N, but can have different row dimensions. The output matrix has dimension ΣM_i-by-N, where M_i is the number of rows in input u_i (i = 1, 2, ..., n).

When some of the inputs are length-M_i 1-D vectors while others are M_i-by-1 matrices, the vector inputs are treated as M_i-by-1 matrices. (1-D vector inputs are not accepted for vertical concatenation when the other inputs have column dimension greater than 1.)

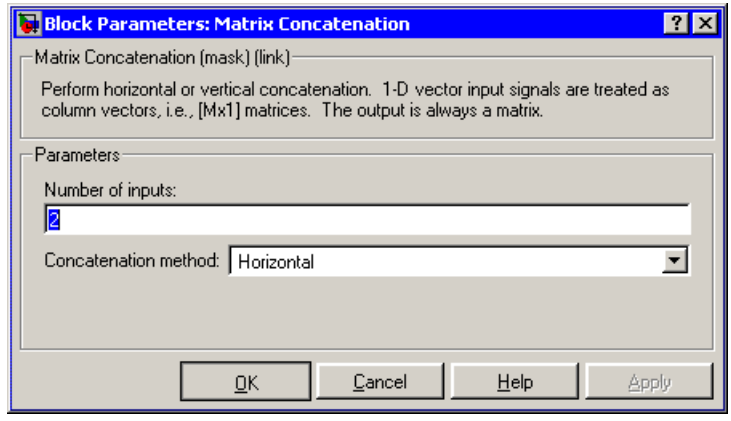
1-D Vector Concatenation

When all inputs to the Matrix Concatenation block are length-M_i 1-D vectors, the output is a ΣM_i-by-1 matrix containing all input elements concatenated in port order: the elements in the vector input to the top port appear as the first elements in the output, and the elements in the vector input to the bottom port appear as the last elements in the output.

Data Type Support

The Matrix Concatenation block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Number of inputs
The number of matrices to concatenate.

Concatenation method
The dimension along which to concatenate the inputs.

Purpose

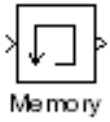
Output the input from the previous time step

Library

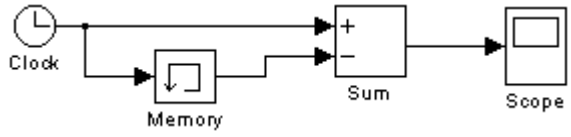
Discrete

Description

The Memory block outputs its input from the previous time step, applying a one integration step sample-and-hold to its input signal.



This sample model demonstrates how to display the step size used in a simulation. The Sum block subtracts the time at the previous step, generated by the Memory block, from the current time, generated by the clock.



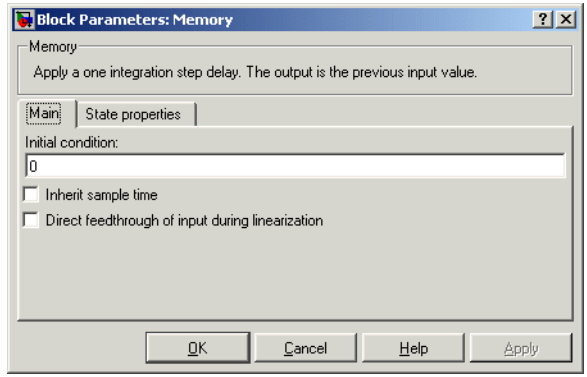
Note Avoid using the Memory block when integrating with `ode15s` or `ode113`, unless the input to the block does not change.

Data Type Support

The Memory block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box



Initial condition

The output at the initial integration step. This must be set to 0 if the input data type is user-defined. Simulink does not allow the initial output of this block to be `inf` or `NaN`.

Inherit sample time

Check this check box to cause the sample time to be inherited from the driving block. If this option is not selected, the block's sample time depends on the type of solver used to simulate the model. If the solver is a variable-step solver, the sample time is zero-order-hold (ZOH) continuous. If the solver is a fixed-step solver, the sample time equals the solver's step size.

Direct feedthrough of input during linearization

Causes the block to output its input during linearization and trim. This sets the block's mode to direct feedthrough.

Enabling this check box can cause a change in the ordering of states in the model when using the functions `linmod`, `dlinmod`, or `trim`. To extract this new state ordering, use the following commands.

First compile the model using the following command, where `model` is the name of the Simulink model.

```
[sizes, x0, x_str] = model([],[],[],'lincompile');
```

Next, terminate the compilation with the following command.

```
model([],[],[],'term');
```

The output argument, `x_str`, which is a cell array of the states in the Simulink model, contains the new state ordering. When passing a vector of states as input to the `linmod`, `dlinmod`, or `trim` functions, the state vector must use this new state ordering.

The **State properties** pane of this block pertains to code generation and has no effect on model simulation. See "Block States: Storing and Interfacing" in the Real-Time Workshop documentation for more information.

Characteristics

Direct Feedthrough	No, except when **Direct feedthrough of input during linearization is enabled**.
Sample Time	Continuous, but inherited from the driving block if you select the **Inherit sample time** check box
Scalar Expansion	Yes, of the **Initial condition** parameter
Dimensionalized	Yes
Zero Crossing	No

Merge

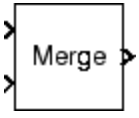
Purpose

Combine multiple signals into a single signal

Library

Signal Routing

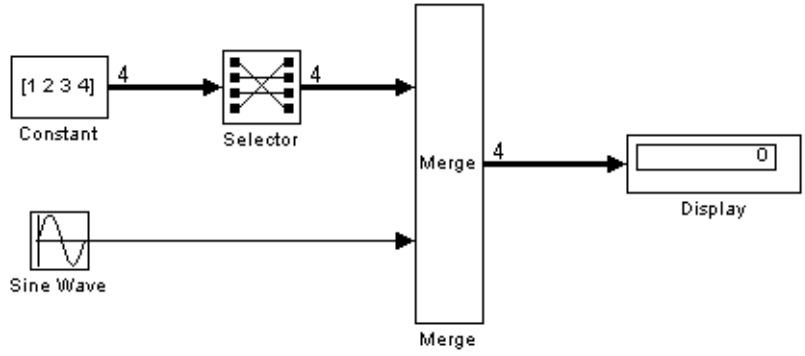
Description



The Merge block combines its inputs into a single output line whose value at any time is equal to the most recently computed output of its driving blocks. You can specify any number of inputs by setting the block's **Number of inputs** parameter.

Note Merge blocks facilitate creation of alternately executing subsystems. See "Creating Alternately Executing Subsystems" for an application example.

A Merge block does not accept signals whose elements have been reordered. For example, in the following diagram, the Merge block does not accept the output of the Selector block because the Selector block interchanges the first and fourth elements of the vector signal.

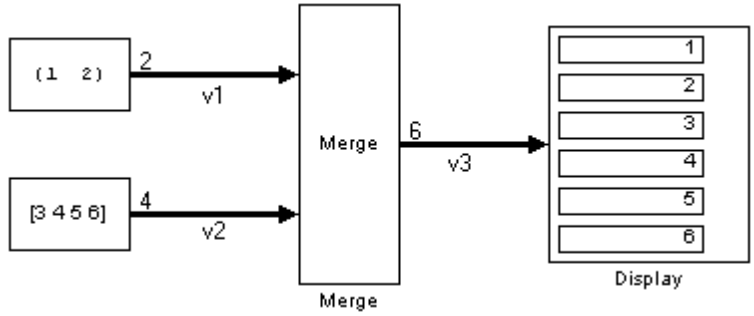


If the **Allow unequal port widths** parameter is not selected, the block accepts only inputs of equal dimensions and outputs a signal of the same dimensions as the inputs. If you select the **Allow unequal port widths** option, the block accepts scalars and vectors (but not matrices)

having differing numbers of elements. Further, the block allows you to specify an offset for each input signal relative to the beginning of the output signal. The width of the output signal is

$$\max(w_1+o_1,\ w_2+o_2,\ \ldots\ w_n+o_n)$$

where $w_1, \ldots w_n$ are the widths of the input signals and $o_1, \ldots o_n$ are the offsets for the input signals. For example, the Merge block in the following diagram merges signals `v1` and `v2` to produce signal `v3`.



In this example, the offset of `v1` is `0` and the offset of v2 is `2`, resulting in an output signal six elements wide. The Merge block maps the elements of `v1` to the first two elements of `v3` and the elements of `v2` to the last four elements of `v3`.

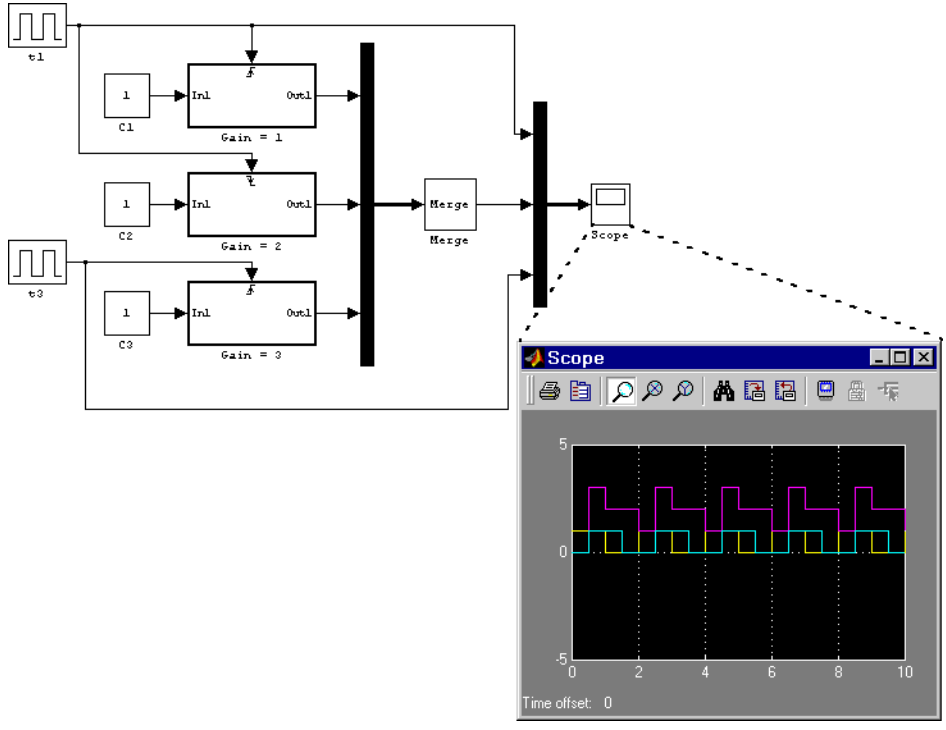
You can specify an initial output value by setting the block's **Initial output** parameter. If you do not specify an initial output and one or more of the driving blocks do, the Merge block's initial output equals the most recently evaluated initial output of the driving blocks.

Merging S-Function Outputs

The Merge block can merge a signal from an S-Function block only if the memory used to store the S-Function block's output is reusable. Simulink displays an error message if you attempt to update or simulate a model that connects a nonreusable port of an S-Function block to a Merge block. See `ssSetOutputPortOptimOpts` for more information.

Muxing Signals to be Merged

Instead of connecting signals directly to a Merge block, you can connect them via a Mux block as illustrated in the following example.



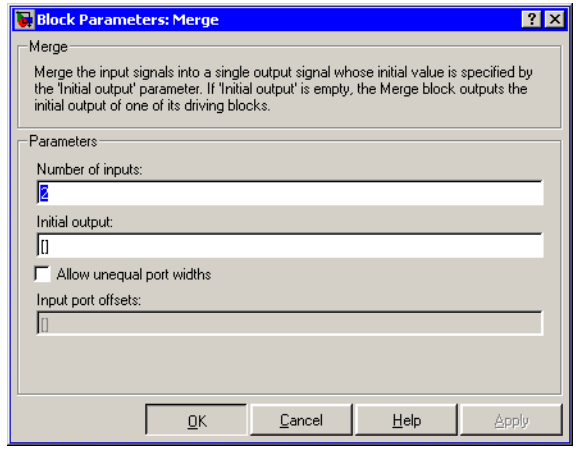
This example connects three amplifiers to a Merge block via a Mux block. The top and bottom amplifiers trigger on a rising pulse; the middle, on a falling pulse. The trigger signal connected to the bottom amplifier has a phase delay of `.5` s compared to the trigger signal connected to the top amplifier. The output of the Merge block at each time step equals that of the amplifier triggered at that time step. Muxing the signals to be merged rather than connecting them directly to the Merge block can result in a clearer diagram.

Data Type Support

The Merge block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box



Number of inputs

The number of input ports to merge.

Initial output

Initial value of output. If unspecified, the initial output equals the initial output, if any, of one of the driving blocks. Simulink does not allow you to set the initial output of this block to `inf` or `NaN`.

Allow unequal port widths

Allows the block to accept inputs having different numbers of elements.

Input port offsets

Vector specifying the offset of each input signal relative to the beginning of the output signal.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from the driving block
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

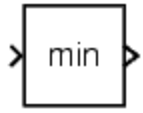
Purpose

Output the minimum or maximum input value

Library

Math Operations

Description



The MinMax block outputs either the minimum or the maximum element or elements of the inputs. You can choose the function to apply by selecting one of the choices from the **Function** parameter list.

If the block has one input port, the input must be a scalar or a vector. The block outputs a scalar equal to the minimum or maximum element of the input vector.

If the block has multiple input ports, the nonscalar inputs must all have the same dimensions. The block expands any scalar inputs to have the same dimensions as the nonscalar inputs. The block outputs a signal having the same dimensions as the input. Each output element equals the minimum or maximum of the corresponding input elements.

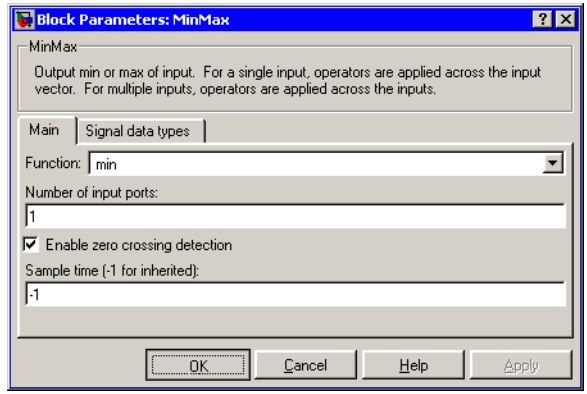
Data Type Support

The MinMax block accepts and outputs real signals of any data type supported by Simulink, except `Boolean`. The MinMax block supports fixed-point data types.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box

The **Main** pane of the MinMax block dialog appears as follows:



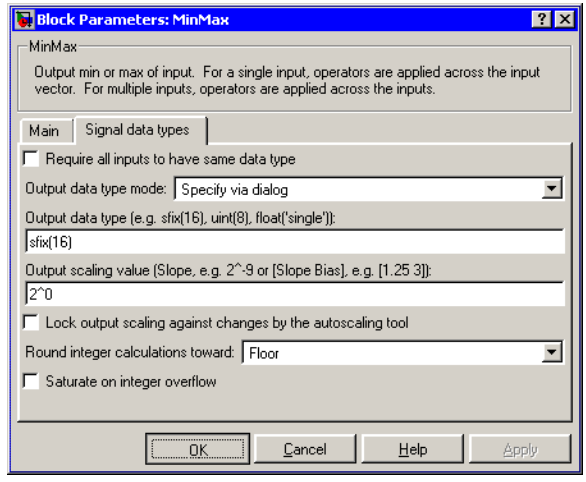
Function
: Specify whether to apply the function `min` or `max` to the input.

Number of input ports
: Specify the number of inputs to the block.

Enable zero crossing detection
: Select to enable zero crossing detection to detect minimum and maximum values. For more information, see "Zero Crossing Detection" in the Using Simulink documentation.

Sample time (-1 for inherited)
: Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See “Specifying Sample Time” in the online documentation for more information.

The **Signal data types** pane of the MinMax block dialog appears as follows:



Require all inputs to have same data type

Select this parameter to require that all inputs must have the same data type.

Output data type mode

Specify the output data type and scaling by choosing a built-in data type from the drop-down list, or inherit the data type and scaling by an internal rule or by backpropagation. Lastly, if you select `Specify via dialog`, the **Output data type**, **Output scaling value**, and **Lock output scaling against changes by the autoscaling tool** parameters become visible.

Output data type

Specify any data type, including fixed-point data types. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Output scaling value

Set the output scaling using binary point-only or [Slope Bias] scaling. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Lock output scaling against changes by the autoscaling tool
Select to lock scaling of outputs. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Round integer calculations toward
Select the rounding mode for fixed-point operations.

Saturate on integer overflow
Select to have overflows saturate.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of the inputs
Dimensionalized	Yes
Zero Crossing	Yes, if enabled.

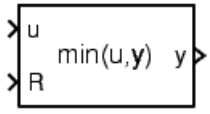
Purpose

Determine the minimum or maximum of a signal over time

Library

Math Operations

Description



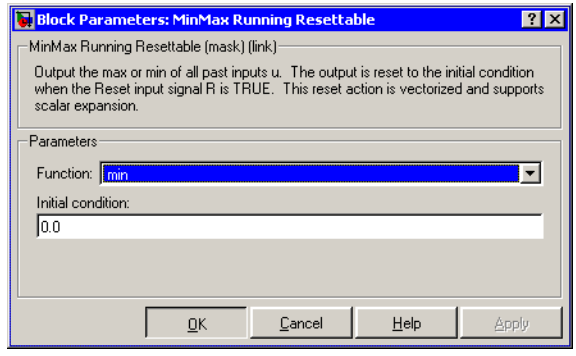
The MinMax Running Resettable block outputs the minimum or maximum of all past inputs u. You specify whether the block outputs the minimum or the maximum with the **Function** parameter.

The block can reset its state based on an external reset signal R. When the reset signal R is TRUE, the block resets the output to the value of the **Initial condition** parameter.

Data Type Support

The MinMax Running Resettable block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Function
Specify whether the block outputs the minimum or the maximum.

Initial condition
Initial condition.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

Model

Purpose

Include a model as a block in another model.

Library

Ports & Subsystems

Description

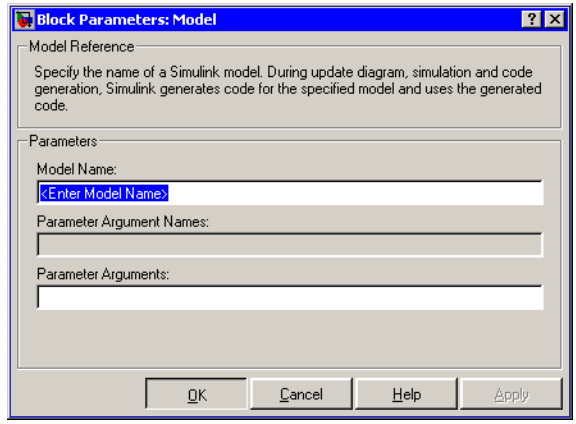
The Model block allows you to include a model as a block in another model. The Model block displays input ports and output ports corresponding to the model's top-level input and output ports. This allows you to connect the included model to other blocks in the containing model.

During simulation, Simulink invokes an S-function called the simulation target to compute the model's outputs. If the simulation target does not exist at the beginning of a simulation or when you update the model's block diagram, Simulink generates the target from the referenced model. If the target exists, Simulink checks whether the included model has changed since the last time the target was built. If so, Simulink regenerates the target to reflect changes in the model. Simulink uses the same simulation target for all instances of an included model whether in the same model or different model. See "Referencing Models" for more information.

Data Type Support

Determined by the root-level inputs and outputs of the model referenced by the Model block.

Parameters and Dialog Box



Model Name

Name of the model referenced by this block. This name must be a valid MATLAB identifier. The model must exist on the MATLAB path and the MATLAB path must contain no other model having the same name.

Parameter Argument Names

Names of parameter arguments accepted by the model referenced by this block (see “Parameterizing Model References” for more information).

Parameter Arguments

Values to be passed as parameter arguments to the model referenced by this block each time the model is invoked during a simulation. Enter the values in this field as a comma-separated list in the same order as the corresponding argument names appear in the **Parameter Argument Names** field.

Characteristics

Direct Feedthrough	Depends on model referenced by this block.
Scalar Expansion	Depends on model referenced by this block.

Model Info

Purpose

Display revision control information in a model

Library

Model-Wide Utilities

Description

The Model Info block displays revision control information about a model as an annotation block in the model's block diagram. The following diagram illustrates use of a Model Info block to display information about the `vdp` model.

A Model Info block can show revision control information embedded in the model itself and/or information maintained by an external revision control or configuration management system. A Model Info block's

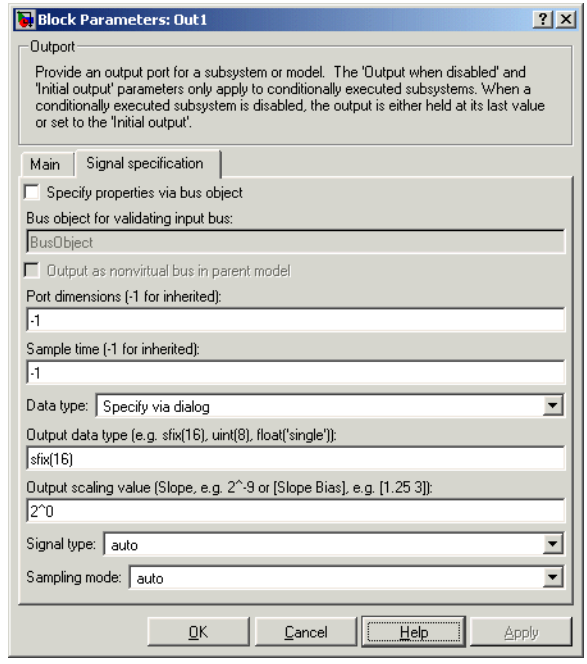
Output when disabled

This option is enabled only if the Outport resides in an Enabled Subsystem. It specifies what happens to the block output when the system is disabled.

Initial output

For conditionally executed subsystems, specify the block output before the subsystem executes and while it is disabled. You can specify `[]` if your model does not depend on the initial output of the conditionally executed subsystem. Simulink does not allow the initial output of this block to be `inf` or `NaN`.

The **Signal specification** pane of the Output block dialog appears as follows:



Specify properties via bus object
Select this option to use a bus object (see "Working with Data Objects" and `Simulink.Bus` class in the online documentation) to define the properties of a bus connected to this Outport block.

Bus object for validating input bus
Specifies the name of the bus object that defines the structure of that a bus must have to be connected to this Outport block. At the beginning of a simulation or when you update the model's diagram, Simulink checks whether the bus connected to this block has the specified structure. If not, Simulink displays an error message.

Output as structure in parent model
Select this option if you want code generated from this model to use a C structure to define the structure of the bus signal output by this block.

Port dimensions (-1 for inherited)
Specifies the dimensions that a signal must have in order to be connected to this Outport block. Valid values are:

`-1`	A signal of any dimensions can be connected to this port.
`N`	The signal connected to this port must be a vector of size `N`.
`[R C]`	The signal connected to this port must be a matrix having `R` rows and `C` columns.

Sample time (-1 for inherited)
Specify the sample time of this Outport block. See "Specifying Sample Time" in the online documentation for information on specifying sample times. The output of this block changes at the specified rate to reflect the value of its input.

Data type
Specify the data type of the signal output by this block. To output any data type, set this parameter to `auto`.

Output data type
Specify any data type, including fixed-point data types. This parameter is only visible if you select `Specify via dialog` for the **Data type** parameter.

Output scaling value
Set the output scaling using binary point-only or [Slope Bias] scaling. This parameter is only visible if you select `Specify via dialog` for the **Data type** parameter.

Signal type
Specifies the numeric type of the signal output by this block. The options are:

`real`	This Outport block outputs a real-valued signal. The signal connected to this block must be real. If it is not, Simulink displays an error if you try to update the diagram or simulate the model that contains this block.
`complex`	This Outport block outputs a complex signal. The signal connected to this block must be complex. If it is not, Simulink displays an error if you try to update the diagram or simulate the model that contains this block.
`auto`	This block outputs the numeric type of the signal that is connected to its input.

Sampling mode
Specify the sampling mode (`Sample based` or `Frame based`) that the input signal must match. To accept any sampling mode, set this parameter to `auto`. This parameter is intended to support signal processing applications based on Simulink. See the documentation for the `buffer` function provided by the Signal Processing Toolbox or "Frame-Based Operations" in the documentation for the Signal Processing Blockset for information about frame-based signals.

Characteristics

Sample Time	Inherited from driving block
Dimensionalized	Yes

Purpose

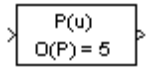
Perform evaluation of polynomial coefficients on input values

Library

Math Operations

Description

The Polynomial block uses a coefficients parameter to evaluate a real polynomial for the input value.

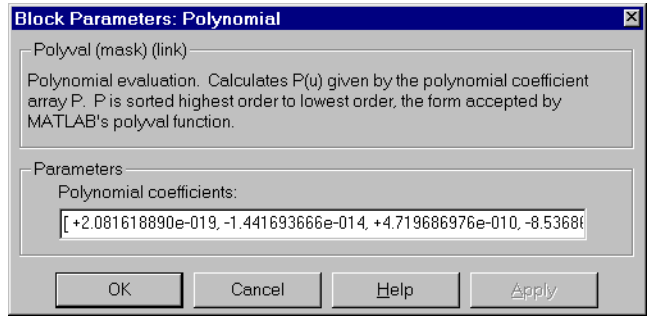


You define a set of polynomial coefficients in the form accepted by the MATLAB `polyval` command. The block then calculates *P(u)* at each time step for the input *u*. Inputs and coefficients must be real.

Data Type Support

The Polynomial block accepts real signals of types `double` or `single`. The **Polynomial coefficients** parameter must be of the same type as the inputs. The output data type is set to the input data type.

Parameters and Dialog Box



Polynomial coefficients

Values are in coefficients of a polynomial in MATLAB `polyval` form, with the first coefficient representing x^N, then decreasing in order until the last coefficient, which represents the constant for the polynomial. See `polyval` in the MATLAB documentation for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block

Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

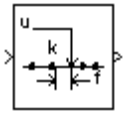
Purpose

First stage of high-performance constant or linear interpolation that performs index search and interval fraction calculation for input on a breakpoint set

Library

Lookup Tables

Description



The PreLookup Index Search block calculates the indices and interval fractions for the input value in the **Breakpoint data** parameter. By using this combination of blocks, you can replace multiple Interpolation (n-D) blocks with one set of PreLookup Index Search blocks. In models that have many interpolation blocks simulation performance can be greatly increased.

To use this block, you must define a set of breakpoint values. In normal use, this breakpoint data set corresponds to one dimension of a **Table data** parameter in an Interpolation (n-D) using PreLookup block. The block generates a pair of outputs for each input value by calculating the index of the breakpoint set element that is less than or equal to the input value and the resulting fractional value that is a number $0 \le f < 1$ that represents the input value's normalized position between the index and the next index value for in-range input.

For example, if the breakpoint data is

```
[ 0 5 10 20 50 100 ]
```

and the input value `u` is 55, the (index, fraction) pair is (4, 0.1), denoted as `k` and `f` on the block. Note that the index value is zero-based.

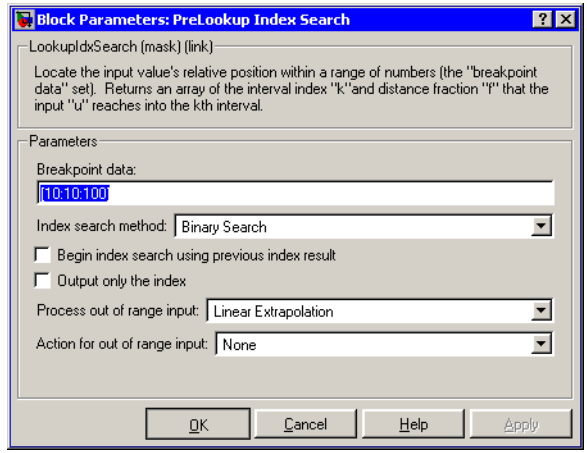
Note The interval fraction can be negative or greater than `1` for out-of-range input. See the documentation for the block's **Process out of range input parameter** for more information.

Data Type Support

The PreLookup Index Search block accepts signals of types `double` or `single`, but for any given block the inputs must all be of the same type.

The **Breakpoint data** parameter must be of the same type as the inputs. The output data type is set to the input data type.

Parameters and Dialog Box



Breakpoint data

The set of numbers to search.

Index search method

Binary search, evenly spaced points, or linear search. Use linear search in combination with **Begin index search using previous index result** for higher performance than a binary search when the input values do not change much from one time step to the next. For large breakpoint sets, a linear search can be very slow if the input value changes by more than a few intervals from one time step to the next.

Begin index search using previous index result

Select this option if you want the block to start its search using the index that was found on the previous time step. For inputs that change slowly with respect to the interval size, you can realize a large performance gain.

Output only the index

If this block is not being used to feed an Interpolation (n-D) Using PreLookup block, the interval fraction output can be dropped and the resulting index value output is a `uint32` instead.

Process out of range input

Specifies how to handle out-of-range input. Options include:

- `Clip to Range`

  If the input is less than the first breakpoint, return the index of the first breakpoint (i.e., `0`) and `0` for the interval fraction. If the input is greater than the last breakpoint, return the index of the next-to-the-last breakpoint and `1` for the interval fraction. For example, suppose the range is `[1 2 3]` and you select this option. Then, if the input is `0.5`, the block returns `[0 0]`; if the input is `3.5`, the block returns `[1 1]`.

- `Linear Extrapolation`

  If the input is less than the first breakpoint, return the index of the first breakpoint and an interval fraction representing the linear distance from the input to the first breakpoint. If the input is greater than the last breakpoint, return the index of the next-to-the-last breakpoint and an interval fraction that represents the linear distance from the next-to-the-last breakpoint to the input. For example, suppose the range is `[1 2 3]` and you select this option. Then, if the input is `0.5`, the block returns `[0 -0.5]`; if the input is `3.5`, the block returns `[1 1.5]`.

Action for out of range input

Specifies whether to produce a warning or error message if the input is out of range. The options are `None` (the default, no warning or error message), `Warning` (display a warning message in the MATLAB command window and continue the simulation), `Error` (halt the simulation and display an error message in the Simulation Diagnostics Viewer).

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving blocks
Scalar Expansion	Yes
Dimensionalized	Yes
Zero Crossing	No

Purpose

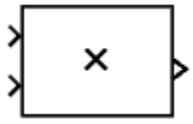
Multiply or divide inputs

Library

Math Operations

Description

The Product block performs multiplication or division of its inputs.



This block produces outputs using either element-wise or matrix multiplication, depending on the value of the **Multiplication** parameter. You specify the operations with the **Number of inputs** parameter. Multiply(`*`) and divide(`/`) characters indicate the operations to be performed on the inputs:

- If there are two or more inputs, then the number of characters must equal the number of inputs. For example, "`*/*`" requires three inputs. For this example, if the **Multiplication** parameter is set to `Element-wise`, the block divides the elements of the first (top) input by the elements of the second (middle) input, and then multiplies by the elements of the third (bottom) input. In this case, all nonscalar inputs to this block must have the same dimensions.

  If, however, the **Multiplication** parameter is set to `Matrix`, the block output is the matrix product of the inputs marked "`*`" and the inverse of inputs marked "`/`", with the order of operations following the entry in the **Number of inputs** parameter. The dimensions of the inputs must be such that the matrix product is defined.

  Note To perform a dot product on input vectors, use the Dot Product block.

- If only multiplication of inputs is required, then a numeric parameter value equal to the number of inputs can be supplied instead of "`*`" characters. This may be used in conjunction with either element-wise or matrix multiplication.
- If a single vector is input and the **Multiplication** parameter is set to `Element-wise`, then a single "`*`" will cause the block to output the

scalar product of the vector elements. A single "/" will cause the block to output the inverse of the scalar product of the vector elements.

- If a single matrix is input and the **Multiplication** parameter is set to `Element-wise`, then a single "*" or "/" will cause the block to error out. If, however, the **Multiplication** parameter is set to `Matrix`, then a single "*" will cause the block to output the matrix unchanged, and a single "/" will cause the block to output the inverse of the matrix.

The Product block first performs the specified multiply or divide operations on the inputs, and then converts the results to the output data type using the specified rounding and overflow modes.

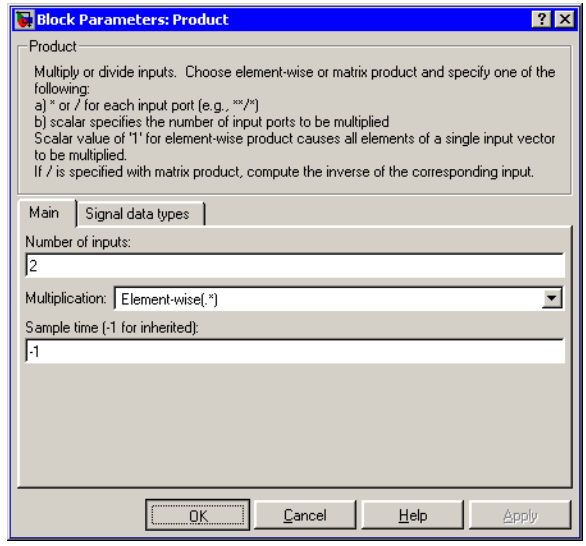
Data Type Support

The Product block accepts real or complex signals of any data type supported by Simulink including fixed-point data types.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box

The **Main** pane of the Product dialog appears as follows:



Number of inputs

Enter the number of inputs or a combination of "*" and "/" symbols. See Description above for a complete discussion of this parameter.

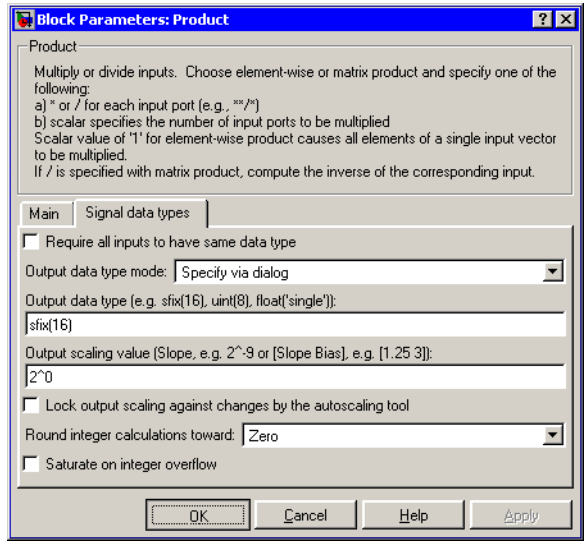
Multiplication

Specify element-wise or matrix multiplication. See Description above for a complete discussion of this parameter.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See “Specifying Sample Time” in the online documentation for more information.

The **Signal data types** pane of the Product dialog appears as follows:



Require all inputs to have same data type

Select this parameter to require that all inputs have the same data type.

Output data type mode

Specify the output data type and scaling to be the same as the first input, or inherit the data type and scaling by an internal rule or by backpropagation. You can also choose a built-in data type from the drop-down list. Lastly, if you choose `Specify via dialog`, the **Output data type**, **Output scaling value**, and **Lock output scaling against changes by the autoscaling tool** parameters become visible.

If you select `Inherit via internal rule` for this parameter, Simulink chooses a combination of output scaling and data type that requires the smallest amount of memory consistent with accommodating the output range and maintaining the output precision (and avoiding underflow in the case of division operations). If the **Device type** parameter on the **Hardware**

Implementation pane of the **Configuration Parameters** dialog is set to `custom`, Simulink chooses the data type without regard to hardware constraints. Otherwise, Simulink chooses the smallest available hardware data type capable of meeting range, precision, and underflow constraints. For example, if the block multiplies inputs of type `int8` and `int16` and `custom` is specified as the device type, the output data type is `sfix24`. If `Unspecified (assume 32-bit generic)` is specified, the output data type is `int32`. If none of the word lengths provided by the target hardware can accommodate the output range, Simulink displays an error message in the Simulation Diagnostics Viewer.

Output data type
Specify any data type, including fixed-point data types. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Output scaling value
Set the output scaling using binary point-only or [Slope Bias] scaling. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Lock output scaling against changes by the autoscaling tool
Select to lock scaling of outputs. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Round integer calculations toward
Select the rounding mode for fixed-point operations.

Saturate on integer overflow
Select to have overflows saturate.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

Dimensionalized	Yes
Zero Crossing	No

Purpose

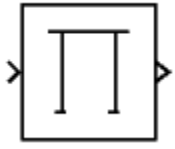
Multiply or divide inputs

Library

Math Operations

Description

The Product of Elements block is an implementation of the Product block. See `Product` for more information.



Probe

Purpose

Output a signal's attributes, including width, dimensionality, sample time, and/or complex signal flag

Library

Signal Attributes

Description

The Probe block outputs selected information about the signal on its input. The block can output the input signal's width, dimensionality, sample time, and/or a flag indicating whether the input is a complex-valued signal. The block has one input port. The number of output ports depends on the information that you select for probing, that is, signal dimensionality, sample time, and/or complex signal flag. Each probed value is output as a separate signal on a separate output port. The block accepts real or complex-valued signals of any built-in data type. It outputs signals of type `double`. During simulation, the block's icon displays the probed data.

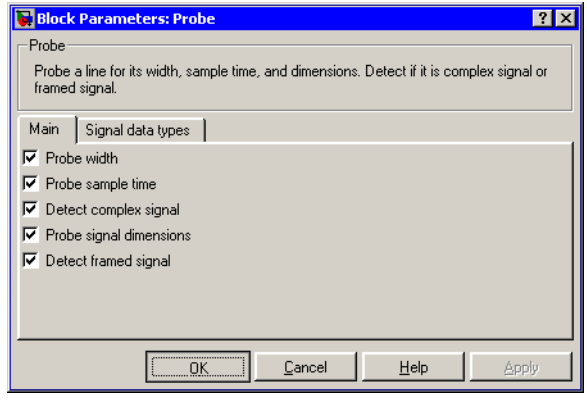
Data Type Support

The Probe block accepts and outputs any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see"Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box

The **Main** pane of the Probe block dialog appears as follows:



Probe width
Select to output the width, or number of elements, of the probed signal.

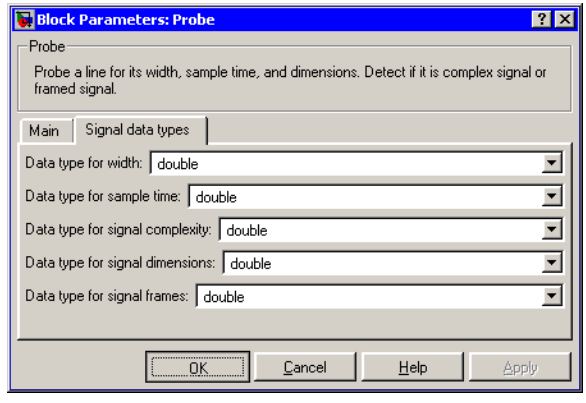
Probe sample time
Select to output the sample time of the probed signal. The output is a 2x1 vector that specifies the period and offset of the sample time, respectively. See “Specifying Sample Time” for more information.

Probe complex signal
Select to output 1 if the probed signal is complex; otherwise, 0.

Probe signal dimensions
Select to output the dimensions of the probed signal.

Detect framed signal
Select to output 1 if the probed signal is framed; otherwise, 0.

The **Signal data types** pane of the Probe block dialog appears as follows:



Note The Probe block ignores the **Data Type Override** setting of the Fixed-Point Settings interface.

Data type for width
Select the output data type for the width information.

Data type for sample time
Select the output data type for the sample time information.

Data type for signal complexity
Select the output data type for the complexity information.

Data type for signal dimensions
Select the output data type for the dimensions information.

Data type for signal frames
Select the output data type for the frames information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes
Dimensionalized	Yes
Zero Crossing	No

Purpose

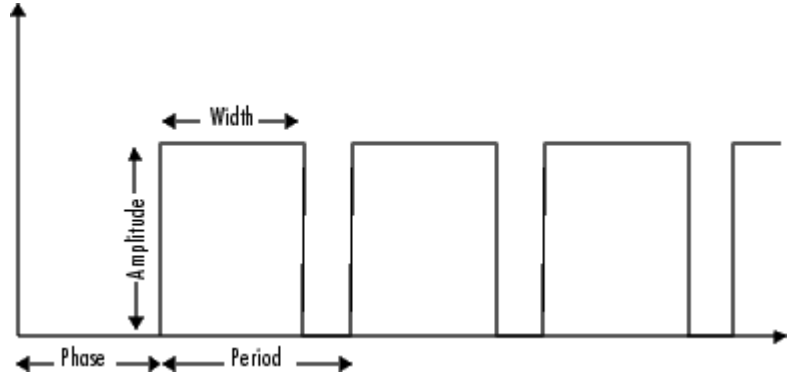
Generate square wave pulses at regular intervals

Library

Sources

Description

The Pulse Generator block generates square wave pulses at regular intervals. The block's waveform parameters, **Amplitude**, **Pulse Width**, **Period**, and **Phase Delay**, determine the shape of the output waveform. The following diagram shows how each parameter affects the waveform.



The Pulse Generator can emit scalar, vector, or matrix signals of any real data type. To cause the block to emit a scalar signal, use scalars to specify the waveform parameters. To cause the block to emit a vector or matrix signal, use vectors or matrices, respectively, to specify the waveform parameters. Each element of the waveform parameters affects the corresponding element of the output signal. For example, the first element of a vector amplitude parameter determines the amplitude of the first element of a vector output pulse. All the waveform parameters must have the same dimensions after scalar expansion. The data type of the output is the same as the data type of the **Amplitude** parameter.

The block's **Pulse type** parameter allows you to specify whether the block's output is time-based or sample-based. If you select *sample-based*, the block computes its outputs at fixed intervals that you

specify. If you select *time-based*, Simulink computes the block's outputs only at times when the output actually changes. This can result in fewer computations being required to compute the block's output over the simulation time period.

Depending on the pulse's waveform characteristics, the intervals between changes in the block's output can vary. For this reason, a time-based Pulse Generator block is said to have a variable sample time. Simulink uses yellow as the sample time color of such blocks (see "Displaying Sample Time Colors" for more information).

Simulink cannot use a fixed solver to compute the output of a time-based pulse generator. If you specify a fixed-step solver for models that contain time-based pulse generators, Simulink computes a fixed sample time for the time-based pulse generators. It then simulates the time-based pulse generators as sample-based.

Note If you use a fixed-step solver and the **Pulse type** is `time-based`, you must choose the step size such that the period, phase delay, and pulse width (in seconds) are integer multiples of the step size. For example, suppose that the period is `4` seconds, the pulse width is `75%` (i.e., `3` s), and the phase delay is `1` s. In this case, the computed sample time is `1` s. Therefore, you must choose a fixed-step size that is `1` or that divides `1` exactly (e.g., `0.25`). You can guarantee this by setting the fixed-step solver's step size to `auto` on the **Configuration Parameters** dialog box.

If you select time-based as the block's pulse type, you must specify the pulse's phase delay and period in units of seconds. If you specify sample-based, you must specify the block's sample time in seconds, using the **Sample Time** parameter, then specify the block's phase delay and period as integer multiples of the sample time. For example, suppose that you specify a sample time of 0.5 second. And suppose you want the pulse to repeat every two seconds. In this case, you would specify 4 as the value of the block's **Period** parameter.

Data Type Support

The Pulse Generator block outputs real signals of any data type supported by Simulink. The data type of the output signal is the same as that of the **Amplitude** parameter.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Initial output
The initial value of the signal. The default is `0`.

Interpret vector parameters as 1-D
If you select this option and the other parameters are one-row or one-column matrices, after scalar expansion, the block outputs a 1-D signal (vector). Otherwise, the output dimensionality is the same as that of the other parameters.

Characteristics

Sample Time	Inherited from driven block
Scalar Expansion	Yes
Dimensionalized	Yes
Zero Crossing	Yes

Random Number

Purpose

Generate normally distributed random numbers

Library

Sources

Description

The Random Number block generates normally distributed random numbers. The seed is reset to the specified value each time a simulation starts.

By default, the sequence produced has a mean of 0 and a variance of 1, although you can vary these parameters. The sequence of numbers is repeatable and can be produced by any Random Number block with the same seed and parameters. To generate a vector of random numbers with the same mean and variance, specify the **Initial seed** parameter as a vector.

To generate uniformly distributed random numbers, use the Uniform Random Number block.

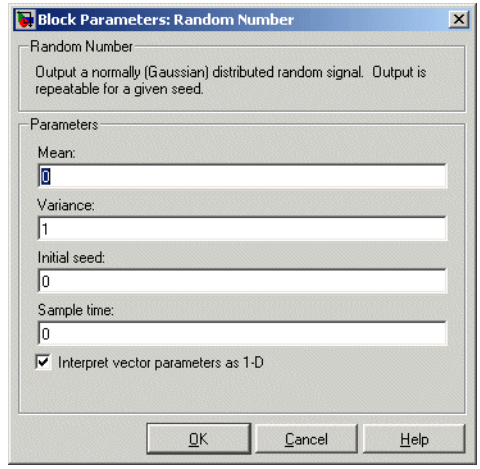
Avoid integrating a random signal, because solvers are meant to integrate relatively smooth signals. Instead, use the Band-Limited White Noise block.

All the block's numeric parameters must be of the same dimension after scalar expansion.

Data Type Support

The Random Number block accepts and outputs signals of type `double`.

Parameters and Dialog Box



Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in the online Simulink documentation for details.

Mean

The mean of the random numbers. The default is `0`.

Variance

The variance of the random numbers. The default is `1`.

Initial seed

The starting seed for the random number generator. The seed must be 0 or a positive integer. The default is `0`.

Sample time

The time interval between samples. The default is `0`, causing the block to have continuous sample time. See “Specifying Sample Time” in the online documentation for more information.

Interpret vector parameters as 1-D

If you select this option and the other parameters are one-row or one-column matrices, after scalar expansion, the block outputs a 1-D signal (vector). Otherwise, the output dimensionality is the same as that of the other parameters.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of parameters
Dimensionalized	Yes
Zero Crossing	No

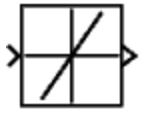
Purpose

Limit the rate of change of a signal

Library

Discontinuities

Description



The Rate Limiter block limits the first derivative of the signal passing through it. The output changes no faster than the specified limit. The derivative is calculated using this equation.

$$rate = \frac{u(i) - y(i-1)}{t(i) - t(i-1)}$$

u(i) and *t(i)* are the current block input and time, and *y(i-1)* and *t(i-1)* are the output and time at the previous step. The output is determined by comparing *rate* to the **Rising slew rate** and **Falling slew rate** parameters:

- If *rate* is greater than the **Rising slew rate** parameter (*R*), the output is calculated as

  $$y(i) = \Delta t \cdot R + y(i-1)$$

- If *rate* is less than the **Falling slew rate** parameter (*F*), the output is calculated as

  $$y(i) = \Delta t \cdot F + y(i-1)$$

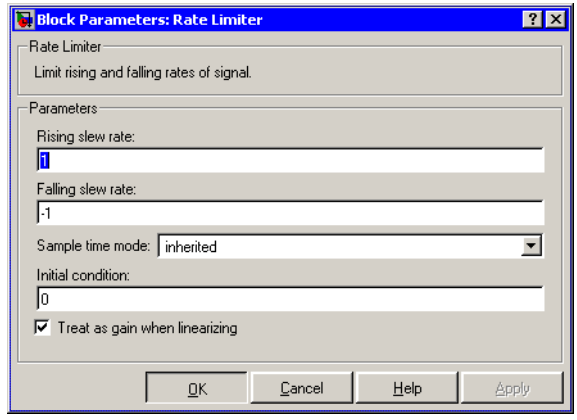
- If *rate* is between the bounds of *R* and *F*, the change in output is equal to the change in input:

  $$y(i) = u(i)$$

Data Type Support

The Rate Limiter block accepts and outputs signals of any data type supported by Simulink, except `Boolean`. The Rate Limiter block supports fixed-point data types.

Parameters and Dialog Box



Rising slew rate

Specify the limit of the derivative of an increasing input signal. This parameter is tunable for fixed-point inputs.

Falling slew rate

Specify the limit of the derivative of a decreasing input signal. This parameter is tunable for fixed-point inputs.

Sample time mode

Specify the sample time mode, `continuous` or `inherited` from the driving block.

Initial condition

Set the initial output of the simulation. Simulink does not allow you to set the initial condition of this block to `inf` or `NaN`.

Treat as gain when linearizing

Linearization commands in Simulink treat this block as a gain in state space. Select this check box to cause the linearization commands to treat the gain as 1; otherwise, the commands treat the gain as 0.

Characteristics

Direct Feedthrough	Yes
Sample Time	Continuous or inherited (specified in the **Sample time mode** parameter)
Scalar Expansion	Yes, of input and parameters
Dimensionalized	Yes
Zero Crossing	No

See Also

Rate Limiter Dynamic

Rate Limiter Dynamic

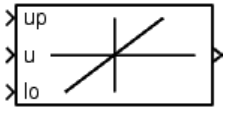
Purpose

Limit the rising and falling rates of the signal

Library

Discontinuities

Description



The Rate Limiter Dynamic block limits the rising and falling rates of the signal.

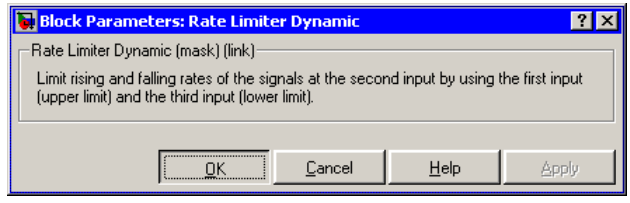
The external signal `up` sets the upper limit on the rising rate of the signal.

The external signal `lo` sets the lower limit on the falling rate of the signal.

Data Type Support

The Rate Limiter Dynamic block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also

Rate Limiter

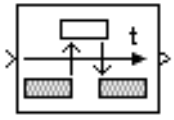
Purpose

Handle transfer of data between blocks operating at different rates

Library

Signal Attributes

Description



The Rate Transition block transfers data from the output of a block operating at one rate to the input of another block operating at a different rate. The Rate Transition block's parameters allows you to specify options that trade data integrity and deterministic transfer for faster response and/or lower memory requirements.

Note See "Data Transfer Problems" in the online Real-Time Workshop documentation for a discussion of data integrity and deterministic data transfer.

In particular, the block supports the following options:

- Deterministic transfer of data with data integrity between blocks operating at different speeds at the cost of maximum latency of data transfer This is the default option.
- Nondeterministic data transfer with minimum latency and assured data integrity but increased memory requirements

  To specify this option, check the **Ensure data integrity during data transfer** parameter and uncheck the **Ensure deterministic data transfer** parameter.
- Minimum latency and target size at the cost of nondeterministic data transfer and possible loss of data integrity

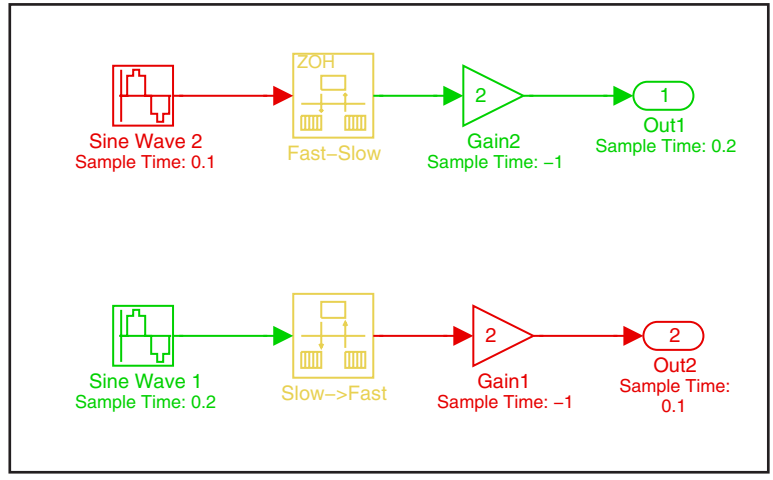
  To specify this option, uncheck the **Ensure data integrity during data transfer** and **Ensure deterministic data transfer** parameters.

The behavior of the Rate Transition block depends on the sample times of the ports between which it is connected, the priorities of the tasks corresponding to the source and destination sample times (see

"Sample time properties"), and whether the model specifies a fixed- or variable-step solver. Updating the model causes a label to appear on the block that indicates its behavior during simulation as follows:

Label	Block Behavior
`ZOH`	Acts as a zero-order hold
`1/z`	Acts as a unit delay
`Buf`	Copies input to output under semaphore control
`Db_buf`	Copies input to output, using double buffers
`Copy`	Unprotected copy of input to output
`NoOp`	Does nothing

The behavior label lets you see at a glance the method that the Rate Transition block uses to ensure safe transfer of data between tasks operating at different rates. You can use Simulink's sample-time colors feature (see "Displaying Sample Time Colors") to display the relative rates that the block bridges. Consider, for example, the following diagram.



Sample-time colors and the block behavior label allow you to see at a glance that the Rate Transition block at the top of the diagram acts as a zero-order hold in a fast-to-slow transition and the bottom Rate Transition block acts as a unit delay in a slow-to-fast transition.

See “Sample Rate Transitions” in the online Real-Time Workshop documentation for more information.

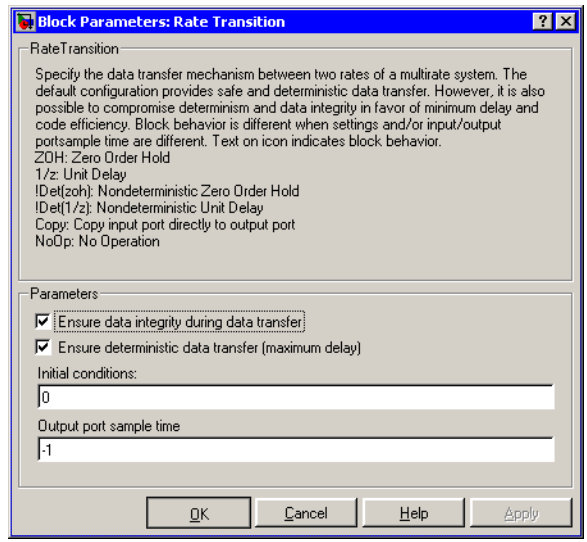
Note The Zero-Order Hold and Unit Delay blocks also enable transfer of data between blocks operating at different rates. However, you should use the Rate Transition block for this purpose because it offers a wider range of options and is easier to use.

Data Type Support

The Rate Transition block accepts signals of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box



Ensure data integrity during data transfer

Selecting this option results in generation of code that ensures the integrity of data transferred by the Rate Transition block. If you select this option and the transfer is nondeterministic (see **Ensure deterministic data transfer** option below), the generated code uses double-buffering to prevent the fast block from interrupting the data transfer. Otherwise the generated code uses a copy operation to effect the data transfer. The copy operation consumes less memory than double-buffering but is also interruptible and hence can lead to loss of data during nondeterministic data transfers. Thus, you should select this option if you want the generated code to operate both with maximum responsiveness (i.e., nondeterministically) and assured data integrity. See "Rate Transition Block Options" in the online Real-Time Workshop documentation for more information.

Ensure deterministic data transfer (maximum delay)

Selecting this option causes code generation to generate code that transfers data at the sample rate of the slower block, i.e., deterministically. If this option is not selected, data transfers occur as soon as new data is available from the source block and the receiving block is ready to receive the data. This avoids the need to delay transfers, thus ensuring that the system operates with maximum responsiveness. However, it also means that transfers can occur unpredictably, which is undesirable in some applications. See "Rate Transition Block Options" in the online Real-Time Workshop documentation for more information.

Initial conditions

This parameter applies only to `Slow to fast` transitions. It specifies the Rate Transition's initial output at the beginning of a transition when there is not yet any output from the slow block connected to the Rate Transition block's input. Simulink does not allow the initial output of this block to be `inf` or `NaN`.

Output port sample time

Specifies the output rate to which the input rate is converted. The default value (`-1`) specifies that the output rate is inherited

from the block to which the Rate Transition block's output port is connected. See "Specifying Sample Time" in the online documentation for information on how to specify the output rate.

Characteristics

Direct Feedthrough	No for slow-to-fast transitions that are protected, i.e., for which you have checked the **Ensure data integrity during data transfer** option; otherwise, yes.
Sample Time	This block supports discrete-to-discrete and discrete-to-continuous transitions.
Scalar Expansion	Yes, of input.
Dimensionalized	Yes
Zero Crossing	No

Real-Imag to Complex

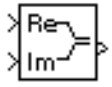
Purpose

Convert real and/or imaginary inputs to a complex signal

Library

Math Operations

Description



The Real-Imag to Complex block converts real and/or imaginary inputs to a complex-valued output signal.

The inputs can both be arrays (vectors or matrices) of equal dimensions, or one input can be an array and the other a scalar. If the block has an array input, the output is a complex array of the same dimensions. The elements of the real input are mapped to the real parts of the corresponding complex output elements. The imaginary input is similarly mapped to the imaginary parts of the complex output signals. If one input is a scalar, it is mapped to the corresponding component (real or imaginary) of all the complex output signals.

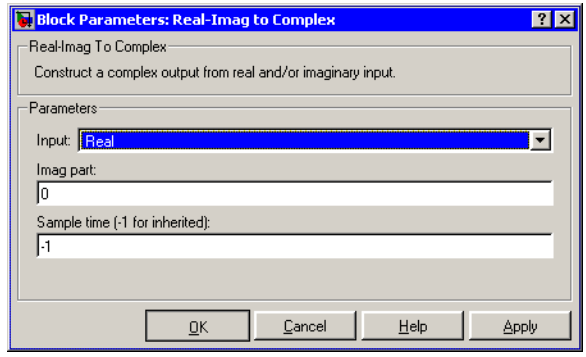
The input signals and real or imaginary output parameter can be of any data type supported by Simulink, except `Boolean`. The Real-Imag to Complex block supports fixed-point data types. The output is of the same type as the input or parameter that determines the output.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Data Type Support

See the preceding description.

Parameters and Dialog Box



Input

Specifies the kind of input: a real input, an imaginary input, or both.

Real (Imag) part

If the input is a real-part signal, this parameter specifies the constant imaginary part of the output signal. If the input is the imaginary part, this parameter specifies the constant real part of the output signal. Note that the title of this field changes to reflect its usage.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See “Specifying Sample Time” in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes, of the input when the function requires two inputs
Dimensionalized	Yes
Zero Crossing	No

Relational Operator

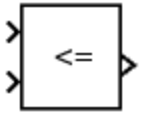
Purpose

Perform the specified relational operation on the inputs

Library

Logic and Bit Operations

Description



The Relational Operator block performs the specified comparison of its two inputs.

You select the relational operator connecting the two inputs with the **Relational Operator** parameter. The block updates to display the selected operator. The supported operations are given below.

Operation	Description
==	TRUE if the first input is equal to the second input
~=	TRUE if the first input is not equal to the second input
<	TRUE if the first input is less than the second input
<=	TRUE if the first input is less than or equal to the second input
>=	TRUE if the first input is greater than or equal to the second input
>	TRUE if the first input is greater than the second input

You can specify inputs as scalars, arrays, or a combination of a scalar and an array:

- For scalar inputs, the output is a scalar.
- For array inputs, the output is an array of the same dimensions, where each element is the result of an element-by-element comparison of the input arrays.
- For mixed scalar/array inputs, the output is an array, where each element is the result of a comparison between the scalar and the corresponding array element.

The input with the smaller positive range is converted to the data type of the other input offline using round-to-nearest and saturation. This conversion is performed prior to comparison.

The output data type is specified with the **Output data type mode** and **Output data type** parameters. The output equals 1 for TRUE and 0 for FALSE.

Note The output data type selected should represent zero exactly. Data types that satisfy this condition include signed and unsigned integers and any floating-point data type.

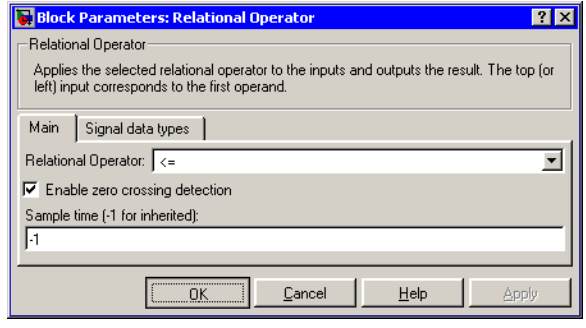
Data Type Support

The Relational Operator block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types. However, if the **Output data type mode** parameter is set to `Logical`, the input may only be `Boolean` or `double`. One input can be real and the other complex if the operator is `==` or `!=`.

For a discussion on the data types supported by Simulink, see"Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box

The **Main** pane of the Relational Operator block appears as follows:



Relational Operator
Designate the relational operator used to compare the two inputs.

Enable zero crossing detection
Select to enable zero crossing detection. For more information, see "Zero Crossing Detection" in the Using Simulink documentation.

Sample time (-1 for inherited)
Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See “Specifying Sample Time” in the online documentation for more information.

The **Signal data types** pane of the Relational Operator block appears as follows:

Purpose

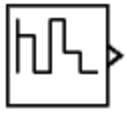
Output and repeat the discrete time sequence

Library

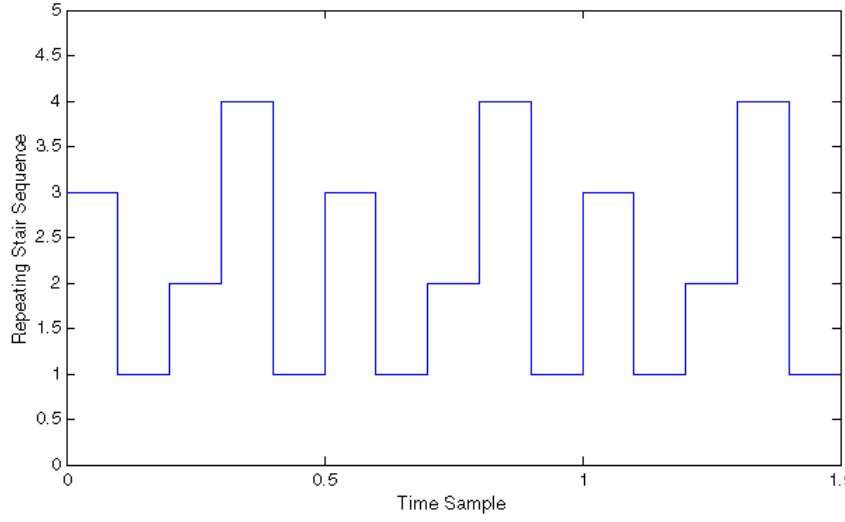
Sources

Description

The Repeating Sequence Stair block outputs and repeats a discrete time sequence.



You can specify the stair sequence with the **Vector of output values** parameter. For example, the vector can be specified as `[3 1 2 4 1]'`, producing the stair sequence shown in the plot.



You can specify the sample time with the **Sample time** parameter.

You can select the output data type and scaling with the **Output data type and scaling** parameter, and set the output data type with the **Output data type** parameter.

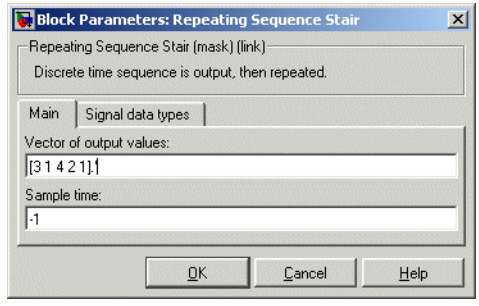
For fixed-point data types, you can set the output scaling with the **Output scaling** parameter, and, below that parameter, select the method for scaling the output with the **Output scaling** parameter.

Data Type Support

The Repeating Sequence Stair block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box

The **Main** pane of the Repeating Sequence Stair block dialog appears as follows:



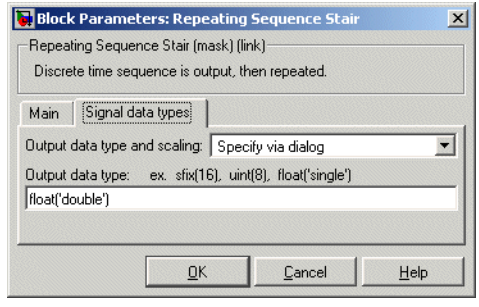
Vector of output values

Vector containing values of the repeating stair sequence.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See “Specifying Sample Time” in the online documentation for more information.

The **Signal data types** pane of the Repeating Sequence Stair block dialog appears as follows:



Output data type and scaling
Specify the output data type and scaling via the dialog box, or by inheriting the data type and scaling by backpropagation.

Output data type
Specify any data type supported by the blockset.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No

See Also

Repeating Sequence, Repeating Sequence Interpolated

Reshape

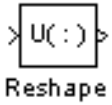
Purpose

Change the dimensionality of a signal

Library

Math Operations

Description



The Reshape block changes the dimensionality of the input signal to a dimensionality that you specify, using the block's **Output dimensionality** parameter. For example, you can use the block to change an N-element vector to a 1-by-N or N-by-1 matrix signal, and vice versa.

The **Output dimensionality** parameter lets you select any of the following output options.

Output Dimensionality	Description
1-D array	Converts a matrix (2-D array) to a vector (1-D array) array signal. The output vector consists of the first column of the input matrix followed by the second column, etc. (This option leaves a vector input unchanged.)
Column vector	Converts a vector or matrix input signal to a column matrix, i.e., an M-by-1 matrix, where M is the number of elements in the input signal. For matrices, the conversion is done in column-major order.

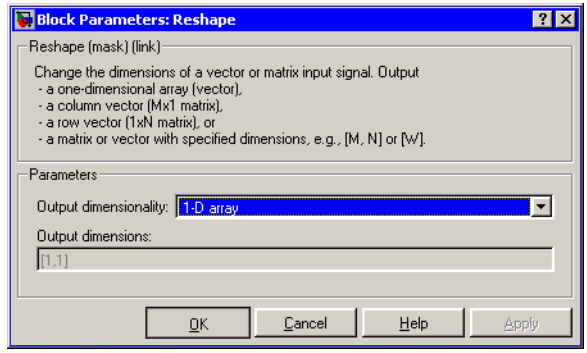
Output Dimensionality	Description
Row vector	Converts a vector or matrix input signal to a row matrix, i.e., a 1-by-N matrix where N is the number of elements in the input signal. For matrices, the conversion is done in column-major order.
Customize	Converts the input signal to an output signal whose dimensions you specify, using the **Output dimensions** parameter. The value of the **Output dimensions** parameter can be a one- or two-element vector. A value of `[N]` outputs a vector of size N. A value of `[M N]` outputs an M-by-N matrix. The number of elements of the input signal must match the number of elements specified by the **Output dimensions** parameter. For matrices, the conversion is done in column-major order.

Data Type Support

The Reshape block accepts and outputs signals of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box



Output dimensionality

The dimensionality of the output signal.

Output dimensions

Specifies a custom output dimensionality. This option is enabled only if you select Customize as the value of the **Output dimensionality** parameter.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	N/A
Dimensionalized	Yes
Zero Crossing	No

Purpose

Apply a rounding function to a signal

Library

Math Operations

Description

The Rounding Function block applies a rounding function to the input signal to produce the output signal.

You can select one of the following rounding functions from the **Function** list:

- `floor`

  Rounds each element of the input signal to the nearest integer value towards minus infinity.

- `ceil`

  Rounds each element of the input signal to the nearest integer towards positive infinity.

- `round`

  Rounds each element of the input signal to the nearest integer.

- `fix`

  Rounds each element of the input signal to the nearest integer towards zero.

The name of the selected function appears on the block.

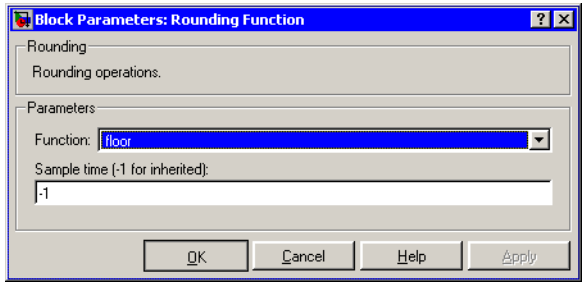
The input signal can be a scalar, vector, or matrix signal having real- or complex-valued elements of type `double`. The output signal has the same dimensions, data type, and numeric type as the input. Each element of the output signal is the result of applying the selected rounding function to the corresponding element of the input signal.

Use the Rounding Function block instead of the Fcn block when you want vector or matrix output, because the Fcn block can produce only scalar output.

Data Type Support

The Rounding Function block accepts and outputs real signals of type `double` or `single`.

Parameters and Dialog Box



Function

The rounding function.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See “Specifying Sample Time” in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	N/A
Dimensionalized	Yes
Zero Crossing	No

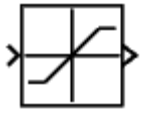
Purpose

Limit the range of a signal

Library

Discontinuities

Description



The Saturation block imposes upper and lower bounds on a signal. When the input signal is within the range specified by the **Lower limit** and **Upper limit** parameters, the input signal passes through unchanged. When the input signal is outside these bounds, the signal is clipped to the upper or lower bound.

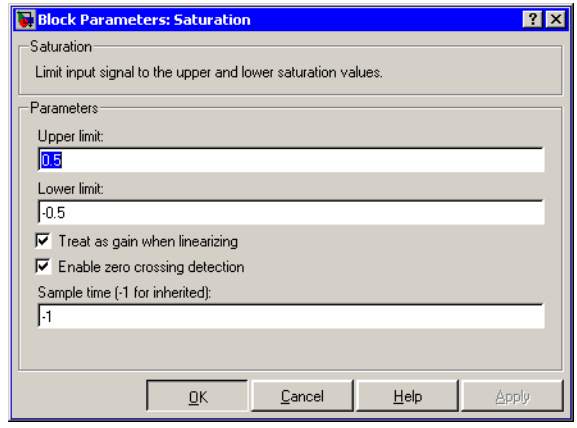
When the **Lower limit** and **Upper limit** parameters are set to the same value, the block outputs that value.

Data Type Support

The Saturation block accepts real signals of any data type supported by Simulink, except `Boolean`. The Saturation block supports fixed-point data types. The output data type is the same as the input data type.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box



Upper limit
Specify the upper bound on the input signal. When the input signal to the Saturation block is above this value, the output of the block is clipped to this value.

The **Upper limit** parameter is converted to the input data type offline using round-to-nearest and saturation.

Lower limit
Specify the lower bound on the input signal. When the input signal to the Saturation block is below this value, the output of the block is clipped to this value.

The **Lower limit** parameter is converted to the input data type offline using round-to-nearest and saturation.

Treat as gain when linearizing
Linearization commands in Simulink treat this block as a gain in state space. Select this parameter to cause the linearization commands to treat the gain as 1; otherwise, the commands treat the gain as 0.

Enable zero crossing detection
Select to enable zero crossing detection. For more information, see "Zero-Crossing Detection" in the Using Simulink documentation.

Sample time (-1 for inherited)
Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of parameters and input
Dimensionalized	Yes
Zero Crossing	Yes, if enabled.

See Also

Saturation Dynamic

Saturation Dynamic

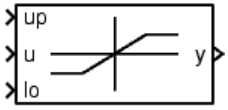
Purpose

Bound the range of the input

Library

Discontinuities

Description



The Saturation Dynamic block bounds the range of the input signal to upper and lower saturation values. The input signal outside of these limits saturates to one of the bounds where

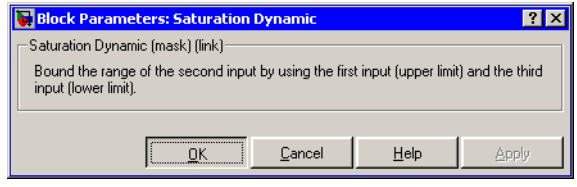
- The input below the lower limit is set to the lower limit.
- The input above the upper limit is set to the upper limit.

The input for the upper limit is the `up` port, and the input for the lower limit is the `lo` port.

Data Type Support

The Saturation Dynamic block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also

Saturation

Purpose

Display signals generated during a simulation

Library

Sinks

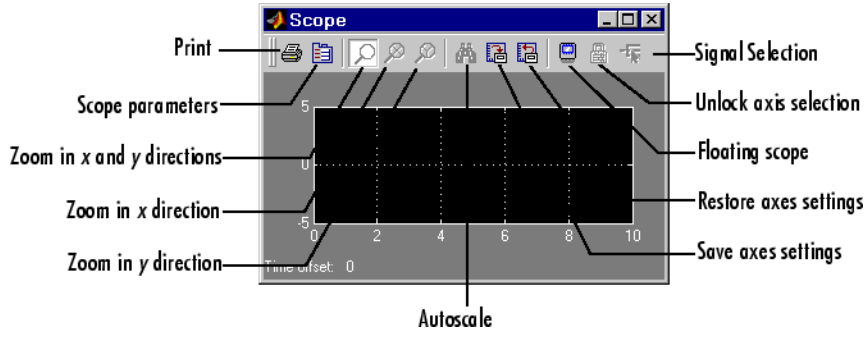
Description

The Scope block displays its input with respect to simulation time. The Scope block can have multiple axes (one per port); all axes have a common time range with independent *y*-axes. The Scope allows you to adjust the amount of time and the range of input values displayed. You can move and resize the Scope window and you can modify the Scope's parameter values during the simulation.

When you start a simulation, Simulink does not open Scope windows, although it does write data to connected Scopes. As a result, if you open a Scope after a simulation, the Scope's input signal or signals will be displayed.

If the signal is continuous, the Scope produces a point-to-point plot. If the signal is discrete, the Scope produces a stair-step plot.

The Scope provides toolbar buttons that enable you to zoom in on displayed data, display all the data input to the Scope, preserve axis settings from one simulation to the next, limit data displayed, and save data to the workspace. The toolbar buttons are labeled in this figure, which shows the Scope window as it appears when you open a Scope block.



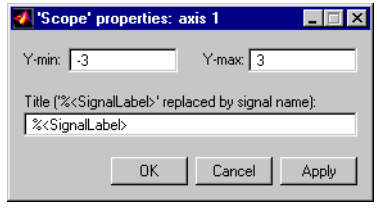
Note Do not use Scope blocks inside library blocks that you create. Instead, provide the library blocks with output ports to which scopes can be connected to display internal data.

Displaying Vector Signals

When displaying a vector or matrix signal, the Scope assigns colors to each signal element in this order: yellow, magenta, cyan, red, green, and dark blue. When more than six signals are displayed, the Scope cycles through the colors in the order listed.

Y-Axis Limits

You set *y*-limits by right-clicking an axis and choosing **Axes Properties**. The following dialog box appears.



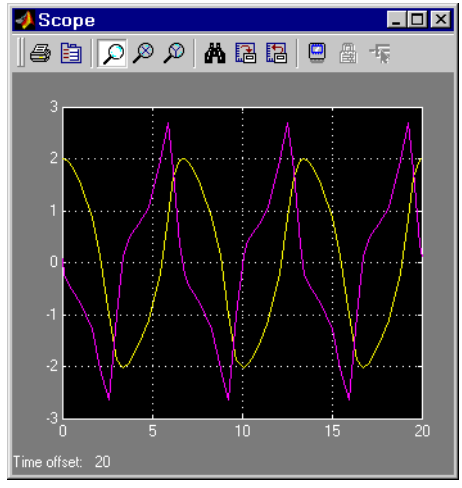
Y-min
: Enter the minimum value for the *y*-axis.

Y-max
: Enter the maximum value for the *y*-axis.

Title
: Enter the title of the plot. You can include a signal label in the title by typing `%<SignalLabel>` as part of the title string (`%<SignalLabel>` is replaced by the signal label).

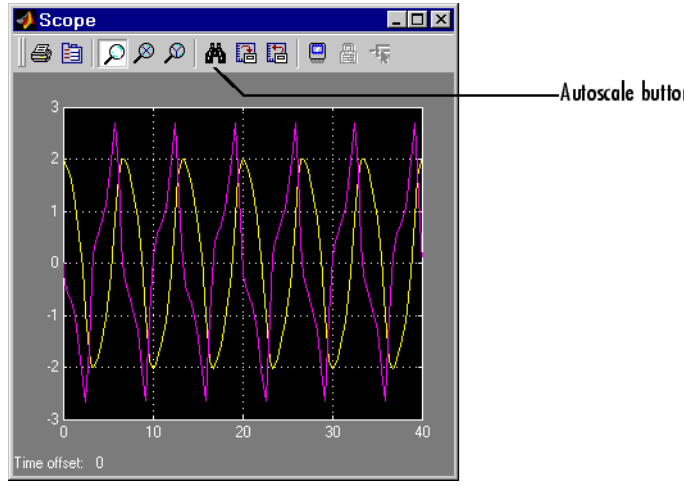
Time Offset

This figure shows the Scope block displaying the output of the `vdp` model. The simulation was run for `40` seconds. Note that this scope shows the final `20` seconds of the simulation. The **Time offset** field displays the time corresponding to `0` on the horizontal axis. Thus, you have to add the offset to the fixed time range values on the *x*-axis to get the actual time.



Autoscaling the Scope Axes

This figure shows the same output after you click the **Autoscale** toolbar button, which automatically scales both axes to display all stored simulation data. In this case, the *y*-axis was not scaled because it was already set to the appropriate limits.



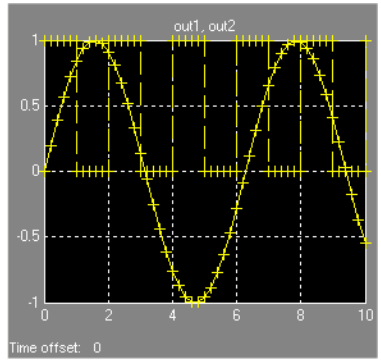
If you click the **Autoscale** button while the simulation is running, the axes are autoscaled based on the data displayed on the current screen, and the autoscale limits are saved as the defaults. This enables you to use the same limits for another simulation.

Note Simulink does not buffer the data that it displays on a floating Scope. It can therefore scale the contents of a floating Scope only when data is being displayed, i.e., when a simulation is running. When a simulation is not running, Simulink disables (grays) the **Zoom** button on the toolbar of a floating Scope to indicate that it cannot scale its contents.

Zooming

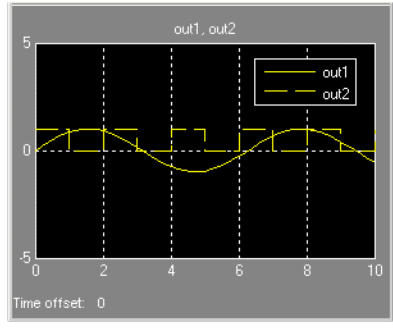
You can zoom in on data in both the *x* and *y* directions at the same time, or in either direction separately. The zoom feature is not active while the simulation is running.

To zoom in on data in both directions at the same time, make sure you select the leftmost **Zoom** toolbar button. Then, define the zoom region



Legends

Displays a legend on the scope that indicates the line style used to display each signal.



Floating scope

This option appears only on the **General** parameters pane for the Scope block.

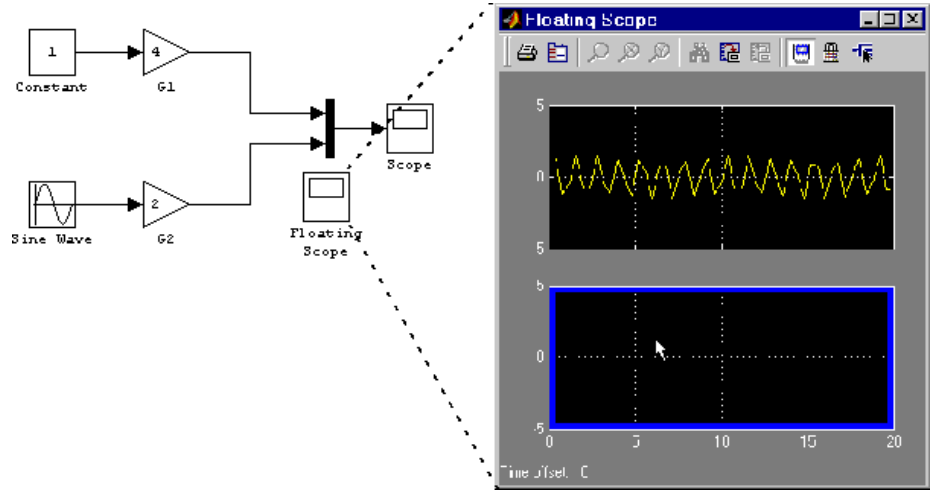
Selecting this option turns a Scope block into a floating scope. A floating scope is a Scope block that can display the signals carried on one or more lines. You can create a Floating Scope block in a model either by copying a Scope block from the Simulink Sinks library into a model and selecting this option or, more simply, by

copying the Floating Scope block from the Sinks library into the model window. The Floating Scope block has the **Floating scope** parameter selected by default.

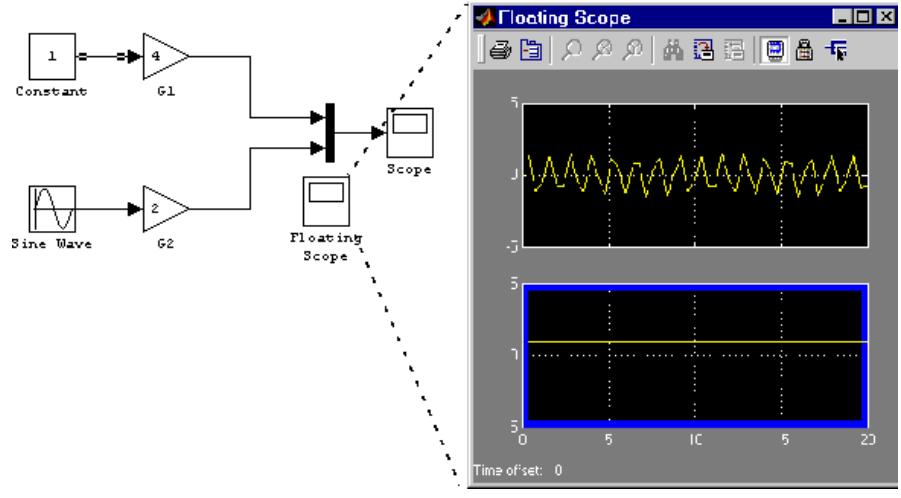
To use a floating scope during a simulation, first open the scope. To display the signals carried on a line, select the line. Hold down the **Shift** key while clicking another line to select multiple lines. It might be necessary to click the **Autoscale data** button on the floating scope's toolbar to find the signal and adjust the axes to the signal values. Or you can use the floating scope's Signal Selector (see "The Signal Selector" in the online Simulink documentation) to select signals for display. To display a floating scope's Signal Selector, first start simulation of your model with the floating scope open. Then right-click your mouse in the floating scope and select **Signal Selection** from the pop-up menu that appears.

You can have more than one floating scope in a model, but only one set of axes in one scope can be active at a given time. Active floating scopes show the active axes by making them blue. Selecting or deselecting lines affects the active floating scope only. Other floating scopes continue to display the signals that you selected when they were active. In other words, inactive floating scopes are locked, in that their signal displays cannot change.

To specify display of a signal on one of the axes of a multiaxis floating scope, click the axis. Simulink draws a blue border around the axis.



Then click the signal you want to display in the block diagram or the Signal Selector. When you run the model, the selected signal appears in the selected axis.



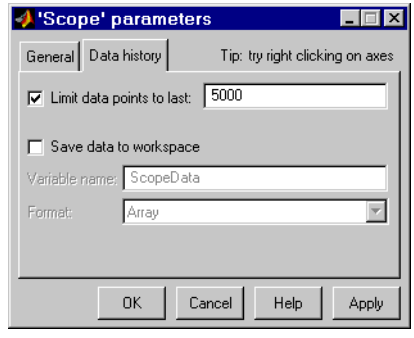
If you plan to use a floating scope during a simulation, you should disable signal storage reuse. See "Signal storage reuse" in "The Optimization Pane" for more information.

Sampling

To specify a decimation factor, enter a number in the data field to the right of the **Decimation** choice. To display data at a sampling interval, select the **Sample time** choice and enter a number in the data field.

Data History Parameters Pane

The **Data History** parameters pane appears only on the **Parameters** dialog box for the Scope block. The pane appears as follows.



This pane lets you control the amount of data that the Scope stores and displays. You can also choose to save data to the workspace in this pane. You apply the current parameters and options by clicking the **Apply** or **OK** button. The values that appear in these fields are the values that are used in the next simulation.

Limit data points to last

You can limit the number of data points saved to the workspace by selecting the **Limit data points to last** check box and entering a value in its data field. The Scope relies on its data history for zooming and autoscaling operations. If the number of data points

is limited to 1,000 and the simulation generates 2,000 data points, only the last 1,000 are available for regenerating the display.

Save data to workspace

You can automatically save the data collected by the Scope at the end of the simulation by selecting the **Save data to workspace** check box. If you select this option, the **Variable name** and **Format** fields become active.

Variable name

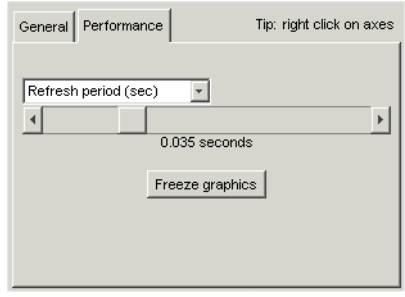
Enter a variable name in the **Variable name** field. The specified name must be unique among all data logging variables being used in the model. Other data logging variables are defined on other Scope blocks, To Workspace blocks, and simulation return variables such as time, states, and outputs. Being able to save Scope data to the workspace means that it is not necessary to send the same data stream to both a Scope block and a To Workspace block.

Format

Data can be saved in one of three formats: `Array`, `Structure`, or `Structure with time`. Use `Array` only for a Scope with one set of axes. For Scopes with more than one set of axes, use `Structure` if you do not want to store time data and use `Structure with time` if you want to store time data.

Performance Parameters Pane

The **Performance** parameters pane appears only on the **Parameters** dialog box for the Scope viewer. The pane appears as follows.



This pane lets you control how frequently Simulink refreshes the Scope viewer. Reducing the refresh rate can speed up the simulation in some cases. The pane contains the following controls.

Refresh Period

This list control lets you select the units in which the refresh period is expressed. Options are either seconds or frames where a frame is the width of the scope's screen in seconds, i.e., it equals the value of the scope's **Time range** parameter.

Refresh Slider

Drag the slider button to the right to increase the refresh period and hence decrease the refresh rate.

Freeze Button

Click the button to freeze (stop refreshing) or unfreeze the Scope viewer.

History Pane

The **History** parameters pane appears only on the **Parameters** dialog box for the Scope viewer.

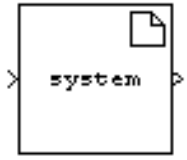
Purpose

Create an S-function from C code that you provide

Library

User-Defined Functions

Description



The S-Function Builder block creates a C MEX-file S-function from specifications and C source code that you provide. See “Building S-Functions Automatically” for detailed instructions on using the S-Function Builder block to generate an S-function.

Instances of the S-Function Builder block also serve as wrappers for generated S-functions in Simulink models. When simulating a model containing instances of an S-Function Builder block, Simulink invokes the generated S-function associated with each instance to compute the instance’s output at each time step.

Data Type Support

The S-Function Builder can accept and output complex, 1-D or 2-D signals of any data type supported by Simulink.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box

See “S-Function Builder Dialog Box” in the online documentation for information on using the S-Function Builder block’s parameter dialog box.

Shift Arithmetic

Purpose

Shift the bits and/or binary point of a signal

Library

Logic and Bit Operations

Description

The Shift Arithmetic block can be used to shift the bits or the binary point of a signal, or both.

For example, the effects of binary point shifts two places to the right and two places to the left on an input of data type `sfix(8)` are shown below.

Shift Operation	Binary Value	Decimal Value
No shift (original number)	11001.011	-6.625
Binary point shift right by two places	1100101.1	-26.5
Binary point shift left by two places	110.01011	-1.65625

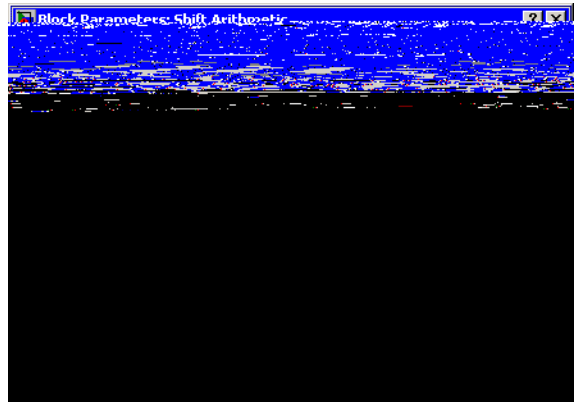
This block performs arithmetic bit shifts on signed numbers. Therefore, the most significant bit is recycled for each bit shift. The effects of bit shifts two places to the right and two places to the left on an input of data type `sfix(8)` follow.

Shift Operation	Binary Value	Decimal Value
No shift (original number)	11001.011	-6.625
Bit shift right by two places	11110.010	-1.75
Bit shift left by two places	00101.100	5.5

Data Type Support

The Shift Arithmetic block accepts signals of any data type supported by Simulink, including fixed-point data types, except `boolean` type.

Parameters and Dialog Box



Shift bits right how many places (negative is shift left)
The number of places the bits of the input signal is shifted. A positive value indicates a shift right, while a negative value indicates a shift left.

Shift binary point right how many places (negative is shift left)
The number of places the binary point of the input signal is shifted. A positive value indicates a shift right, while a negative value indicates a shift left.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited
Scalar Expansion	Yes

A negative **Amplitude** parameter value causes a 180-degree phase shift. You can generate a phase-shifted wave at other than 180 degrees in a variety of ways, including connecting a Clock block signal to a MATLAB Fcn block and writing the equation for the particular wave.

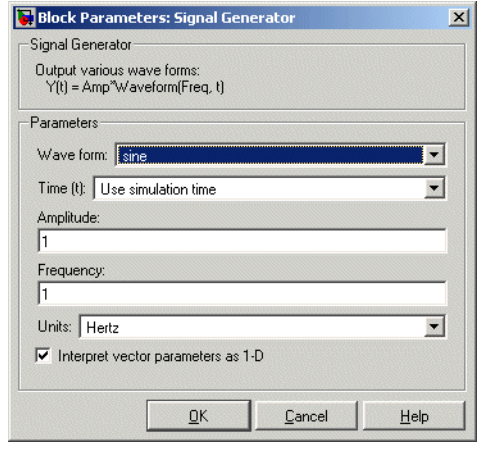
You can vary the output settings of the Signal Generator block while a simulation is in progress. This is useful to determine quickly the response of a system to different types of inputs.

The block's **Amplitude** and **Frequency** parameters determine the amplitude and frequency of the output signal. The parameters must be of the same dimensions after scalar expansion. If the **Interpret vector parameters as 1-D** option is off, the block outputs a signal of the same dimensions as the **Amplitude** and **Frequency** parameters (after scalar expansion). If the **Interpret vector parameters as 1-D** option is on, the block outputs a vector (1-D) signal if the **Amplitude** and **Frequency** parameters are row or column vectors, i.e. single row or column 2-D arrays. Otherwise, the block outputs a signal of the same dimensions as the parameters.

Data Type Support

The Signal Generator block outputs a scalar or array of real signals of type `double`.

Parameters and Dialog Box



Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in the online Simulink documentation for details.

Wave form
The wave form: a sine wave, square wave, or sawtooth wave. The default is a sine wave. This parameter cannot be changed while a simulation is running.

Time
Specifies whether to use simulation time as the source of values for the waveform's time variable or an external signal. If you specify an external time source, the block displays an input port for the time source.

Amplitude
The signal amplitude. The default is `1`.

Frequency
The signal frequency. The default is `1`.

Units
The signal units: Hertz or radians/sec. The default is Hertz.

Interpret vector parameters as 1-D

If selected, column or row matrix values for the **Amplitude** and **Frequency** parameters result in a vector output signal.

Characteristics

Sample Time	Continuous
Scalar Expansion	Yes, of parameters
Dimensionalized	Yes
Zero Crossing	No

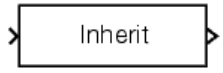
Purpose

Specify the desired dimensions, sample time, data type, numeric type, and other attributes of a signal.

Library

Signal Attributes

Description



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 displays an error when it compiles the model, for example, at the beginning of a simulation. If no conflict exists, Simulink eliminates the Signal Specification block from the compiled model. In other words, the Signal Specification block is a virtual block. It exists only to specify the attributes of a signal and plays no role in the simulation of the model.

You can use the Signal Specification block to ensure that the actual attributes of a signal meet desired attributes. For example, suppose that you and a colleague are working on different parts of the same model and you use Signal Specification blocks to connect your part of the model with your colleague's. Now, if your colleague changes the attributes of a signal without informing you, the attributes entering the corresponding Signal Specification block do not match and Simulink reports an appropriate error.

The Signal Specification block can also be used to ensure correct propagation of signal attributes throughout a model. The capability of allowing many attributes to be propagated from block to block in Simulink is very powerful. However, because blocks may not specify some or all of the attributes of the signals they accept or output, it is possible to create models that don't have enough information to correctly propagate attributes around the model. For these cases, the Signal Specification block is a good way of providing the information Simulink needs. Using the Signal Specification block also helps speed up model compilation when blocks are missing signal attributes.

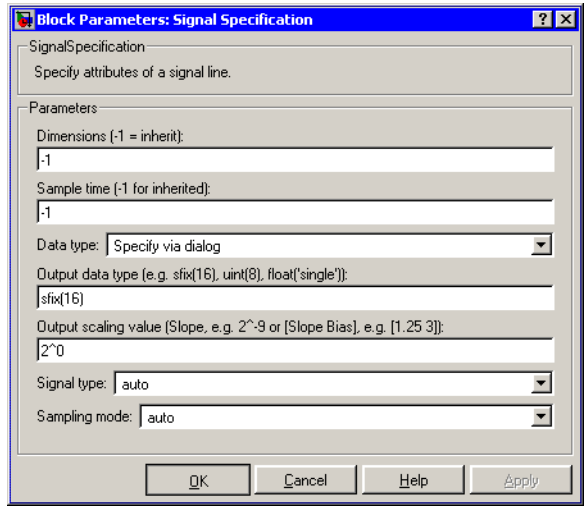
Data Type Support

The Signal Specification block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types. The

input data type must match the data type specified by the **Data type** parameter.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box



Dimensions

Specify the dimension’s of the block’s input and output signals. Valid values are

- `-1`–Inherited from the block to which it is connected
- `n`–Vector signal of width `n`
- `[m n]`–Matrix signal having `m` rows and `n` columns

Sample Time

Specify the sample time at which the block is updated. Valid values are

- `-1`–inherited from the block to which it is connected

- `period >= 0`
- `[period, offset]`
- `[0, -1]`
- `[-1, -1]`

where `period` is the sample rate and `offset` is the offset of the sample period from time zero. See "Specifying Sample Time" in the online documentation for more information.

Data type
Specify the data type of the input and output signals. To let Simulink determine the data type, set this parameter to `auto`.

Output data type
Specify any data type, including fixed-point data types. This parameter is only visible if you select `Specify via dialog` for the **Data type** parameter.

Output scaling value
Set the output scaling using binary point-only or [Slope Bias] scaling. This parameter is only visible if you select `Specify via dialog` for the **Data type** parameter.

Signal type
Specify the numeric type (`real` or `complex`) of the input and output signal. To let Simulink determine the numeric type, set this parameter to `auto`.

Sampling mode
Specify the sampling mode (`sample-based` or `frame-based`) of this block. To let Simulink determine the sampling mode, set this parameter to `auto`.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified by the block's Sample Time parameter.
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

Purpose

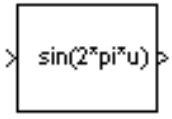
Implement a sine wave in fixed-point using a lookup table approach that exploits quarter wave symmetry

Library

Lookup Tables

Description

The Sine block is an implementation of the Sine and Cosine block. See `Sine and Cosine` for more information.



Sine and Cosine

Purpose

Implement a sine and/or cosine wave in fixed point using a lookup table approach that exploits quarter wave symmetry

Library

Lookup Tables (Sine block or Cosine block)

Description

The Sine and Cosine block implements a sine and/or cosine wave in fixed point using a lookup table method that exploits quarter wave symmetry.

The Sine and Cosine block can output the following functions of the input signal, depending upon what you select for the **Output formula** parameter:

- $\sin(2\pi u)$
- $\cos(2\pi u)$
- $\exp(i2\pi u)$
- $\sin(2\pi u)$ and $\cos(2\pi u)$

You define the number of lookup table points in the **Number of data points for lookup table** parameter. The block implementation is most efficient when you specify the lookup table data points to be `(2^n)+1`, where `n` is an integer.

Use the **Output word length** parameter to specify the word length of the fixed-point output data type. The fraction length of the output is the output word length minus 2.

Data Type Support

The Sine and Cosine block accepts signals of any data type supported by Simulink, including fixed-point data types. The output of the block is a fixed-point data type.

Parameters and Dialog Box

Output formula
Select the signal(s) to output.

Number of data points for lookup table
Specify the number of data points to retrieve from the lookup table. The implementation is most efficient when you specify the lookup table data points to be `(2^n)+1`, where `n` is an integer.

Output word length
Specify the word length for the fixed-point data type of the output signal. The fraction length of the output is the output word length minus 2.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	N/A

specify an external time source, the block displays an input port for the time source.

Amplitude
The amplitude of the signal. The default is `1`.

Bias
Constant value added to the sine to produce the output of this block.

Frequency
The frequency, in radians/second. The default is `1` rad/s. This parameter appears only if you choose time-based as the **Sine type** of the block.

Samples per period
Number of samples per period. This parameter appears only if you choose sample-based as the **Sine type** of the block.

Phase
The phase shift, in radians. The default is `0` radians. This parameter appears only if you choose time-based as the **Sine type** of the block.

Number of offset samples
The offset (discrete phase shift) in number of sample times. This parameter appears only if you choose sample-based as the **Sine type** of the block.

Sample time
The sample period. The default is `0`. If the sine type is sample-based, the sample time must be greater than 0. See "Specifying Sample Time" in the online documentation for more information.

Interpret vector parameters as 1-D
If selected, column or row matrix values for the Sine Wave block's numeric parameters result in a vector output signal; otherwise, the block outputs a signal of the same dimensionality as the parameters. If this option is not selected, the block always

outputs a signal of the same dimensionality as the block's numeric parameters.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of parameters
Dimensionalized	Yes
Zero Crossing	No

Sine Wave Function

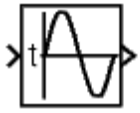
Purpose

Generate a sine wave, using an external signal as a time source.

Library

Math Operations

Description



This block is the same as the Sine Wave block with its **Time** parameter set to `Use external source`. See the documentation for the Sine Wave block for more information.

Purpose

Vary a scalar gain using a slider

Library

Math Operations

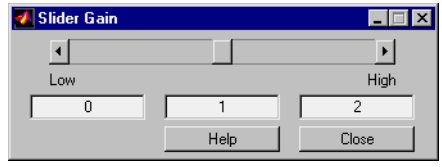
Description

The Slider Gain block allows you to vary a scalar gain during a simulation using a slider. The block accepts one input and generates one output.

Data Type Support

Data type support for the Slider Gain block is the same as that for the Gain block (see `Gain`).

Parameters and Dialog Box



Low

The lower limit of the slider range. The default is `0`.

High

The upper limit of the slider range. The default is `2`.

The edit fields indicate (from left to right) the lower limit, the current value, and the upper limit. You can change the gain in two ways: by manipulating the slider, or by entering a new value in the current value field. You can change the range of gain values by changing the lower and upper limits. Close the dialog box by clicking the **Close** button.

If you click the slider's left or right arrow, the current value changes by about 1% of the slider's range. If you click the rectangular area to either side of the slider's indicator, the current value changes by about 10% of the slider's range.

To apply a vector or matrix gain to the block input, consider using the Gain block.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes, of the gain
States	0
Dimensionalized	Yes
Zero Crossing	No

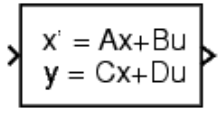
Purpose

Implement a linear state-space system

Library

Continuous

Description

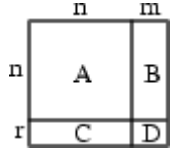


The State-Space block implements a system whose behavior is defined by

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

where x is the state vector, u is the input vector, and y is the output vector. The matrix coefficients must have these characteristics, as illustrated in the following diagram:

- **A** must be an n-by-n matrix, where n is the number of states.
- **B** must be an n-by-m matrix, where m is the number of inputs.
- **C** must be an r-by-n matrix, where r is the number of outputs.
- **D** must be an r-by-m matrix.



The block accepts one input and generates one output. The input vector width is determined by the number of columns in the B and D matrices. The output vector width is determined by the number of rows in the C and D matrices.

Simulink converts a matrix containing zeros to a sparse matrix for efficient multiplication.

Specifying the Absolute Tolerance for the Block's States

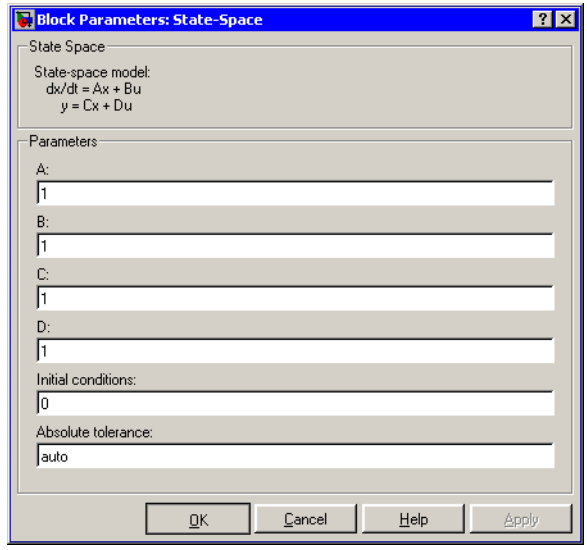
By default Simulink uses the absolute tolerance value specified in the **Configuration Parameters** dialog box (see "Solver Pane") to solve the

states of the State-Space block. If this value does not provide sufficient error control, specify a more appropriate value in the **Absolute tolerance** field of the State-Space block's dialog box. The value that you specify is used to solve all the block's states.

Data Type Support

A State-Space block accepts and outputs real signals of type `double`.

Parameters and Dialog Box



A, B, C, D
The matrix coefficients.

Initial conditions
The initial state vector. Simulink does not allow the initial conditions of this block to be `inf` or `NaN`.

Absolute tolerance
Absolute tolerance used to solve the block's states. You can enter `auto` or a numeric value. If you enter `auto`, Simulink determines the absolute tolerance (see "Solver Pane"). If you enter a numeric

value, Simulink uses the specified value to solve the block's states. Note that a numeric value overrides the setting for the absolute tolerance in the **Configuration Parameters** dialog box.

Characteristics

Direct Feedthrough	Only if $D \neq 0$
Sample Time	Continuous
Scalar Expansion	Yes, of the initial conditions
States	Depends on the size of A
Dimensionalized	Yes
Zero Crossing	No

Step

Purpose

Generate a step function

Library

Sources

Description

The Step block provides a step between two definable levels at a specified time. If the simulation time is less than the **Step time** parameter value, the block's output is the **Initial value** parameter value. For simulation time greater than or equal to the **Step time**, the output is the **Final value** parameter value.

The block's numeric parameters must be of the same dimensions after scalar expansion. If the **Interpret vector parameters as 1-D** option is off, the block outputs a signal of the same dimensions and dimensionality as the parameters. If the **Interpret vector parameters as 1-D** option is on and the numeric parameters are row or column vectors (i.e., single row or column 2-D arrays), the block outputs a vector (1-D array) signal; otherwise, the block outputs a signal of the same dimensionality and dimensions as the parameters.

Data Type Support

The Step block outputs real signals of type `double`.

Parameters and Dialog Box

this function-call subsystem's inputs directly or indirectly during execution of a function-call (see the "Function-call systems" examples in the Simulink "Subsystem Semantics" library for examples of such function-call subsystems.

Sample time
Specifies the sample time of this subsystem.

Note Parameters whose names begin with RTW are used by Real-Time Workshop for code generation. See "Building Subsystems and Working with Referenced Models" in the Real-Time Workshop documentation for more information.

Characteristics

Sample Time	Depends on the blocks in the subsystem
Dimensionalized	Depends on the blocks in the subsystem
Zero Crossing	Yes, for enable and trigger ports if present

Purpose

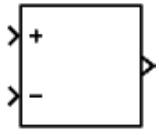
Add or subtract inputs

Library

Math Operations

Description

The Subtract block is an implementation of the Sum block. See `Sum` for more information.



Sum

Purpose

Add or subtract inputs

Library

Math Operations

Description

The Sum block performs addition or subtraction on its inputs. This block can add or subtract scalar, vector, or matrix inputs. It can also collapse the elements of a single input vector.

You specify the operations of the block with the **List of Signs** parameter. Plus (+), minus (-), and spacer (|) characters indicate the operations to be performed on the inputs:

- If there are two or more inputs, then the number of characters must equal the number of inputs. For example, "+-+" requires three inputs and configures the block to subtract the second (middle) input from the first (top) input, and then add the third (bottom) input.

  All nonscalar inputs must have the same dimensions. Scalar inputs will be expanded to have the same dimensions as the other inputs.
- A spacer character creates extra space between ports on the block's icon.
- If only addition of all inputs is required, then a numeric parameter value equal to the number of inputs can be supplied instead of "+" characters.
- If only one vector is input, then a single "+" or "-" will collapse the vector using the specified operation.

The Sum block first converts the input data type(s) to the output data type using the specified rounding and overflow modes, and then performs the specified operations.

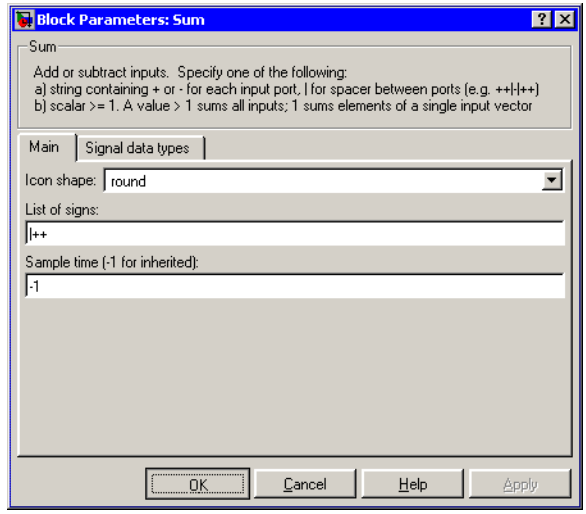
Data Type Support

The Sum block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types. The inputs may be of different data types unless you select the **Require all inputs to have same data type** parameter.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box

The **Main** pane of the Sum block dialog appears as follows:



Icon shape

Designate the icon shape of the block.

List of signs

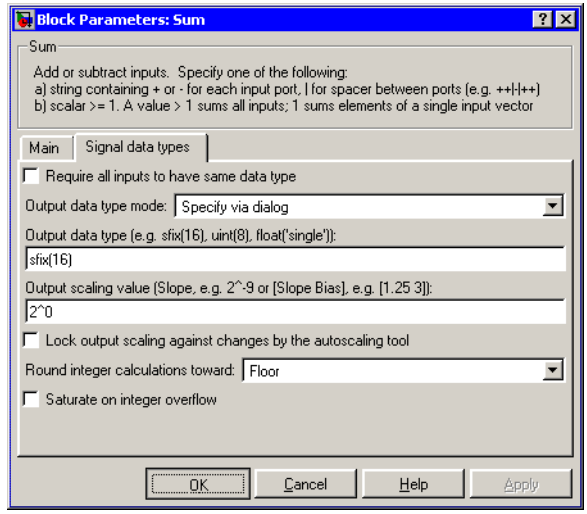
Enter as many plus (+) and minus (-) characters as there are inputs. Addition is the default operation, so if you only want to add the inputs, enter the number of input ports. For a single vector input, "+" or "-" will collapse the vector using the specified operation.

You can manipulate the positions of the input ports on the block by inserting spacers (|) between the signs in the **List of signs** parameter. For example, "++|--" creates an extra space between the second and third input ports.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

The **Signal data types** pane of the Sum block dialog appears as follows:



Require all inputs to have same data type

Select this parameter to require that all inputs must have the same data type.

Output data type mode

Specify the output data type and scaling to be the same as the first input, or inherit the data type and scaling from an internal rule or by backpropagation. You can also choose a built-in data type from the drop-down list. Lastly, if you choose `Specify via dialog`, the **Output data type**, **Output scaling value**, and **Lock output scaling against changes by the autoscaling tool** parameters become visible.

Output data type
Specify any data type, including fixed-point data types. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Output scaling value
Set the output scaling using binary point-only or [Slope Bias] scaling. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Lock output scaling against changes by the autoscaling tool
Select to lock scaling of outputs. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

Round integer calculations toward
Select the rounding mode for fixed-point operations.

Saturate on integer overflow
Select to have overflows saturate.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes
States	0
Dimensionalized	Yes
Zero Crossing	No

Sum of Elements

Purpose

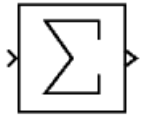
Add or subtract inputs

Library

Math Operations

Description

The Sum of Elements block is an implementation of the Sum block. See `Sum` for more information.



Purpose

Switch output between the first input and the third input based on the value of the second input

Library

Signal Routing

Description

The Switch block passes through the first (top) input or the third (bottom) input based on the value of the second (middle) input. The first and third inputs are called *data inputs*. The second input is called the *control input*.

You select the conditions under which the first input is passed with the **Criteria for passing first input** parameter. You can make the block check whether the control input is greater than or equal to the threshold value, purely greater than the threshold value, or nonzero. If the control input meets the condition set in the **Criteria for passing first input parameter**, then the first input is passed. Otherwise, the third input is passed.

Note If the data inputs to the switch are buses, the element names of both buses must be the same to ensure that the output bus has the same element names no matter which input bus is selected. You can ensure that your model meets this requirement by using a bus object to define the buses with the model's **Element name mismatch** diagnostic set to error. See "Connectivity Diagnostics" for more information.

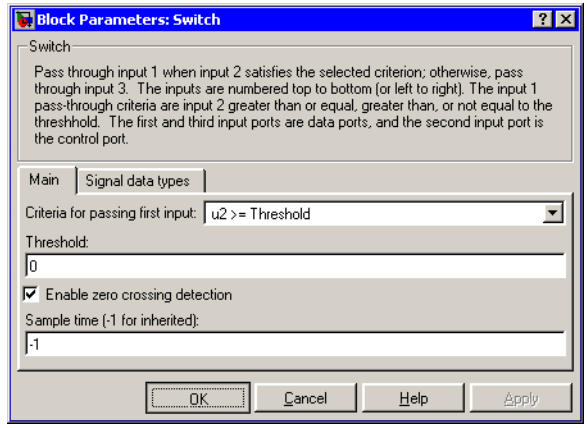
Data Type Support

The data and control inputs of a Switch block accept real or complex signals of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see"Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box

The **Main** pane of the Switch block dialog appears as follows:



Criteria for passing first input

Select the conditions under which the first input is passed. You can make the block check whether the control input is greater than or equal to the threshold value, purely greater than the threshold value, or nonzero. If the control input meets the condition set in this parameter, then the first input is passed. Otherwise, the third input is passed.

Threshold

Assign the switch threshold that determines which input is passed to the output.

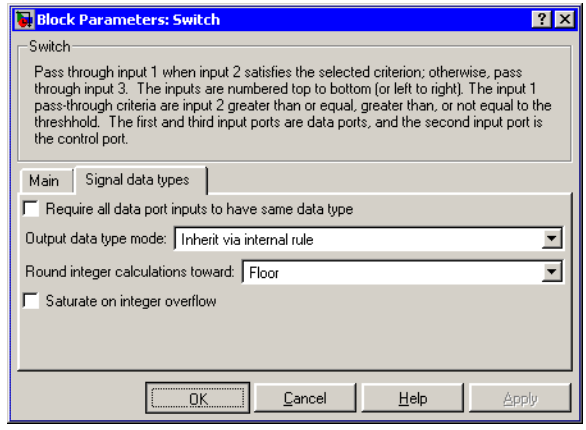
Enable zero crossing detection

Select to enable zero crossing detection. For more information, see "Zero Crossing Detection" in the Using Simulink documentation.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

The **Signal data types** pane of the Switch block dialog appears as follows:



Require all data port inputs to have same data type
Select to require all data inputs to have the same data type.

Output data type mode
Choose to inherit the output data type and scaling by backpropagation or by an internal rule. The internal rule causes the output of the block to have the same data type and scaling as the input with the larger positive range.

Round integer calculations toward
Select the rounding mode for fixed-point operations.

Saturate on integer overflow
Select to have overflows saturate.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

Dimensionalized	Yes
Zero Crossing	Yes, if enabled

See Also

Multiport Switch

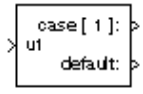
Purpose

Implement a C-like `switch` control flow statement

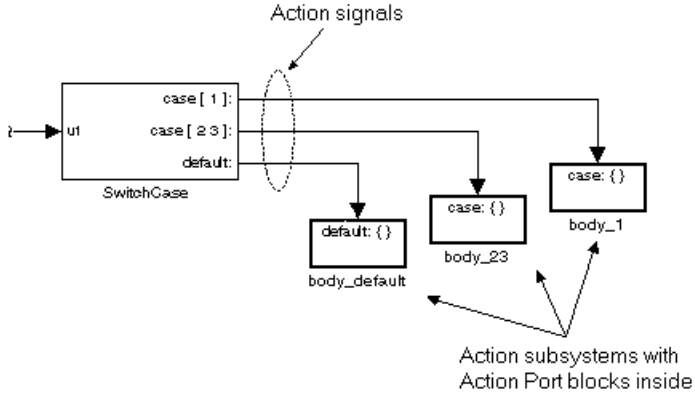
Library

Ports & Subsystems

Description



The following shows a completed Simulink C-like `switch` control flow statement in the subsystem of the Switch Case block.



A Switch Case block receives a single input, which it uses to form case conditions that determine which the subsystem to execute. Each output port case condition is attached to a Switch Case Action subsystem. The cases are evaluated top down starting with the top case. If a case value (in brackets) corresponds to the actual value of the input, its Switch Case Action subsystem is executed.

The preceding `switch` control flow statement can be represented by the following pseudocode:

```
switch (u1) {
 case [u1=1]:
  body_1;
  break;
 case [u1=2 or u1=3]:
  body_23;
  break;
```

```
 default:
  bodydefault;
}
```

You construct a Simulink `switch` control flow statement like the example shown as follows:

1. Place a Switch Case block in the current system and attach the input port labeled `u1` to the source of the data you are evaluating.
2. Open the **Block Parameters** dialog of the Switch Case block and enter as follows:
   a. Enter the **Case conditions** field with the individual cases.

      Each case can be an integer or set of integers specified with MATLAB cell notation. See the **Case conditions** field in the "Parameters and Dialog Box" section of this reference.

   b. Select the **Show default case** check box to show a default case output port on the Switch Case block.

      If all other cases are false, the default case is taken.
3. Create a Switch Case Action subsystem for each case port you added to the Switch Case block.

   These consist of subsystems with Action Port blocks inside them. When you place the Action Port block inside a subsystem, the subsystem becomes an atomic subsystem with an input port labeled `Action`.
4. Connect each case output port and the default output port of the Switch Case block to the Action port of an Action subsystem.

   Each connected subsystem becomes a case body. This is indicated by the change in label for the Switch Case Action subsystem block and the Action Port block inside of it to the name `case{}`.

During simulation of a `switch` control flow statement, the Action signals from the Switch Case block to each Switch Case Action subsystem turn from solid to dashed.

5 In each Switch Case Action subsystem, enter the Simulink logic appropriate to the case it handles.

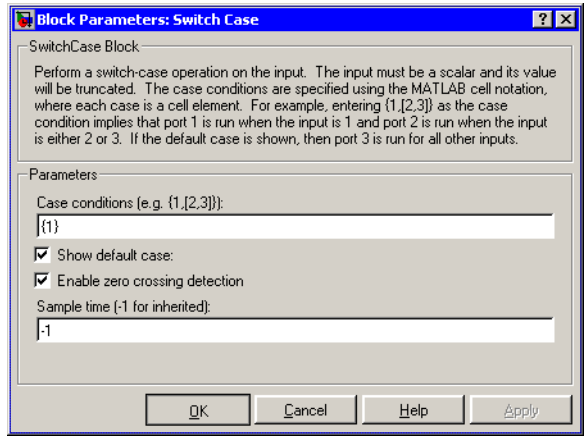
Note As demonstrated in the preceding pseudocode example, cases for the Switch Case block contain an implied break after their Switch Case Action subsystems are executed. There is no fall-through behavior for the Simulink `switch` control flow statement as found in standard C `switch` statements.

Data Type Support

Input to the port labeled `u1` of a Switch Case block can be a scalar value of any data type supported by Simulink except `Boolean`. The input to `u1` can also be a fixed-point data type. Noninteger inputs are truncated. For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Data outputs are action signals to Switch Case Action subsystems that are created with Action Port blocks and subsystems.

Parameters and Dialog Box



Case conditions

Case conditions are specified using MATLAB cell notation where each cell is a case condition consisting of integers or arrays of integers. In the preceding dialog example, entering `{1,[7,9,4]}` specifies that output port case[1] is run when the input value is `1`, and output port case[7 9 4] is run when the input value is `7`, `9`, or `4`.

You can use colon notation to specify a range of case conditions. For example, entering `{[1:5]}` specifies that output port case[1 2 3 4 5] is run when the input value is `1`, `2`, `3`, `4`, or `5`.

Depending on block size, cases with long lists of conditions are displayed in shortened form in the Switch Case block, using a terminating ellipsis (...).

Show default case

If you select this check box, the default output port appears as the last case on the Switch Case block. This case is run when the input value does not match any of the case values specified in the **Case conditions** field.

Enable zero crossing detection

Select to enable use of zero crossing detection. For more information, see "Zero Crossing Detection" in the Using Simulink documentation.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	No
Zero Crossing	Yes, if enabled

Switch Case Action Subsystem

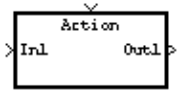
Purpose

Represent a subsystem whose execution is triggered by a Switch Case block

Library

Ports & Subsystems

Description



This block is a Subsystem block that is preconfigured to serve as a starting point for creating a subsystem whose execution is triggered by a Switch Case block. See the `Switch Case` block and "Control Flow Blocks" for more information.

Purpose

Delay a scalar signal multiple sample periods and output all the delayed versions

Library

Discrete

Description

The Tapped Delay block delays its input by the specified number of sample periods, and outputs all the delayed versions.

This block provides a mechanism for discretizing a signal in time, or resampling the signal at a different rate. You specify the time between samples with the **Sample time** parameter. You specify the number of delays with the **Number of delays** parameter. A value of `-1` instructs the block to inherit the number of delays by backpropagation. Each delay is equivalent to the z^{-1} discrete-time operator, which is represented by the Unit Delay block.

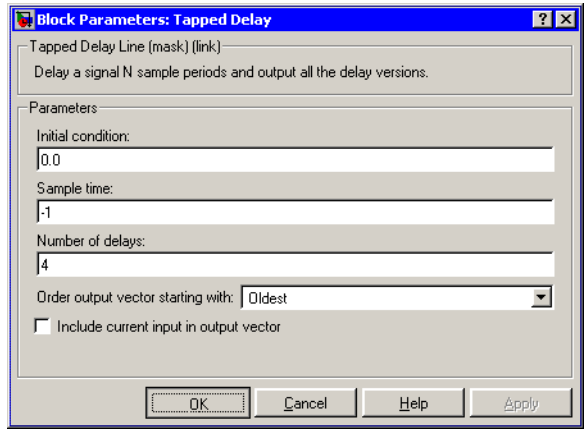
The block accepts one scalar input and generates an output for each delay. The input must be a scalar. You specify the order of the output vector with the **Order output vector starting with** parameter list. **Oldest** orders the output vector starting with the oldest delay version and ending with the newest delay version. **Newest** orders the output vector starting with the newest delay version and ending with the oldest delay version.

The block output for the first sampling period is specified by the **Initial condition** parameter. Careful selection of this parameter can minimize unwanted output behavior.

Data Type Support

The Tapped Delay block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Initial condition

Specify the initial output of the simulation. The **Initial condition** parameter is converted from a double to the input data type offline using round-to-nearest and saturation. Simulink does not allow you to set the initial condition of this block to `inf` or `NaN`.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Number of delays

Specify the number of discrete-time operators.

Order output vector starting with

Specify whether the oldest delay version is output first, or the newest delay version is output first.

Include current input in output vector

Select to include the current input in the output vector.

Characteristics

Direct Feedthrough	Yes, when **Include current input in output vector** parameter is checked. No otherwise.
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of initial conditions

Terminator

Purpose

Terminate an unconnected output port

Library

Sinks

Description

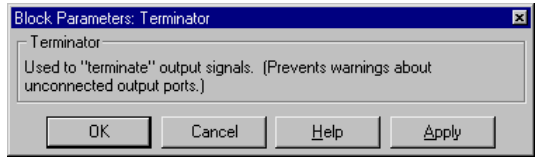
The Terminator block can be used to cap blocks whose output ports are not connected to other blocks. If you run a simulation with blocks having unconnected output ports, Simulink issues warning messages. Using Terminator blocks to cap those blocks avoids warning messages.

Data Type Support

The Terminator block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box



Characteristics

Sample Time	Inherited from driving block
Dimensionalized	Yes

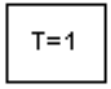
Purpose

Generate linear models in the base workspace at specific times

Library

Model-Wide Utilities

Description



This block calls `linmod` or `dlinmod` to create a linear model for the system when the simulation clock reaches the time specified by the **Linearization time** parameter. No trimming is performed. The linear model is stored in the base workspace as a structure, along with information about the operating point at which the snapshot was taken. Multiple snapshots are appended to form an array of structures.

The name of the structure used to save the snapshots is the name of the model appended by `_Timed_Based_Linearization`, for example, `vdp_Timed_Based_Linearization`. The structure has the follow fields:

Field	Description
`a`	The A matrix of the linearization
`b`	The B matrix of the linearization
`c`	The C matrix of the linearization
`d`	The D matrix of the linearization
`StateName`	Names of the model's states
`OutputName`	Names of the model's output ports
`InputName`	Names of the model's input ports
`OperPoint`	A structure that specifies the operating point of the linearization. The structure specifies the operating point time (`OperPoint.t`). The states (`OperPoint.x`) and inputs (`OperPoint.u`) fields are not used.
`Ts`	The sample time of the linearization for a discrete linearization

Use the Trigger-Based Linearization block if you need to generate linear models conditionally.

You can use state and simulation time logging to extract the model states and inputs at operating points. For example, suppose that you want to get the states of the f14 demo model at linearization times of 2 seconds and 5 seconds.

1. Open the model and drag an instance of this block from the Model-Wide Utilities library and drop the instance into the model.
2. Open the block's parameter dialog box and set the **Linearization time** to 2 and 5.
3. Open the model's **Configuration Parameters** dialog box.
4. Select the **Data Import/Export** pane.
5. Check **States** and **Time** on the **Save to Workspace** control panel
6. Select OK to confirm the selections and close the dialog box.
7. Simulate the model.

   At the end of the simulation, the following variables appear in the MATLAB workspace: `f14_Timed_Based_Linearization`, `tout`, and `xout`.
8. Get the indices to the operating point times by entering the following at the MATLAB command line:

```
ind1 = find(f14_Timed_Based_Linearization(1).OperPoint.t==tout);
ind2 = find(f14_Timed_Based_Linearization(1).OperPoint.t==tout);
```

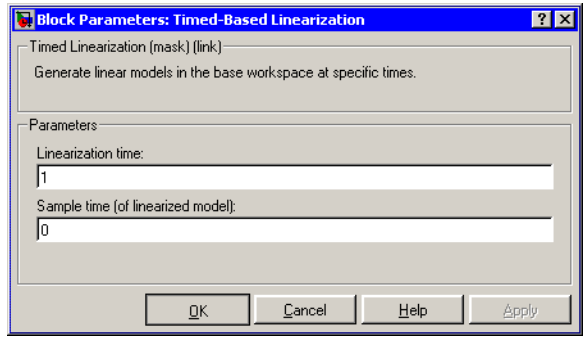
9. Get the state vectors at the operating points.

```
x1 = xout(ind1,:);
x2 = xout(ind2,:);
```

Data Type Support

Not applicable.

Parameters and Dialog Box



Linearization time

Time at which you want the block to generate a linear model. Enter a vector of times if you want the block to generate linear models at more than one time step.

Sample time (of linearized model)

Specify a sample time to create discrete-time linearizations of the model (see “Discrete-Time System Linearization” on page 3-6).

Characteristics

Sample Time	Specified in the **Sample time** parameter
Dimensionalized	No

To File

Purpose

Write data to a file

Library

Sinks

Description

The To File block writes its input to a matrix in a MAT-file. The block writes one column for each time step: the first row is the simulation time; the remainder of the column is the input data, one data point for each element in the input vector. The matrix has this form.

$$\begin{bmatrix} t_1 & t_2 & \dots t_{final} \\ u1_1 & u1_2 & \dots u1_{final} \\ \dots & & \\ un_1 & un_2 & \dots un_{final} \end{bmatrix}$$

The From File block can use data written by a To File block without any modifications. However, the form of the matrix expected by the From Workspace block is the transposition of the data written by the To File block.

The block writes the data as well as the simulation time after the simulation is completed. Its icon shows the name of the specified output file.

The amount of data written and the time steps at which the data is written are determined by block parameters:

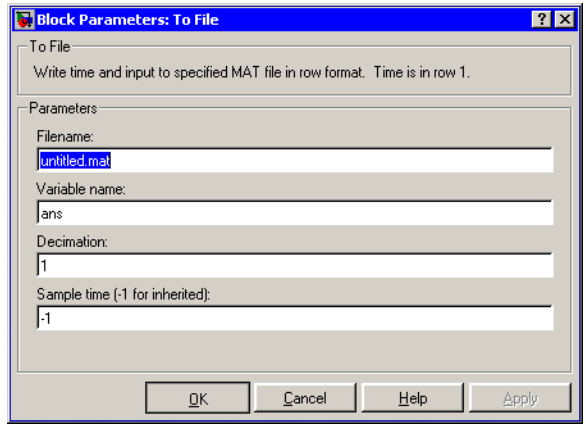
- The **Decimation** parameter allows you to write data at every nth sample, where n is the decimation factor. The default decimation, 1, writes data at every time step.
- The **Sample time** parameter allows you to specify a sampling interval at which to collect points. This parameter is useful when you are using a variable-step solver where the interval between time steps might not be the same. The default value of -1 causes the block to inherit the sample time from the driving block when determining the points to write. See "Specifying Sample Time" in the online documentation for more information.

If the file exists at the time the simulation starts, the block overwrites its contents.

Data Type Support

The To File block accepts real signals of type `double`.

Parameters and Dialog Box



Filename

The fully qualified pathname or filename of the MAT-file in which to store the output. On UNIX, the pathname may start with a tilde (~) character signifying your home directory. The default filename is `untitled.mat`. If you specify an unqualified filename, Simulink stores the file in the MATLAB working directory. (To determine the working directory, type `pwd` at the MATLAB command line.)

Variable name

The name of the matrix contained in the named file.

Decimation

A decimation factor. The default value is `1`.

Sample time

The sample period and offset at which to collect points. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Dimensionalized	Yes

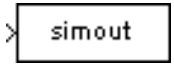
Purpose

Write data to the workspace

Library

Sinks

Description



The To Workspace block writes its input to the workspace. The block writes its output to an array or structure that has the name specified by the block's **Variable name** parameter. The **Save format** parameter determines the output format.

Array

Selecting this option causes the To Workspace block to save the input as an N-dimensional array where N is one more than the number of dimensions of the input signal. For example, if the input signal is a 1-D array (i.e., a vector), the resulting workspace array is two-dimensional. If the input signal is a 2-D array (i.e., a matrix), the array is three-dimensional.

The way samples are stored in the array depends on whether the input signal is a scalar or vector or a matrix. If the input is a scalar or a vector, each input sample is output as a row of the array. For example, suppose that the name of the output array is `simout`. Then, `simout(1,:)` corresponds to the first sample, `simout(2,:)` corresponds to the second sample, etc. If the input signal is a matrix, the third dimension of the workspace array corresponds to the values of the input signal at specified sampling point. For example, suppose again that `simout` is the name of the resulting workspace array. Then, `simout(:,:,1)` is the value of the input signal at the first sample point; `simout(:,:,2)` is the value of the input signal at the second sample point; etc.

The amount of data written and the time steps at which the data is written are determined by block parameters:

- The **Limit data points to last** parameter indicates how many sample points to save. If the simulation generates more data points than the specified maximum, the simulation saves only the most recently generated samples. To capture all the data, set this value to `inf`.

- The **Decimation** parameter allows you to write data at every `n`th sample, where `n` is the decimation factor. The default decimation, 1, writes data at every time step.
- The **Sample time** parameter allows you to specify a sampling interval at which to collect points. This parameter is useful when you are using a variable-step solver where the interval between time steps might not be the same. The default value of -1 causes the block to inherit the sample time from the driving block when determining the points to write. See "Specifying Sample Time" in the online documentation for more information.

During the simulation, the block writes data to an internal buffer. When the simulation is completed or paused, that data is written to the workspace. Its icon shows the name of the array to which the data is written.

Structure

This format consists of a structure with three fields: `time`, `signals`, and `blockName`. The `time` field is empty. The `blockName` field contains the name of the To Workspace block. The `signals` field contains a structure with three fields: `values`, `dimensions`, and `label`. The `values` field contains the array of signal values. The `dimensions` field specifies the dimensions of the values array. The `label` field contains the label of the input line.

Structure with Time

This format is the same as Structure except that the time field contains a vector of simulation time steps.

Using Saved Data with a From Workspace Block

If the data written using a To Workspace block is intended to be played back in another simulation using a From Workspace block, use the `Structure with Time` format to save the data.

Examples

In a simulation where the start time is 0, the **Maximum number of sample points** is 100, the **Decimation** is 1, and the **Sample time** is 0.5. The To Workspace block collects a maximum of 100 points, at time values of 0, 0.5, 1.0, 1.5, ..., seconds. Specifying a **Decimation** value of `1` directs the block to write data at each step.

In a similar example, the **Maximum number of sample points** is 100 and the **Sample time** is 0.5, but the **Decimation** is 5. In this example, the block collects up to 100 points, at time values of 0, 2.5, 5.0, 7.5, ..., seconds. Specifying a **Decimation** value of `5` directs the block to write data at every fifth sample. The sample time ensures that data is written at these points.

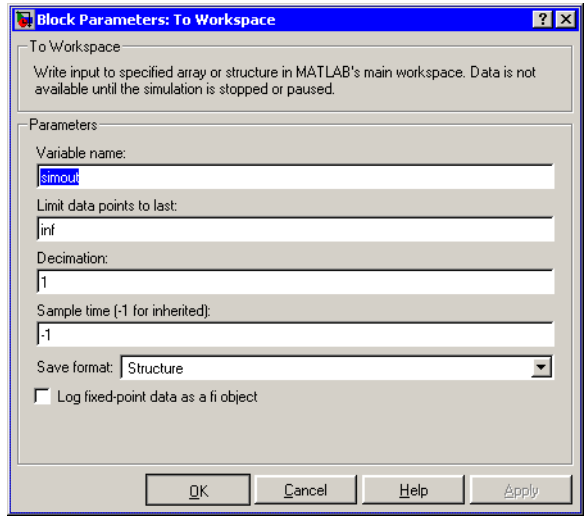
In another example, all parameters are as defined in the first example except that the **Limit data points to last** is 3. In this case, only the last three sample points collected are written to the workspace. If the simulation stop time is 100, data corresponds to times 99.0, 99.5, and 100.0 seconds (three points).

Data Type Support

The To Workspace block can save real or complex inputs of any data type supported by Simulink, including fixed-point data types, to the MATLAB workspace.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box



Variable name
The name of the array that holds the data.

Limit data points to last
The maximum number of input samples to be saved. The default is `1000` samples.

Decimation
A decimation factor. The default is `1`.

Sample time
The sample time at which to collect points. See "Specifying Sample Time" in the online documentation for more information.

Save format
Format in which to save simulation output to the workspace. The default is `structure`.

Log fixed-point data as a fi object
Select to log fixed-point data to the MATLAB workspace as a Simulink Fixed-Point `fi` object. Otherwise, fixed-point data is logged to the workspace as `double`.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Dimensionalized	Yes

Transfer Fcn

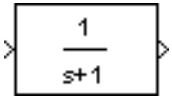
Purpose

Implement a linear transfer function

Library

Continuous

Description



The Transfer Fcn block implements a transfer function where the input (u) and output (y) can be expressed in transfer function form as the following equation

$$H(s) = \frac{y(s)}{u(s)} = \frac{num(s)}{den(s)} = \frac{num(1)s^{nn-1} + num(2)s^{nn-2} + \ldots + num(nn)}{den(1)s^{nd-1} + den(2)s^{nd-2} + \ldots + den(nd)}$$

where *nn* and *nd* are the number of numerator and denominator coefficients, respectively. *num* and *den* contain the coefficients of the numerator and denominator in descending powers of *s*. *num* can be a vector or matrix, *den* must be a vector, and both are specified as parameters on the block dialog box. The order of the denominator must be greater than or equal to the order of the numerator.

A Transfer Fcn block takes a scalar input. If the numerator of the block's transfer function is a vector, the block's output is also scalar. However, if the numerator is a matrix, the transfer function expands the input into an output vector equal in width to the number of rows in the numerator. For example, a two-row numerator results in a block with scalar input and vector output. The width of the output vector is two.

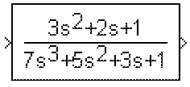
Initial conditions are preset to zero. If you need to specify initial conditions, convert to state-space form using `tf2ss` and use the State-Space block. The `tf2ss` utility provides the A, B, C, and D matrices for the system. For more information, type `help tf2ss` or consult the *Control System Toolbox documentation*.

Transfer Fcn Display

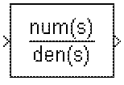
The numerator and denominator are displayed on the Transfer Fcn block depending on how they are specified:

- If each is specified as an expression, a vector, or a variable enclosed in parentheses, the icon shows the transfer function with the specified coefficients and powers of *s*. If you specify a variable in parentheses,

the variable is evaluated. For example, if you specify **Numerator** as `[3,2,1]` and **Denominator** as `(den)` where `den` is `[7,5,3,1]`, the block looks like this:



- If each is specified as a variable, the block shows the variable name followed by `(s)`. For example, if you specify **Numerator** as `num` and **Denominator** as `den`, the block looks like this:



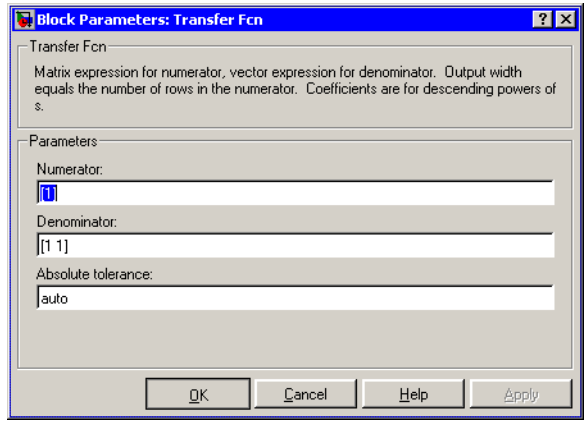
Specifying the Absolute Tolerance for the Block's States

By default Simulink uses the absolute tolerance value specified in the **Configuration Parameters** dialog box (see "Absolute tolerance") to solve the states of the Transfer Fcn block. If this value does not provide sufficient error control, specify a more appropriate value in the **Absolute tolerance** field of the Transfer Fcn block's dialog box. The value that you specify is used to solve all the block's states.

Data Type Support

The Transfer Fcn block accepts and outputs signals of type `double`.

Parameters and Dialog Box



Numerator

The row vector of numerator coefficients. A matrix with multiple rows can be specified to generate multiple output. The default is `[1]`.

Denominator

The row vector of denominator coefficients. The default is `[1 1]`.

Absolute tolerance

Absolute tolerance used to solve the block's states. You can enter `auto` or a numeric value. If you enter `auto`, Simulink determines the absolute tolerance (see "Specifying Variable-Step Solver Error Tolerances"). If you enter a numeric value, Simulink uses the specified value to solve the block's states. Note that a numeric value overrides the setting for the absolute tolerance in the **Configuration Parameters** dialog box.

Characteristics

Direct Feedthrough	Only if the lengths of the **Numerator** and **Denominator** parameters are equal
Sample Time	Continuous

Scalar Expansion	No
States	Length of **Denominator** -1
Dimensionalized	Yes, in the sense that the block expands scalar input into vector output when the transfer function numerator is a matrix. See the preceding block description.
Zero Crossing	No

Transfer Fcn Direct Form II

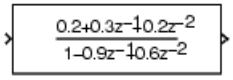
Purpose

Implement a Direct Form II realization of a transfer function

Library

Additional Math & Discrete / Additional Discrete

Description



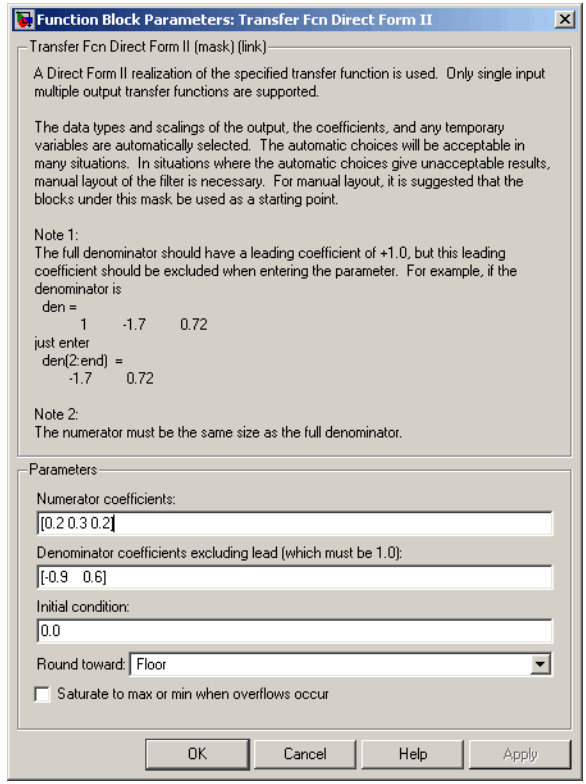
The Transfer Fcn Direct Form II block implements a Direct Form II realization of the transfer function specified by the **Numerator coefficients** and the **Denominator coefficients excluding lead** parameters. The block only supports single input-single output transfer functions.

The block automatically selects the data types and scalings of the output, the coefficients, and any temporary variables.

Data Type Support

The Transfer Fcn Direct Form II block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Numerator coefficients
Specify the numerator coefficients.

Denominator coefficients excluding lead
Specify the denominator coefficients, excluding the leading coefficient, which must be 1.0.

Initial condition
Set the initial condition.

Round toward
Rounding mode for the fixed-point output.

Saturate to max or min when overflows occur
If selected, fixed-point overflows saturate.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

See Also

Transfer Fcn Direct Form II Time Varying

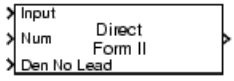
Purpose

Implement a time varying Direct Form II realization of a transfer function

Library

Additional Math & Discrete / Additional Discrete

Description



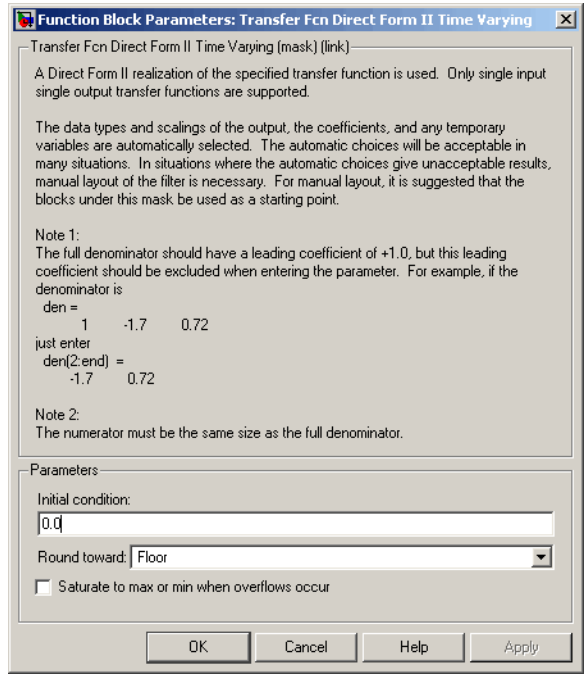
The Transfer Fcn Direct Form II Time Varying block implements a Direct Form II realization of the specified transfer function. The block only supports single input-single output transfer functions.

The block automatically selects the data types and scalings of the output, the coefficients, and any temporary variables.

Data Type Support

The Transfer Fcn Direct Form II Time Varying block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Initial condition
Set the initial condition.

Round toward
Rounding mode for the fixed-point output.

Saturate to max or min when overflows occur
If selected, fixed-point overflows saturate.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

See Also

Transfer Fcn Direct Form II

Purpose

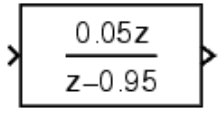
Implement a discrete-time first order transfer function

Library

Discrete

Description

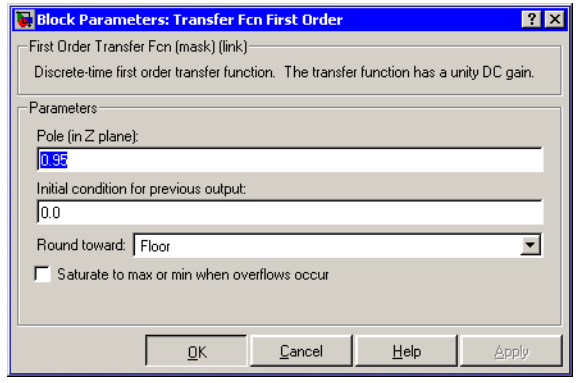
The Transfer Fcn First Order block implements a discrete-time first order transfer function of the input. The transfer function has a unity DC gain.



Data Type Support

The Transfer Fcn First Order block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Pole (in Z plane)
Set the pole.

Initial condition for previous output
Set the initial condition for the previous output.

Round toward
Rounding mode for the fixed-point output.

Saturate to max or min when overflows occur
If selected, fixed-point overflows saturate.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

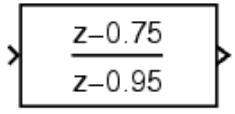
Purpose

Implement a discrete-time lead or lag compensator

Library

Discrete

Description



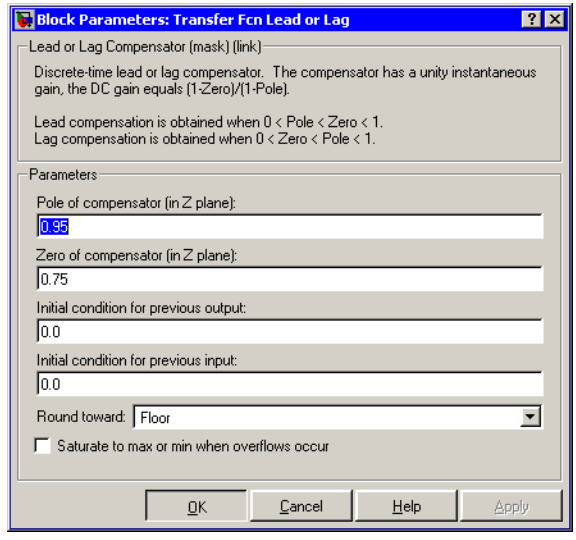
The Transfer Fcn Lead or Lag block implements a discrete-time lead or lag compensator of the input. The instantaneous gain of the compensator is one, and the DC gain is equal to (1-z)/(1-p), where z is the zero and p is the pole of the compensator.

The block implements a lead compensator when $0 < z < p < 1$, and implements a lag compensator when $0 < p < z < 1$.

Data Type Support

The Transfer Fcn Lead or Lag block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Pole of compensator (in Z plane)

Set the pole.

Zero of compensator (in Z plane)
Set the zero.

Initial condition for previous output
Set the initial condition for the previous output.

Initial condition for previous input
Set the initial condition for the previous input.

Round toward
Rounding mode for the fixed-point output.

Saturate to max or min when overflows occur
If selected, fixed-point overflows saturate.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

Purpose

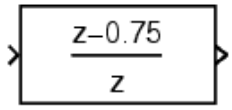
Implement a discrete-time transfer function that has a real zero and no pole

Library

Discrete

Description

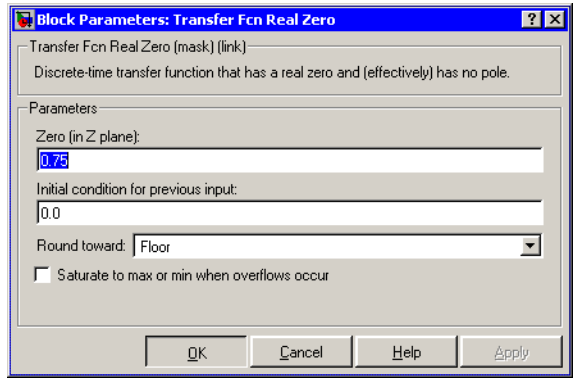
The Transfer Fcn Real Zero block implements a discrete-time transfer function that has a real zero and effectively has no pole.



Data Type Support

The Transfer Fcn Real Zero block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Zero (in Z plane)
Set the zero.

Initial condition for previous input
Set the initial condition for the previous input.

Round toward
Rounding mode for the fixed-point output.

Saturate to max or min when overflows occur
If selected, fixed-point overflows saturate.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

Purpose

Delay the input by a given amount of time

Library

Continuous

Description

The Transport Delay block delays the input by a specified amount of time. It can be used to simulate a time delay.

At the start of the simulation, the block outputs the **Initial input** parameter until the simulation time exceeds the **Time delay** parameter, when the block begins generating the delayed input. The **Time delay** parameter must be nonnegative.

The block stores input points and simulation times during a simulation in a buffer whose initial size is defined by the **Initial buffer size** parameter. If the number of points exceeds the buffer size, the block allocates additional memory and Simulink displays a message after the simulation that indicates the total buffer size needed. Because allocating memory slows down the simulation, define this parameter value carefully if simulation speed is an issue. For long time delays, this block might use a large amount of memory, particularly for dimensionalized input.

When output is required at a time that does not correspond to the times of the stored input values, the block interpolates linearly between points. When the delay is smaller than the step size, the block extrapolates from the last output point, which can produce inaccurate results. Because the block does not have direct feedthrough, it cannot use the current input to calculate its output value. To illustrate this point, consider a fixed-step simulation with a step size of 1 and the current time at t = 5. If the delay is 0.5, the block needs to generate a point at t = 4.5. Because the most recent stored time value is at t = 4, the block performs forward extrapolation.

The Transport Delay block does not interpolate discrete signals. Instead, it returns the discrete value at the required time.

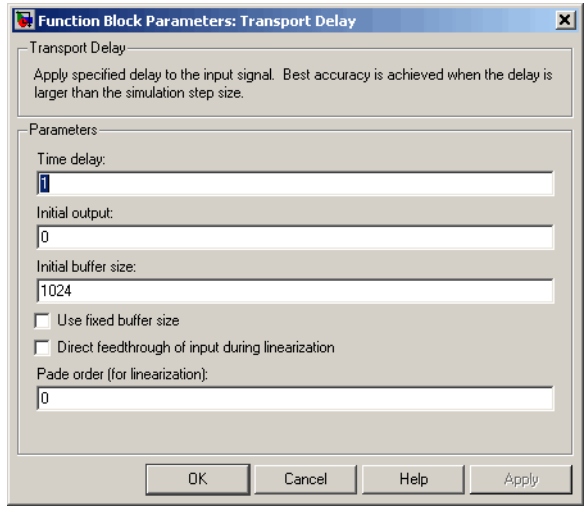
This block differs from the Unit Delay block, which delays and holds the output on sample hits only.

Using `linmod` to linearize a model that contains a Transport Delay block can be troublesome. For more information about ways to avoid the problem, see "Linearizing Models" in the Using Simulink documentation.

Data Type Support

The Transport Delay block accepts and outputs real signals of type `double`.

Parameters and Dialog Box



Time delay

The amount of simulation time that the input signal is delayed before being propagated to the output. The value must be nonnegative.

Initial output

Specifies use of a fixed-size buffer to save input data from previous time steps. The **Initial buffer size** parameter specifies the buffer's size. If the buffer is full, new data replaces data already in the buffer. Simulink uses linear extrapolation to estimate the output value if it is not in the buffer. This option can save memory

if the input data is linear. If the input is not linear, this option may yield inaccurate results.

Note ERT or GRT code generation uses a fixed-size buffer even if you do not select this check box.

Initial buffer size
The initial memory allocation for the number of points to store.

Use fixed buffer size
Select this check box to set the buffer size to store history data. This size equals the value of the **Initial buffer size** parameter. If the value of **Time delay** is not appropriate for this buffer size, the block estimates a more appropriate value. For ERT or GRT code generation, the buffer size is set even if you do not select this check box.

Direct feedthrough of input during linearization
Causes the block to output its input during linearization and trim. This sets the block's mode to direct feedthrough.

Enabling this check box can cause a change in the ordering of states in the model when using the functions `linmod`, `dlinmod`, or `trim`. To extract this new state ordering, use the following commands.

First compile the model using the following command, where `model` is the name of the Simulink model.

```
[sizes, x0, x_str] = model([],[],[],'lincompile');
```

Next, terminate the compilation with the following command.

```
model([],[],[],'term');
```

The output argument, `x_str`, which is a cell array of the states in the Simulink model, contains the new state ordering. When passing a vector of states as input to the `linmod`, `dlinmod`, or `trim` functions, the state vector must use this new state ordering.

Pade order (for linearization)

The order of the Pade approximation for linearization routines. The default value is 0, which results in a unity gain with no dynamic states. Setting the order to a positive integer `n` adds `n` states to your model, but results in a more accurate linear model of the transport delay.

Characteristics

Direct Feedthrough	No
Sample Time	Continuous
Scalar Expansion	Yes, of input and all parameters except **Initial buffer size**
Dimensionalized	Yes
Zero Crossing	No

Purpose

Add a trigger port to a subsystem or function-call model

Library

Ports & Subsystems

Description

Adding a Trigger block to a subsystem or a model allows its execution to be triggered by an external signal. You can configure the Trigger block to enable a change in the value of the external signal (described below) to trigger execution of a subsystem once on each integration step when the value of the signal that passes through the trigger port changes in a specifiable way (see“Triggered Subsystems”). You can also configure the Trigger block to accept a function-call trigger. This allows a Function-Call Generator block or S-function to trigger execution of a subsystem or model multiple times during a time step. A subsystem or model can contain only one Trigger block. For more information, see “Function-Call Models” and “Function-Call Subsystems”.

The **Trigger type** parameter allows you to choose the type of event that triggers execution of the subsystem:

- `rising` triggers execution of the subsystem when the control signal rises from a negative or zero value to a positive value (or zero if the initial value is negative).
- `falling` triggers execution of the subsystem when the control signal falls from a positive or a zero value to a negative value (or zero if the initial value is positive).
- `either` triggers execution of the subsystem when the signal is either rising or falling.
- `function-call` allows a Function-Call Generator or S-function to control execution of the subsystem or model.

Note The **Trigger type** must be `function-call` for Trigger ports at the root-level of a model. In other words, only function-call signals can trigger execution of a model.

You can output the trigger signal by selecting the **Show output port** check box. Selecting this option allows the system to determine what caused the trigger. The width of the signal is the width of the triggering signal. The signal value is

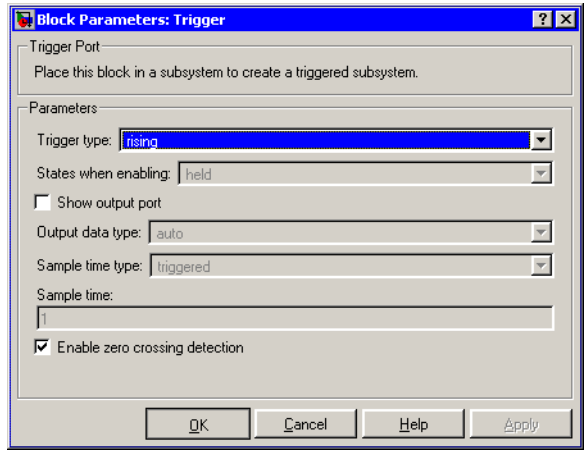
- 1 for a signal that causes a rising trigger
- -1 for a signal that causes a falling trigger
- 2 for a function-call trigger
- 0 otherwise

Data Type Support

The Trigger block accepts signals of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, see“Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box



Trigger type

The type of event that triggers execution of the subsystem.

States when enabling

This option is enabled only if you select `function-call` as the block's trigger type. It specifies whether a function-call enable trigger causes Simulink to reset the states of the subsystem containing this Trigger block to their initial values.

Selecting `held` (the default) causes Simulink to leave the states at their current values.

Selecting `reset` for this option causes Simulink to reset the states.

Selecting `inherit` causes the trigger's `held`/`reset` setting to be the same as that of the function-call initiator's parent subsystem, for example, an enabled subsystem, or the model's root system if the function-call initiator is at the model's root level. If the parent of the initiator is the model root, the inherited setting is `held`. If the trigger has multiple initiators and its **States when enabling** setting is `inherit`, the parents of all initiators must have the same `held`/`reset` setting, i.e., either all `held` or all `reset`. For more information about the **States when enabling** setting, see "Function-Call Subsystems" in the "Implementing Block Features" section of *Writing S-Functions*.

Show output port

If selected and this block is in a subsystem, Simulink displays the Trigger block output port and outputs the trigger signal. Simulink does not allow you to log the output of this port if the block's **Trigger type** is `function-call`.

Note This option is disabled for function-call Trigger blocks residing at the root-level of a model.

Output data type

Specifies the data type (`double` or `int8`) of the trigger output. If you select `auto`, Simulink sets the data type to be the same as

that of the port to which the output is connected. If the port's data type is not `double` or `int8`, Simulink signals an error.

Note The Trigger block ignores the **Data Type Override** setting of the Fixed-Point Settings interface.

Enable zero crossing detection
Select to enable zero crossing detection. For more information, see "Zero Crossing Detection" in the Using Simulink documentation.

Sample time type
This parameter is active only when **Trigger type** is set to `function-call`. Its value may be `triggered` or `periodic`. Select `periodic` if the caller of the parent function-call subsystem, for example, a Stateflow chart, calls the subsystem once per time step when the subsystem is active (enabled). Otherwise, select `triggered`. See "Using Bind Actions to Control Function-Call Subsystems" in *Using Stateflow* and the "Function-Call Subsystems" section of *Writing S-functions* for more information.

Sample time
This parameter is active only when the **Trigger type** is `function-call` and the **Sample time type** is `periodic`. Set this parameter to the sample time at which you expect the function-call subsystem that contains this block to be called. See "Specifying Sample Time" in the online documentation for information on how to the value of this parameter. Simulink displays an error if the actual rate at which the subsystem is called differs from the rate that this parameter specifies.

Characteristics

Sample Time	Determined by the sample time parameter if the trigger type is function-call and the sample time type is periodic; otherwise, by the signal at the trigger port.
Dimensionalized	Yes
Zero Crossing	Yes, if enabled

Trigger-Based Linearization

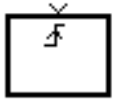
Purpose

Generate linear models in the base workspace when triggered

Library

Model-Wide Utilities

Description



When triggered, this block calls `linmod` or `dlinmod` to create a linear model for the system at the current operating point. No trimming is performed. The linear model is stored in the base workspace as a structure, along with information about the operating point at which the snapshot was taken. Multiple snapshots are appended to form an array of structures.

The name of the structure used to save the snapshots is the name of the model appended by `_Trigger_Based_Linearization`, for example, `vdp_Trigger_Based_Linearization`. The structure has the follow fields:

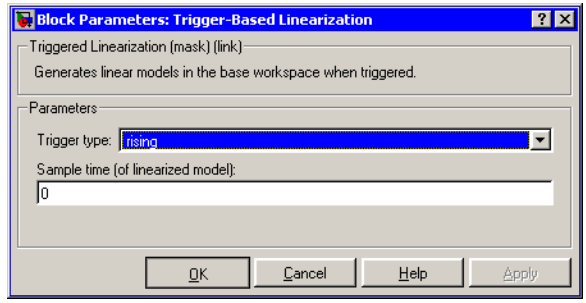
Field	Description
`a`	The A matrix of the linearization
`b`	The B matrix of the linearization
`c`	The C matrix of the linearization
`d`	The D matrix of the linearization
`StateName`	Names of the model's states
`OutputName`	Names of the model's output ports
`InputName`	Names of the model's input ports
`OperPoint`	A structure that specifies the operating point of the linearization. The structure specifies the value of the model's states (`OperPoint.x`) and inputs (`OperPoint.u`) at the operating point time (`OperPoint.t`).
`Ts`	The sample time of the linearization for a discrete linearization

Use the Time-Based Linearization block to generate linear models at predetermined times.

Data Type Support

The trigger port accepts signals of any data type supported by Simulink.

Parameters and Dialog Box



Trigger type

Type of event on the trigger input signal that triggers generation of a linear model. See the **Trigger type** parameter of the Trigger block for an explanation of the various trigger types that you can select.

Sample time (of linearized model)

Specify a sample time to create a discrete-time linearization of the model (see “Discrete-Time System Linearization” on page 3-6).

Characteristics

Sample Time	Specified in the **Sample time** parameter
Dimensionalized	No

Triggered Subsystem

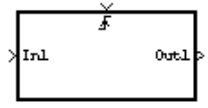
Purpose

Represent a subsystem whose execution is triggered by external input

Library

Ports & Subsystems

Description



This block is a Subsystem block that is preconfigured to serve as the starting point for creating a triggered subsystem (see “Triggered Subsystems”).

Purpose

Perform a trigonometric function

Library

Math Operations

Description

The Trigonometric Function block performs numerous common trigonometric functions.

You can select one of these functions from the **Function** list: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `atan2`, `sinh`, `cosh`, `tanh`, `asinh`, `acosh`, and `atanh`. The block output is the result of the operation of the function on the input or inputs.

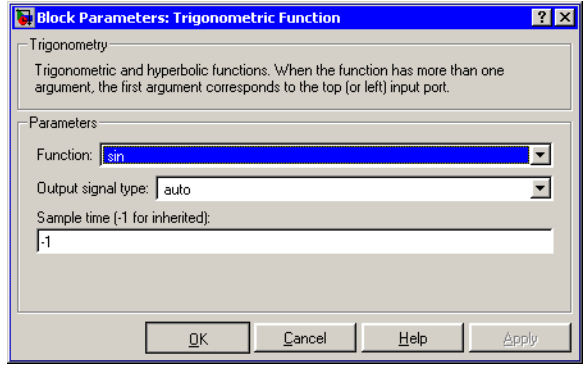
The name of the function appears on the block. If you select the `atan2` function, the block displays two inputs. The first (upper) input is the y-axis or complex part of the function argument. The second (lower) input is the x-axis or real part of the function argument.

Use the Trigonometric Function block instead of the Fcn block when you want dimensionalized output, because the Fcn block can produce only scalar output.

Data Type Support

The Trigonometric Function block accepts and outputs real or complex signals of type `double`.

Parameters and Dialog Box



Function
The trigonometric function.

Output signal type
Type of signal (complex or real) to output.

Sample time (-1 for inherited)
Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes, of the input when the function requires two inputs
Dimensionalized	Yes
Zero Crossing	No

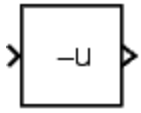
Purpose

Negate the input

Library

Math Operations

Description



The Unary Minus block negates the input. The block accepts only signed data types.

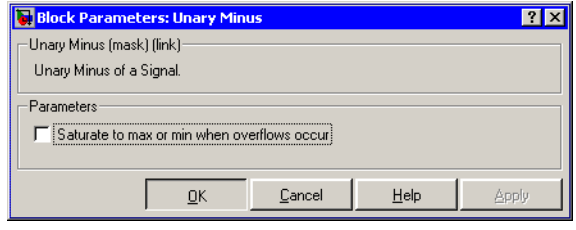
For signed data types, you cannot accurately negate the most negative value since the result is not representable by the data type. In this case, the behavior of the block is controlled by the **Saturate to max or min when overflows occur** check box. If selected, the most negative value of the data type wraps to the most positive value. If not selected, the operation has no effect. If an overflow occurs, then a warning is returned to the MATLAB command line.

For example, suppose the block input is an 8-bit signed integer. The range of this data type is from -128 to 127, and the negation of -128 is not representable. If you select the **Saturate to max or min when overflows occur** check box, then the negation of -128 is 127. If it is not selected, then the negation of -128 remains at -128.

Data Type Support

The Unary Minus block accepts signals of any data type supported by Simulink except unsigned integers, including fixed-point data types.

Parameters and Dialog Box



Saturate to max or min when overflows occur

If selected, fixed-point overflows saturate.

Characteristics

Direct Feedthrough	No
Scalar Expansion	Yes, of input or initial conditions

Purpose

Generate uniformly distributed random numbers

Library

Sources

Description

The Uniform Random Number block generates uniformly distributed random numbers over a specifiable interval with a specifiable starting seed. The seed is reset each time a simulation starts. The generated sequence is repeatable and can be produced by any Uniform Random Number block with the same seed and parameters. To generate normally distributed random numbers, use the `Random Number` block.

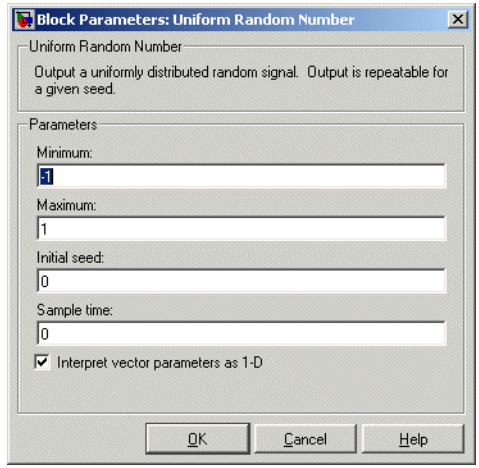
Avoid integrating a random signal, because solvers are meant to integrate relatively smooth signals. Instead, use the Band-Limited White Noise block.

The block's numeric parameters must be of the same dimensions after scalar expansion. If the **Interpret vector parameters as 1-D** option is off, the block outputs a signal of the same dimensions and dimensionality as the parameters. If the **Interpret vector parameters as 1-D** option is on and the numeric parameters are row or column vectors (i.e., single row or column 2-D arrays), the block outputs a vector (1-D array) signal; otherwise, the block outputs a signal of the same dimensions as the parameters.

Data Type Support

The Uniform Random Number block outputs a real signal of type `double`.

Parameters and Dialog Box



Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters" in the online Simulink documentation for details.

Minimum
The minimum of the interval. The default is `-1`.

Maximum
The maximum of the interval. The default is `1`.

Initial seed
The starting seed for the random number generator. The default is `0`.

Sample time
The sample period. The default is `0`. See "Specifying Sample Time" in the online documentation for more information.

Interpret vector parameters as 1-D
If selected, column or row matrix values for the Step block's numeric parameters result in a vector output signal; otherwise, the block outputs a signal of the same dimensionality as the parameters. If this option is not selected, the block always

outputs a signal of the same dimensionality as the block's numeric parameters.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

Unit Delay

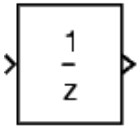
Purpose

Delay a signal one sample period

Library

Discrete

Description



The Unit Delay block delays its input by the specified sample period. This block is equivalent to the z^{-1} discrete-time operator. The block accepts one input and generates one output, which can be either both scalar or both vector. If the input is a vector, all elements of the vector are delayed by the same sample period.

You specify the block output for the first sampling period with the **Initial conditions** parameter. Careful selection of this parameter can minimize unwanted output behavior. The time between samples is specified with the **Sample time** parameter. A setting of `-1` means the sample time is inherited.

Note The Unit Delay block accepts continuous signals. When it has a continuous sample time, the block is equivalent to the Simulink Memory block.

The Unit Delay block provides a mechanism for discretizing one or more signals in time.

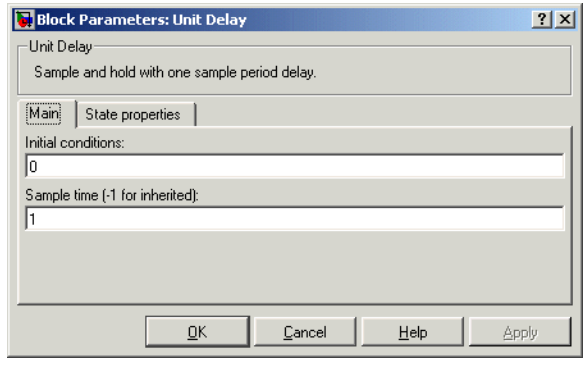
Note Do not use the Unit Delay block to create a slow-to-fast transition between blocks operating at different sample rates. Use the Rate Transition block instead, more information about which is available in the online Simulink block reference.

Data Type Support

The Unit Delay block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types. If the data type of the input signal is user-defined, the initial condition must be zero.

For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box



Initial conditions

The output of the simulation for the first sampling period, during which the output of the Unit Delay block is otherwise undefined. The **Initial conditions** parameter is converted from a double to the input data type offline using round-to-nearest and saturation.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

The **State properties** pane of this block pertains to code generation and has no effect on model simulation. See "Block States: Storing and Interfacing" in the Real-Time Workshop documentation for more information.

Characteristics

Direct Feedthrough	No
Sample Time	Specified in the **Sample time** parameter

Scalar Expansion	Yes, of input or initial conditions
States	Yes–inherited from driving block for nonfixed-point data types.
Dimensionalized	Yes
Zero Crossing	No

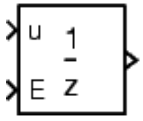
Purpose

Delay a signal one sample period, if the external enable signal is on

Library

Additional Math & Discrete / Additional Discrete

Description



The Unit Delay Enabled block delays a signal by one sample period when the external enable signal `E` is on. While the enable is off, the block is disabled. It holds the current state at the same value and outputs that value. The enable signal is on when `E` is not 0, and off when `E` is 0.

You specify the block output for the first sampling period with the value **Initial condition** parameter.

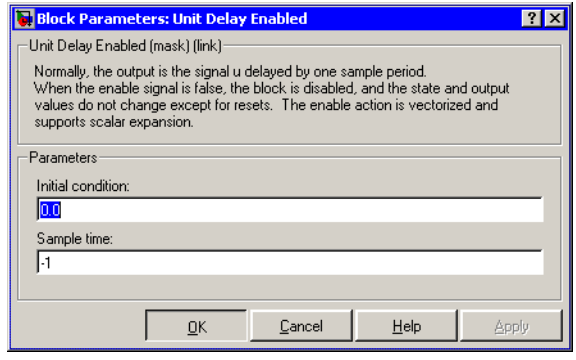
The output data type is the same as the input `u` data type. The data type of the input `u` and the enable `E` can be any data type.

You input the sample time with the **Sample time** parameter**.** A setting of `-1` means the **Sample time** is inherited.

Data Type Support

The Unit Delay Enabled block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Initial condition

Initial condition.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	No
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

See Also

Unit Delay, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable External IC, Unit Delay External IC, Unit Delay Resettable, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

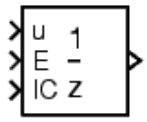
Purpose

Delay a signal one sample period, if the external enable signal is on, with an external initial condition

Library

Additional Math & Discrete / Additional Discrete

Description



The Unit Delay Enabled External IC block delays a signal by one sample period when the enable signal `E` is on. While the enable is off, the block holds the current state at the same value and outputs that value. The enable `E` is on when `E` is not 0, and off when `E` is 0.

The initial condition of this block is given by the signal `IC`.

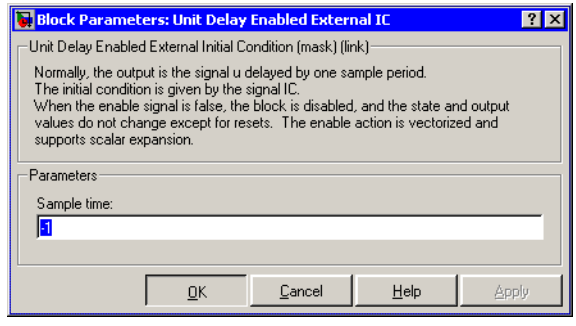
The input `u` and `IC` data types must be the same, and are any data type. The output data type is the same as `u` and `IC`. The enable `E` is any data type.

You specify the time between samples with the **Sample time** parameter**.** A setting of `-1` means the **Sample time** is inherited.

Data Type Support

The Unit Delay Enabled External IC block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Unit Delay Enabled External IC

Characteristics

Direct Feedthrough	Yes, of the reset input port No, of the enable input port Yes, of the external `IC` port
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

See Also

Unit Delay, Unit Delay Enabled, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable External IC, Unit Delay External IC, Unit Delay Resettable, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

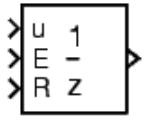
Purpose

Delay a signal one sample period, if the external enable signal is on, with an external Boolean reset

Library

Additional Math & Discrete / Additional Discrete

Description



The Unit Delay Enabled Resettable block combines the features of the Unit Delay Enabled and Unit Delay Resettable blocks.

The block can reset its state based on an external reset signal `R`. When the enable signal `E` is on and the reset signal `R` is false, the block outputs the input signal delayed by one sample period.

When the enable signal `E` is on and the reset signal `R` is true, the block resets the current state to the initial condition, specified by the **Initial condition** parameter, and outputs that state delayed by one sample period.

When the enable signal is off, the block is disabled, and the state and output do not change except for resets. The enable signal is on when `E` is not 0, and off when `E` is 0.

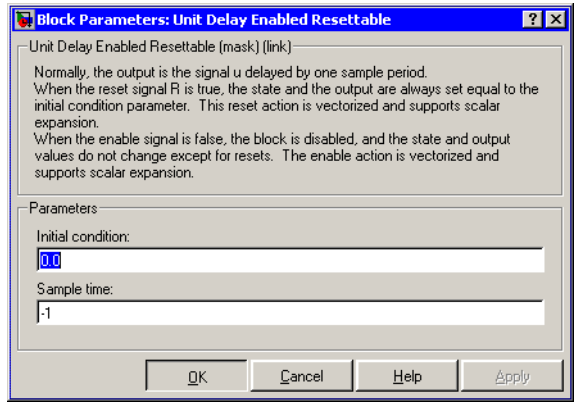
You specify the time between samples with the **Sample time** parameter**.** A setting of `-1` means the **Sample time** is inherited.

Data Type Support

The Unit Delay Enabled Resettable block accepts signals of any data type supported by Simulink, including fixed-point data types.

Unit Delay Enabled Resettable

Parameters and Dialog Box



Initial condition

The initial output of the simulation.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	No, of the input port No, of the enable port Yes, of the reset port
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

See Also

Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable External IC, Unit Delay External IC, Unit Delay Resettable, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit

Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

Unit Delay Enabled Resettable External IC

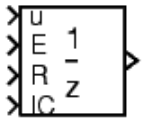
Purpose

Delay a signal one sample period, if the external enable signal is on, with an external Boolean reset and initial condition

Library

Additional Math & Discrete / Additional Discrete

Description



The Unit Delay Enabled Resettable External IC block combines the features of the Unit Delay Enabled, Unit Delay External IC, and Unit Delay Resettable blocks.

The block can reset its state based on an external reset signal `R`. When the enable signal `E` is on and the reset signal `R` is false, the block outputs the input signal delayed by one sample period.

When the enable signal `E` is on and the reset signal `R` is true, the block resets the current state to the initial condition given by the signal `IC`, and outputs that state delayed by one sample period.

When the enable signal is off, the block is disabled, and the state and output do not change except for resets. The enable signal is on when `E` is not 0, and off when `E` is 0.

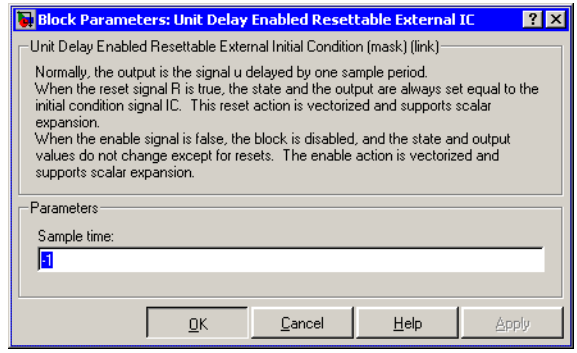
The output data type is the same as the input `u` and the initial condition `IC` data type, which can be any data type, but must be the same. The enable `E` and reset `R` can be any data type.

You specify the time between samples with the **Sample time** parameter**.** A setting of `-1` means the **Sample time** is inherited.

Data Type Support

The Unit Delay Enabled Resettable External IC block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	No, of the input port No, of the enable port Yes, of the enable port Yes, of the external `IC` port
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

See Also

Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay External IC, Unit Delay Resettable, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

Unit Delay External IC

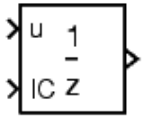
Purpose

Delay a signal one sample period, with an external initial condition

Library

Additional Math & Discrete / Additional Discrete

Description



The Unit Delay External IC block delays its input by one sample period. This block is equivalent to the z^{-1} discrete-time operator. The block accepts one input and generates one output, both of which can be scalar or vector. If the input is a vector, all elements of the vector are delayed by the same sample period.

The block's output for the first sample period is equal to the signal `IC`.

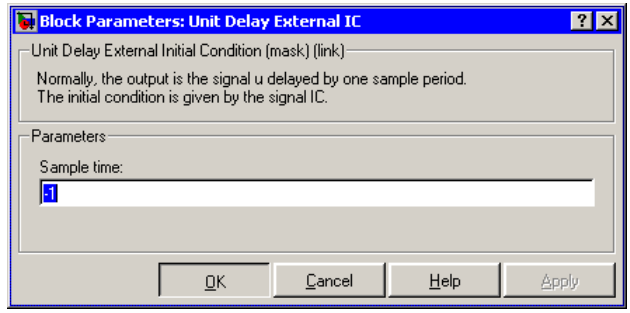
The input `u` and initial condition `IC` data types must be the same, and are any data type.

You specify the time between samples with the **Sample time** parameter**.** A setting of `-1` means the **Sample time** is inherited.

Data Type Support

The Unit Delay External IC block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	No, of the input port Yes, of the external `IC` port
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

See Also

Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable External IC, Unit Delay Resettable, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

Unit Delay Resettable

Purpose

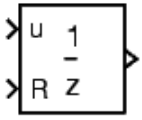
Delay a signal one sample period, with an external Boolean reset

Library

Additional Math & Discrete / Additional Discrete

Description

The Unit Delay Resettable block delays a signal one sample period.



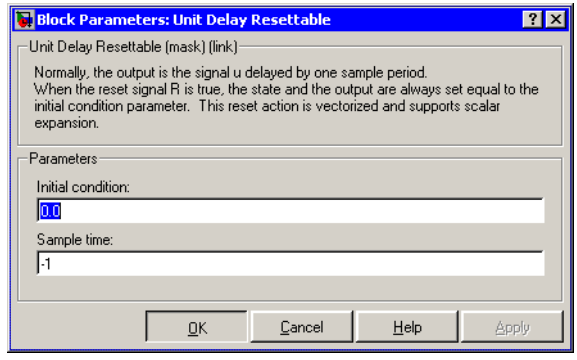
The block can reset its state based on an external reset signal `R`. The block has two input ports, one for the input signal `u` and the other for the external reset signal `R`. When the reset signal is false, the block outputs the input signal delayed by one time step. When the reset signal is true, the block resets the current state to the initial condition, specified by the **Initial condition** parameter, and outputs that state delayed by one time step.

You specify the time between samples with the **Sample time** parameter**.** A setting of `-1` means the **Sample time** is inherited.

Data Type Support

The Unit Delay Resettable block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Initial condition

Specify the initial output of the simulation.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	No, of the input port Yes, of the reset port
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

See Also

Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable External IC, Unit Delay External IC, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

Unit Delay Resettable External IC

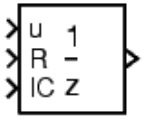
Purpose

Delay a signal one sample period, with an external Boolean reset and initial condition

Library

Additional Math & Discrete / Additional Discrete

Description



The Unit Delay Resettable External IC block delays a signal one sample period.

The block can reset its state based on an external reset signal `R`. The block has two input ports, one for the input signal `u` and the other for the reset signal `R`. When the reset signal is false, the block outputs the input signal delayed by one time step. When the reset signal is true, the block resets the current state to the initial condition given by the signal `IC` and outputs that state delayed by one time step.

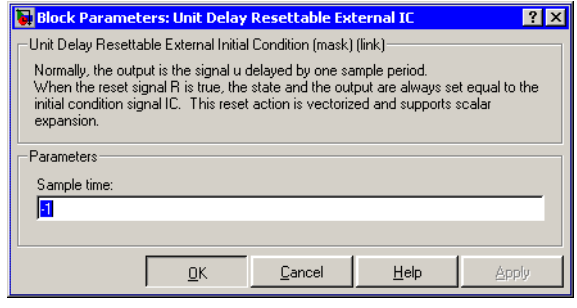
The input `u` and initial condition `IC` must be the same data type, but can be any data type. The output is the same data type as the inputs `u` and `IC`. The reset `R` can be any data type.

You specify the time between samples with the **Sample time** parameter**.** A setting of `-1` means the **Sample time** is inherited.

Data Type Support

The Unit Delay Resettable External IC block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	No, of the input port Yes, of the reset port Yes, of the external `IC` port
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

See Also

Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable External IC, Unit Delay External IC, Unit Delay Resettable, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

Unit Delay With Preview Enabled

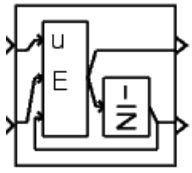
Purpose

Output the signal and the signal delayed by one sample period, if the external enable signal is on

Library

Additional Math & Discrete / Additional Discrete

Description



The Unit Delay With Preview Enabled block supports calculations that have feedback and depend on the current input.

The block has two output ports. When the external enable signal `E` is on, the upper port outputs the signal and the lower port outputs the signal delayed by one sample period. The block has two input ports, one for the input signal `u` and the other for the enable signal `E`.

When the enable signal `E` is off, the block is disabled, and the state and output values do not change, except for resets. The enable signal is on when `E` is not 0, and off when `E` is 0.

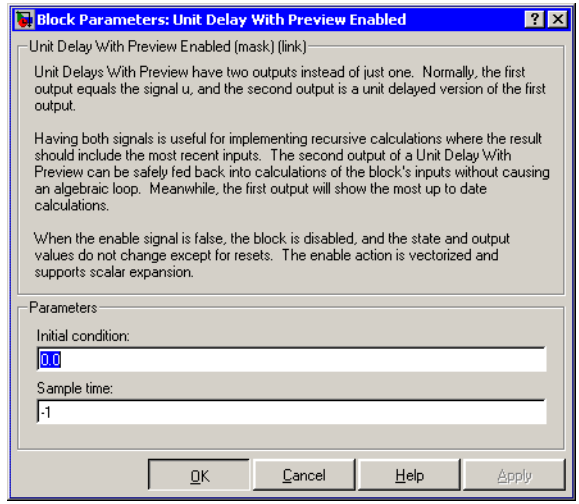
The input `u` and initial condition `IC` must be the same data type, but can be any data type. The output is the same data type as the inputs `u` and `IC`. The reset `R` can be any data type.

You specify the time between samples with the **Sample time** parameter**.** A setting of `-1` means the **Sample time** is inherited.

Data Type Support

The Unit Delay With Preview Enabled block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Initial condition

Specify the initial condition.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes, to upper output port No, to lower output port
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

See Also

Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable External IC, Unit Delay External IC, Unit Delay Resettable, Unit Delay Resettable External IC, Unit Delay With Preview Enabled Resettable, Unit Delay

Unit Delay With Preview Enabled

With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

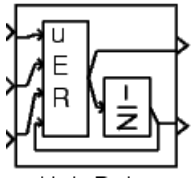
Purpose

Output the signal and the signal delayed by one sample period, if the external enable signal is on, with an external Boolean reset

Library

Additional Math & Discrete / Additional Discrete

Description



The Unit Delay With Preview Enabled Resettable block supports calculations that have feedback and depend on the current input.

The block can reset its state based on an external reset signal `R`. The block has two output ports. When the external enable signal `E` is on and the reset `R` is false, the upper port outputs the signal and the lower port outputs the signal delayed by one sample period. The block has two input ports, one for the input signal `u` and the other for the enable signal `E`.

When the enable signal `E` is on and the reset `R` is true, the block resets the current state to the initial condition given by the **Initial condition** parameter. The block outputs that state delayed by one sample time through the lower output port, and outputs the state without a delay through the upper output port.

When the Enable signal is off, the block is disabled, and the state and output values do not change, except for resets. The enable signal is on when `E` is not 0, and off when `E` is 0.

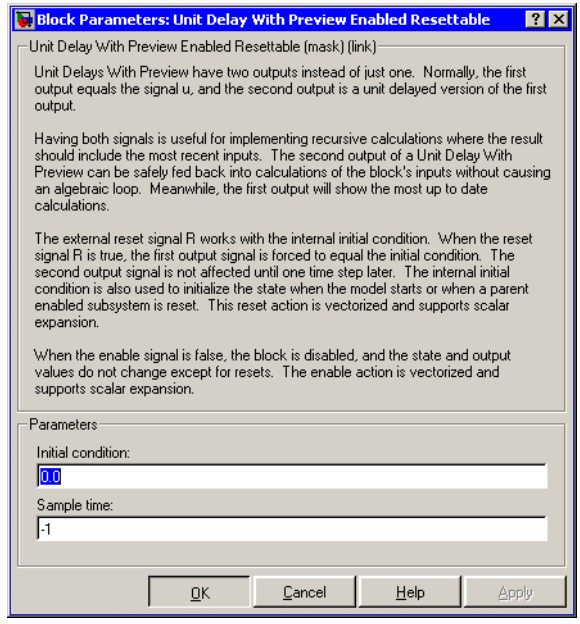
The input `u` and initial condition `IC` must be the same data type, but can be any data type. The output is the same data type as the inputs `u` and `IC`. The reset `R` can be any data type.

You specify the time between samples with the **Sample time** parameter**.** A setting of `-1` means the **Sample time** is inherited.

Data Type Support

The Unit Delay With Preview Enabled Resettable block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Initial condition

Specify the initial condition.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes, to upper output port No, to lower output port
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

See Also

Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable External IC, Unit Delay External IC, Unit Delay Resettable, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

Unit Delay With Preview Enabled Resettable External RV

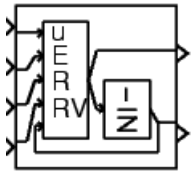
Purpose

Output the signal and the signal delayed by one sample period, if the external enable signal is on, with an external RV reset

Library

Additional Math & Discrete / Additional Discrete

Description



The Unit Delay With Preview Enabled Resettable External RV block supports calculations that have feedback and depend on the current input.

The block can reset its state based on an external reset signal `R`. The block has two output ports. When the external enable signal `E` is on and the reset `R` is false, the upper port outputs the signal and the lower port outputs the signal delayed by one sample period. The block has two input ports, one for the input signal `u` and the other for the enable signal `E`.

When the enable signal `E` is on and the reset `R` is true, the upper output signal is forced to equal the external reset signal `RV`. The lower output signal is not affected until one time step later, at which time it is equal to the external reset signal `RV` at the previous time step. The block uses the internal **Initial condition** only when the model starts or when a parent enabled subsystem is used. The internal **Initial condition** only affects the lower output signal. The first output is only affected through feedback.

When the Enable signal is off, the block is disabled, and the state and output values do not change, except for resets. The enable signal is on when `E` is not 0, and off when `E` is 0.

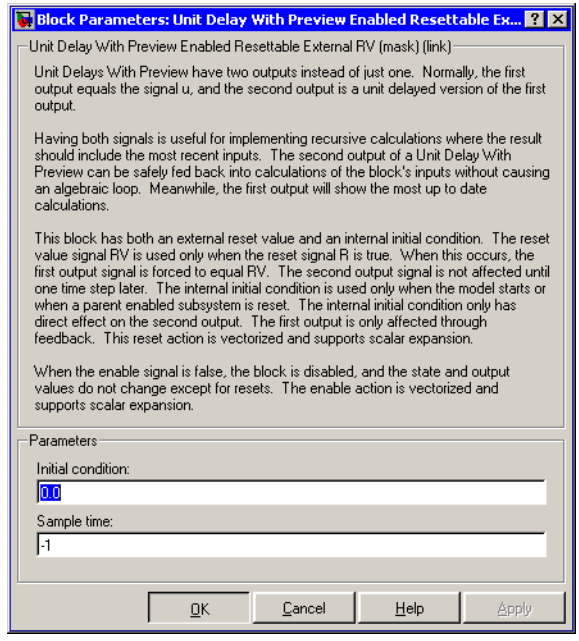
The input `u` and initial condition `IC` must be the same data type, but can be any data type. The output is the same data type as the inputs `u` and `IC`. The reset `R` can be any data type.

You specify the time between samples with the **Sample time** parameter**.** A setting of `-1` means the **Sample time** is inherited.

Data Type Support

The Unit Delay With Preview Enabled Resettable External RV block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Initial condition

Specify the initial condition.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes, to upper output port No, to lower output port
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

Unit Delay With Preview Enabled Resettable External RV

See Also

Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable External IC, Unit Delay External IC, Unit Delay Resettable, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

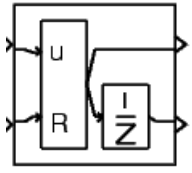
Purpose

Output the signal and the signal delayed by one sample period, with an external Boolean reset

Library

Additional Math & Discrete / Additional Discrete

Description



The Unit Delay With Preview Resettable block supports calculations that have feedback and depend on the current input.

The block can reset its state based on an external reset signal `R`. The block has two output ports. When the reset `R` is false, the upper port outputs the signal and the lower port outputs the signal delayed by one sample period.

When the reset `R` is true, the block resets the current state to the initial condition given by the **Initial condition** parameter. The block outputs that state delayed by one sample time through the lower output port, and outputs the state without a delay through the upper output port.

The input `u` and initial condition `IC` must be the same data type, but can be any data type. The output is the same data type as the inputs `u` and `IC`. The reset `R` can be any data type.

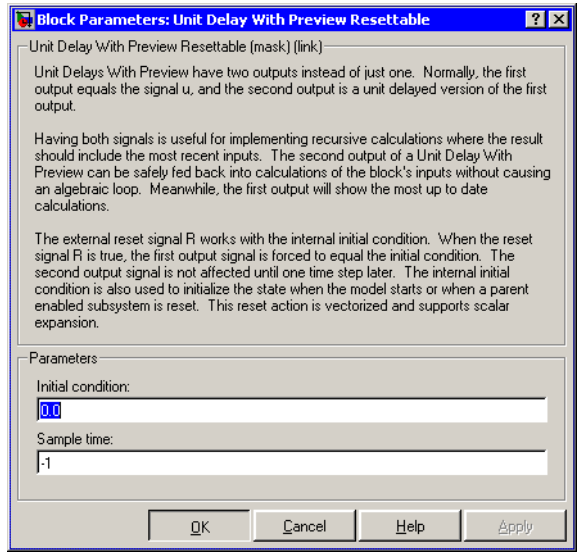
You specify the time between samples with the **Sample time** parameter**.** A setting of `-1` means the **Sample time** is inherited.

Data Type Support

The Unit Delay With Preview Resettable block accepts signals of any data type supported by Simulink, including fixed-point data types.

Unit Delay With Preview Resettable

Parameters and Dialog Box



Initial condition

Specify the initial condition.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes, to upper output port No, to lower output port
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

See Also

Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable

External IC, Unit Delay External IC, Unit Delay Resettable, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable External RV

Unit Delay With Preview Resettable External RV

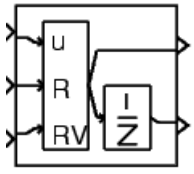
Purpose

Output the signal and the signal delayed by one sample period, with an external RV reset

Library

Additional Math & Discrete / Additional Discrete

Description



The Unit Delay With Preview Resettable External RV block supports calculations that have feedback and depend on the current input.

The block can reset its state based on an external reset signal `R`. The block has two output ports. When the external reset `R` is false, the upper port outputs the signal and the lower port outputs the signal delayed by one sample period.

When the external reset `R` is true, the upper output signal is forced to equal the external reset signal `RV`. The lower output signal is not affected until one time step later, at which time it is equal to the external reset signal `RV` at the previous time step. The block uses the internal **Initial condition** only when the model starts or when a parent enabled subsystem is used. The internal **Initial condition** only affects the lower output signal. The first output is only affected through feedback.

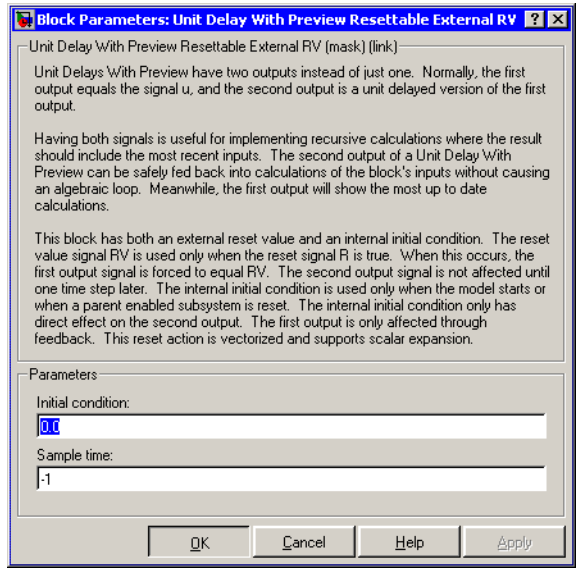
The input `u` and initial condition `IC` must be the same data type, but can be any data type. The output is the same data type as the inputs `u` and `IC`. The reset `R` can be any data type.

You specify the time between samples with the **Sample time** parameter**.** A setting of `-1` means the **Sample time** is inherited.

Data Type Support

The Unit Delay With Preview Resettable External RV block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Initial condition

Specify the initial condition.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes, to upper output port No, to lower output port
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

See Also

Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable

Unit Delay With Preview Resettable External RV

External IC, Unit Delay External IC, Unit Delay Resettable, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable

Purpose

Delay the input by a variable amount of time

Library

Continuous

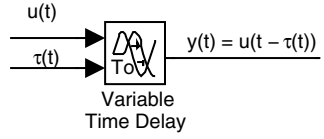
Description

The Variable Transport Delay and Variable Time Delay appear as two blocks in the Simulink block library. However, they are actually the same built-in Simulink block with different settings of a **Select delay type** parameter. This parameter allows you to specify that the block operate in either of the following modes.

Variable Time Delay

In this mode, the block has a data input, a time delay input, and a data output. The block's output at the current time step equals the value of its data input at a previous time equal to the current simulation time minus a delay time specified by the block's time delay input.

$$y(t) = u(t - t_0) = u(t - \tau(t))$$



The block's **Maximum delay** parameter defines the largest value the time delay input can have. The block clips values of the delay that exceed this value. The **Maximum delay** must be greater than or equal to zero. If the time delay becomes negative, the block clips it to zero and issues a warning message.

During the simulation, the block stores time and input value pairs in an internal buffer. At the start of the simulation, the block outputs the value of the **Initial output** parameter until the simulation time exceeds the time delay input. Then, at each simulation step, the block outputs the signal at the time that corresponds to the current simulation time minus the delay time.

When output is required at a time that does not correspond to the times of the stored input values and the solver is a continuous solver, the block interpolates linearly between points. If the time delay is smaller than the step size, the block extrapolates an output point from a previous point. For example, consider a fixed-step simulation with a step size of 1 and the current time at t = 5. If the delay is 0.5, the block needs to generate a point at t = 4.5. Because the most recent stored time value is at t = 4, the block extrapolates its output at t = 5 from the input at t = 4.

Extrapolating forward from the previous time step can produce a less accurate result than extrapolating back from the current time step. However, the block cannot use the current input to calculate its output value because the input port does not have direct feedthrough.

If the model specifies a discrete solver, the block does not interpolate between time steps. Instead, it returns the nearest stored value that precedes the required value.

Variable Transport Delay

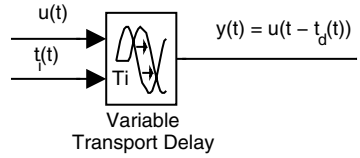
In this mode, the block's output at the current time step is equal to the value of its data input at an earlier time equal to the current time minus a transportation delay

$$y(t) = u(t - t_d(t))$$

Simulink finds the transportation delay, $t_d(t)$, by solving the following equation

$$\int_{t-t_d(t)}^{t} \frac{1}{t_i(\tau)} d\tau = 1$$

This equation involves an instantaneous time delay, $t_i(t)$, given by the block's time delay input.



For example, suppose you want to use this block to model the flow of a fluid through a pipe where the speed of the flow varies with time. In this case, the time delay input to the block would be

$$t_i(t) = \frac{L}{v_i(t)}$$

where L is the length of the pipe and $v_i(t)$ is the speed of the fluid.

Data Type Support

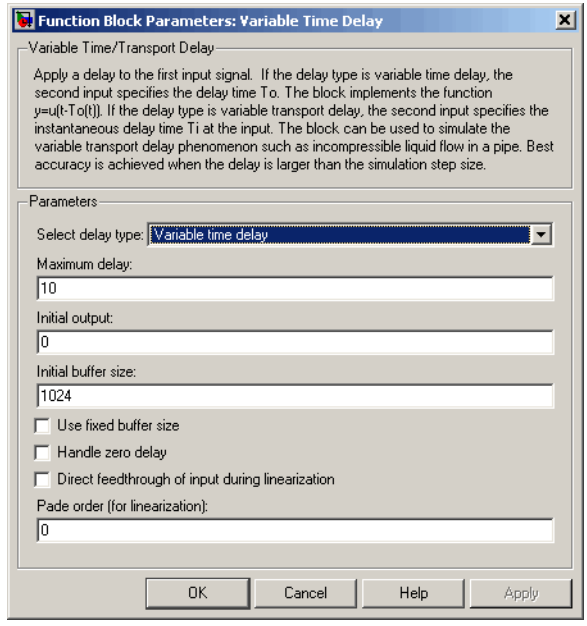
The Variable Time Delay and Variable Transport Delay blocks accept and output real signals of type `double`.

Parameters and Dialog Box

The block's parameters and dialog box differ, depending on whether it is operating in variable time or variable transport delay mode. Most parameters exist in both modes. The following sections note parameters that exist only in one mode.

Variable Time Delay Parameters and Dialog Box

The dialog box for the Variable Time Delay block appears as follows.



Select delay type

The delay type of the block. The Variable Time Delay block in the Simulink library has a preset value of `Variable time delay`. The Variable Transport Delay block has a preset value of `Variable transport delay`.

Maximum delay

The maximum value of the time delay input. The value cannot be negative. The default is `10`.

Initial output

The output generated by the block until the simulation time first exceeds the time delay input. The default is `0`. Simulink does not allow the initial output of this block to be `inf` or `NaN`.

Initial Buffer size

Initial size of the buffer used to store previous input values. The default is `1024`.

Use fixed buffer size

Specifies use of a fixed-size buffer to save input data from previous time steps. The **Initial buffer size** parameter specifies the buffer's size. If the buffer is full, new data replaces data already in the buffer. Simulink uses linear extrapolation to estimate the output value if it is not in the buffer. This option can save memory if the input data is linear. If the input is not linear, this option may yield inaccurate results.

Note ERT or GRT code generation uses a fixed-size buffer even if you do not select this check box.

Handle zero delay

For `Variable time delay` mode. Turns this block into a direct feedthrough block.

Direct feedthrough of input during linearization

Causes the block to output its input during linearization and trim. This sets the block's mode to direct feedthrough.

Enabling this check box can cause a change in the ordering of states in the model when using the functions `linmod`, `dlinmod`, or `trim`. To extract this new state ordering, use the following commands.

First compile the model using the following command, where `model` is the name of the Simulink model.

```
[sizes, x0, x_str] = model([],[],[],'lincompile');
```

Next, terminate the compilation with the following command.

```
model([],[],[],'term');
```

The output argument, `x_str`, which is a cell array of the states in the Simulink model, contains the new state ordering. When

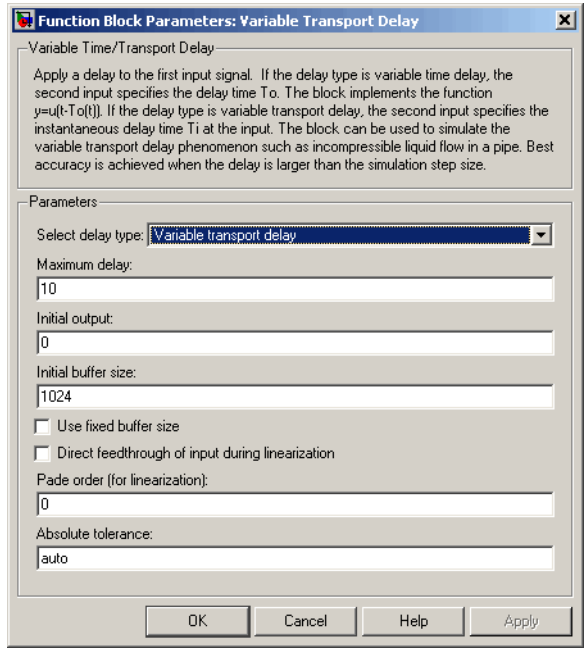
passing a vector of states as input to the `linmod`, `dlinmod`, or `trim` functions, the state vector must use this new state ordering.

Pade order (for linearization)

The order of the Pade approximation for linearization routines. The default value is `0`, which results in a unity gain with no dynamic states. Setting the order to a positive integer `n` adds `n` states to your model, but results in a more accurate linear model of the transport delay.

Variable Transport Delay Parameters and Dialog Box

The block's dialog box in Variable Transport Delay mode appears as follows.



This mode adds the following parameter.

Absolute tolerance

Absolute tolerance used to solve the block's states. You can enter `auto` or a numeric value. If you enter `auto`, Simulink determines the absolute tolerance (see "Absolute tolerance"). If you enter a numeric value, Simulink uses the specified value to solve the block's states. Note that a numeric value overrides the setting for the absolute tolerance in the **Configuration Parameters** dialog box.

Characteristics

Direct Feedthrough	Yes, of the time delay (second) input
Sample Time	Continuous
Scalar Expansion	Yes, of input and all parameters except **Buffer size**
Dimensionalized	Yes
Zero Crossing	No

Weighted Moving Average

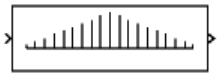
Purpose

Implement a weighted moving average

Library

Discrete

Description



The Weighted Moving Average block samples and holds the N most recent inputs, multiplies each input by a specified value (given by the **Weights** parameter), and stacks them in a vector. This block supports both single-input/single-output (SISO) and single-input/multi-output (SIMO) modes.

For the SISO mode, the **Weights** parameter is specified as a row vector. For the SIMO mode, the weights are specified as a matrix where each row corresponds to a separate output.

The **Initial condition** parameter provides the initial values for all times preceding the start time. You specify the time interval between samples with the **Sample time** parameter.

You can choose whether or not to specify the data type and scaling of the weights in the dialog with the **Gain data type and scaling** parameter. If you select `Specify via dialog` for this parameter, the **Parameter data type**, **Parameter scaling**, and **Parameter scaling mode** parameters become visible.

You can specify the scaling for the weights with the **Parameter scaling** and **Parameter scaling mode** parameters. If **Parameter data type** is a generalized fixed-point number such as `sfix(16)`, the **Parameter scaling mode** list provides you with these scaling modes:

- `Use Specified Scaling`–This mode uses the [Slope Bias] or binary point-only scaling specified by the **Parameter scaling** parameter (for example, `2^-10`).
- `Best Precision: Element-wise`–This mode produces binary points such that the precision is maximized for each element of the **Weights** parameter.
- `Best Precision: Row-wise`–This mode produces a common binary point for each element of the **Weights** row based on the best precision for the largest value of that row.

- `Best Precision: Column-wise`–This mode produces a common binary point for each element of the **Weights** column based on the best precision for the largest value of that column.
- `Best Precision: Matrix-wise`–This mode produces a common binary point for each element of the **Weights** matrix based on the best precision for the largest value of the matrix.

If the weights are specified as a row vector, then scaling element-wise and column-wise produce the same result, while scaling matrix-wise and row-wise produce the same result.

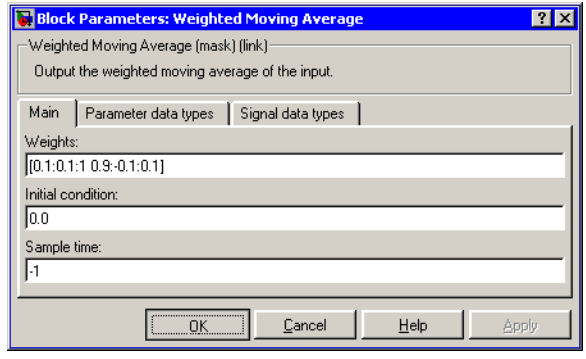
The Weighted Moving Average block first multiplies its inputs by the **Weights** parameter, converts those results to the output data type using the specified rounding and overflow modes, and then carries out the summation.

Data Type Support

The Weighted Moving Average block supports all data types supported by Simulink, including fixed-point data types.

Parameters and Dialog Box

The **Main** pane of the Weighted Moving Average block dialog appears as follows:



Weights

Specify the weights of the moving average; one row per output. The **Weights** parameter is converted from doubles to the specified data type offline using round-to-nearest and saturation.

Initial condition

Specify the initial values for all times preceding the start time. The **Initial condition** parameter is converted from doubles to the input data type offline using round-to-nearest and saturation.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "Specifying Sample Time" in

Purpose

Display an X-Y plot of signals using a MATLAB figure window

Library

Sinks

Description

The XY Graph block displays an X-Y plot of its inputs in a MATLAB figure window.

The block has two scalar inputs. The block plots data in the first input (the *x*

Model Construction Commands — Alphabetical List

addterms
add_block
add_line
add_param
attachConfigSet
bdclose
bdroot
close_system
delete_block
delete_line
delete_param
find_mdlrefs
find_system
gcb
gcbh
gcs
getActiveConfigSet
getCallbackAnnotation
getConfigSet
getConfigSets
get_param
libinfo
load_system
modeladvisor
new_system
open_system
replace_block
save_system
setActiveConfigSet
set_param
signalbuilder
simulink
Simulink.Bus.cellToObject
Simulink.Bus.createObject
Simulink.Bus.save

Simulink.SubSystem.convertToModelReference
slCharacterEncoding
sldiscmdl
slmdldiscui
slreplace_mux
slupdate
view_mdlrefs

add_block

Purpose

Add a block to a Simulink system

Syntax

```
add_block('src', 'dest')
add_block('src', 'dest', 'param1', value1, ...)
add_block('src', 'dest', 'MakeNameUnique', 'on', 'param1', value1,
...)
add_block('src_inport', 'dest_inport', 'copyoption', 'duplicate',
'param1', value1,...)
```

Description

`add_block('src', 'dest')` copies the block with the full pathname *`'src'`* to a new block with the full pathname *`'dest'`*. The block parameters of the new block are identical to those of the original. You can use `'built-in/`*`blocktype`*`'` as a source block path for Simulink built-in blocks (blocks available in Simulink block libraries that are not masked blocks), where *`blocktype`* is the built-in block's type, i.e., the value of its BlockType parameter (see "Common Block Parameters" on page 10-41).

`add_block('src', 'dest', 'param1', value1, ...)` creates a copy as above, in which the named parameters have the specified values. Any additional arguments must occur in parameter/value pairs.

`add_block('src', 'dest', 'MakeNameUnique', 'on', 'parameter1', value1,...)` creates a copy of src. If a block having the full pathname *`'dest'`* already exists, the command creates a unique name for the new block based on *`'dest'`*.

`add_block('src_inport', 'dest_inport', 'copyoption', 'duplicate', 'param1', value1,...)` applies only to Inport blocks. It creates a copy with the same port number as the *`'src_inport'`* block.

Before you add a block, you need to first open the library that contains the block with the `load_system` (library opens invisibly) or `open_system` (library opens visibly) command.

Examples

This command copies the Scope block from the Sinks subsystem of the `simulink` system to a block named `Scope1` in the `timing` subsystem of the `engine` system.

```
add_block('simulink/Sinks/Scope', 'engine/timing/Scope1')
```

This command creates a new subsystem named `controller` in the `F14` system.

```
add_block('built-in/SubSystem', 'F14/controller')
```

This command copies the built-in Gain block to a block named `Volume` in the `mymodel` system and assigns the `Gain` parameter a value of 4.

```
add_block('built-in/Gain', 'mymodel/Volume', 'Gain', '4')
```

The following command

```
block = add_block('vdp/Mu', 'vdp/Mu', 'MakeNameUnique', 'on')
```

copies the block named `Mu` in `vdp` and create a copy. Since `Mu` already exists, the command names the new block `Mu1`.

See Also

`delete_block`, `set_param`

add_line

Purpose

Add a line to a Simulink system

Syntax

```
h = add_line('sys','oport','iport')
h = add_line('sys','oport','iport', 'autorouting','on')
h = add_line('sys', points)
```

Description

The `add_line` command adds a line to the specified system and returns a handle to the new line. You can define the line in two ways:

- By naming the block ports that are to be connected by the line
- By specifying the location of the points that define the line segments

`add_line('sys', 'oport', 'iport')` adds a straight line to a system from the specified block output port `'oport'` to the specified block input port `'iport'`. `'oport'` and `'iport'` are strings consisting of a block name and a port identifier in the form `'block/port'`. Most block ports are identified by numbering the ports from top to bottom or from left to right, such as `'Gain/1'` or `'Sum/2'`. Enable, Trigger, State, and Action ports are identified by name, such as `'subsystem_name/Enable'`, `'subsystem_name/Trigger'`, `'Integrator/State'`, or `if_action_subsystem_name/Ifaction'`.

`add_line('sys','oport','iport', 'autorouting','on')` works like `add_line('sys','oport','iport')` except that it routes the line around intervening blocks. The default value for autorouting is `'off'`.

`add_line(system, points)` adds a segmented line to a system. Each row of the `points` array specifies the *x* and *y* coordinates of a point on a line segment. The origin is the top-left corner of the window. The signal flows from the point defined in the first row to the point defined in the last row. If the start of the new line is close to the output of an existing block or line, a connection is made. Likewise, if the end of the line is close to an existing input, a connection is made.

Examples

This command adds a line to the `mymodel` system connecting the output of the Sine Wave block to the first input of the Mux block.

```
add_line('mymodel','Sine Wave/1','Mux/1')
```

This command adds a line to the `mymodel` system extending from (20,55) to (40,10) to (60,60).

```
add_line('mymodel',[20 55; 40 10; 60 60])
```

See Also

`delete_line`

add_param

Purpose

Add a parameter to a Simulink system

Syntax

```
add_param('sys','parameter1',value1,'parameter2',value2,...)
```

Description

The `add_param` command adds the specified parameters to the specified system and initializes the parameters to the specified values. Case is ignored for parameter names. Value strings are case sensitive. The value of the parameter must be a string. Once the parameter is added to a system, `set_param` and `get_param` can be used on the new parameters as if they were standard Simulink parameters.

Examples

This command

```
add_param('vdp','Param1','Value1','Param2','Value2')
```

adds the parameters `Param1` and `Param2` with values `'Value1'` and `'Value2'` to the `vdp` system.

See Also

`delete_param`, `get_param`, `set_param`

Purpose

Add terminators to unconnected ports in a model

Syntax

```
addterms('sys')
```

Description

`addterms('sys')` adds Terminator and Ground blocks to the unconnected ports in the Simulink block diagram `sys`.

See Also

`slupdate`

attachConfigSet

Purpose

Associates a configuration set with a model

Syntax

```
attachConfigSet('model', configSet)
```

Description

`attachConfigSet('model')` associates the `Simulink.ConfigSet` object specified by `configSet` with `'model'`, where `'model'` is the name of an open model.

Note You cannot attach a configuration set to a model if the configuration set is already attached to another model or has the same name as another configuration set attached to the same model.

Examples

The following example creates a copy of the current model's active configuration set and attaches it to the model as an alternate configuration geared to model development.

```
activeConfig = getActiveConfigSet(gcs);
develConfig = activeConfig.copy;
develConfig.Name = 'develConfig';
attachConfigSet(gcs, develConfig);
```

See Also

`getActiveConfigSet`, `getConfigSet`

Purpose

Close any or all Simulink system windows unconditionally

Syntax

```
bdclose
bdclose('sys')
bdclose('all')
```

Description

`bdclose` with no arguments closes the current system window unconditionally and without confirmation. Any changes made to the system since it was last saved are lost.

`bdclose('sys')` closes the specified system window.

`bdclose('all')` closes all system windows.

Examples

This command closes the `vdp` system.

```
bdclose('vdp')
```

See Also

`close_system`, `new_system`, `open_system`, `save_system`

bdroot

Purpose

Return the name of the top-level Simulink system

Syntax

```
bdroot
bdroot('obj')
```

Description

`bdroot` with no arguments returns the top-level system name.

`bdroot('obj')`, where `'obj'` is a system or block pathname, returns the name of the top-level system containing the specified object name.

Examples

This command returns the name of the top-level system that contains the current block.

```
bdroot(gcb)
```

See Also

`find_system, gcb`

Purpose

Close a Simulink system window or a block dialog box

Syntax

```
close_system
close_system('sys')
close_system('sys', saveflag)
close_system('sys', 'newname')
close_system('sys', 'newname','ErrorIfShadowed')
close_system('blk')
```

Description

`close_system` with no arguments closes the current system or subsystem window. If the current system is the top-level system and it has been modified, `close_system` asks if the changed system should be saved to a file before removing the system from memory. The current system is defined in the description of the `gcs` command.

`close_system('sys')` closes the specified system or subsystem window. This command displays an error if '*sys*' is a MATLAB keyword, `'simulink'`, or more than 63 characters long.

`close_system('sys', saveflag)` closes the specified top-level system window and removes it from memory. If *saveflag* is 0, the system is not saved.

`close_system('sys', 'newname')` saves the specified top-level system to a file with the specified new name, then closes the system.

`close_system('sys','newname','ErrorIfShadowed')` saves the specified top-level system to a file with the specified new name. This command generates an error if the specified model name already exists on the MATLAB path or workspace.

`close_system('blk')`, where *'blk'* is a full block pathname, closes the dialog box associated with the specified block or calls the block's `CloseFcn` callback parameter if one is defined. Any additional arguments are ignored.

Examples

This command closes the current system.

```
close_system
```

This command closes the vdp system.

```
close_system('vdp')
```

This command saves the engine system with its current name, then closes it.

```
close_system('engine', 1)
```

This command saves the mymdl12 system under the new name testsys, then closes it.

```
close_system('mymdl12', 'testsys')
```

This command tries to save the vdp system to a file with the name 'max', but returns an error because 'max' is the name of an existing MATLAB function.

```
close_system('vdp','max','ErrorIfShadowed')
```

This command closes the dialog box of the Unit Delay block in the Combustion subsystem of the engine system.

```
close_system('engine/Combustion/Unit Delay')
```

See Also

bdclose, gcs, new_system, open_system, save_system

Purpose

Delete a block from a Simulink system

Syntax

```
delete_block('blk')
```

Description

`delete_block('blk')`, where `'blk'` is a full block pathname, deletes the specified block from a system.

Examples

This command removes the Out1 block from the `vdp` system.

```
delete_block('vdp/Out1')
```

See Also

`add_block`

delete_line

Purpose

Delete a line from a Simulink system

Syntax

```
delete_line('sys', 'oport', 'iport')
delete_line('handle')
```

Description

`delete_line('sys', 'oport', 'iport')` deletes the line extending from the specified block output port `'oport'` to the specified block input port `'iport'`. `'oport'` and `'iport'` are strings consisting of a block name and a port identifier in the form `'block/port'`. Most block ports are identified by numbering the ports from top to bottom or from left to right, such as `'Gain/1'` or `'Sum/2'`. Enable, Trigger, and State ports are identified by name, such as `'subsystem_name/Enable'`, `'subsystem_name/Trigger'`, `'Integrator/State'`, or `if_action_subsystem_name/Ifaction'`.

`delete_line('sys', [x y])` deletes one of the lines in the system that contains the specified point (x,y), if any such line exists.

`delete_line('handle')` deletes the line specified by the handle, `'handle'`.

Examples

This command removes the line from the `mymodel` system connecting the Sum block to the second input of the Mux block.

```
delete_line('mymodel','Sum/1','Mux/2')
```

See Also

`add_line`

Purpose

Delete a system parameter added via the `add_param` command

Syntax

```
delete_param('sys','parameter1','parameter2',...)
```

Description

This command deletes parameters that were added to the system using the `add_param` command. The command displays an error message if a specified parameter was not added with the `add_param` command.

Examples

The following example

```
delete_param('vdp','Param1')
add_param('vdp','Param1','Value1','Param2','Value2')
```

adds the parameters `Param1` and `Param2` to the `vdp` system, then deletes `Param1` from the system.

See Also

`add_param`

find_mdlrefs

Purpose

Find the Model blocks in a model and the models that the Model blocks reference

Syntax

```
[refMdls, mdlBlks] = find_mdlrefs('modelName')
[refMdls, mdlBlks] = find_mdlrefs('modelName', true)
[refMdls, mdlBlks] = find_mdlrefs('modelName', false)
```

Description

`[refMdls, mdlBlks] = find_mdlrefs('modelName')` or `find_mdlrefs('modelName', true)` finds all Model blocks contained by and models referenced by `'modelName'` directly or indirectly. The commands output arguments are

- `refMdls`

  List of models. The last element in the list is `'modelName'`. The other elements are the names of models referenced by `'modelName'`.

- `mdlBlks`

  Names of Model blocks contained by `'modelName'` and the models that it references directly or indirectly.

`[refMdls, mdlBlks] = find_mdlrefs(modelName, false)` finds only the Model blocks and models directly referenced by `'modelName'`.

Examples

Open the `sldemo_mdlref_basic` demo. Then execute

```
>> [r, b] = find_mdlrefs('sldemo_mdlref_basic')

r =
    'sldemo_mdlref_counter'
    'sldemo_mdlref_basic'

b =
    'sldemo_mdlref_basic/CounterA'
    'sldemo_mdlref_basic/CounterB'
    'sldemo_mdlref_basic/CounterC'
```

See Also

`view_mdlrefs`

find_system

Purpose

Find systems, blocks, lines, ports, and annotations

Syntax

```
find_system(sys, 'c1', cv1, 'c2', cv2,...'p1',
v1, 'p2', v2,...)
```

Description

`find_system(sys, 'c1', cv1, 'c2', cv2,...'p1', v1, 'p2', v2,...)` searches the systems or subsystems specified by *sys*, using the constraints specified by *c1*, *c2*, etc., and returns handles or paths to the objects whose parameters, *p1*, *p2*, etc., have the values, *v1*, *v2*, etc. *sys* can be a pathname (or cell array of pathnames), a handle (or vector of handles), or omitted. If you specify `'BlockDialogParams'` as the parameter name , `find_system` searches for all blocks that have a parameter that has the specified value and appears in the block's dialog box.

Note All the search constraints must precede all the property-value pairs in the argument list.

If `sys` is a pathname or cell array of pathnames, `find_system` returns a cell array of pathnames of the objects it finds. If `sys` is a handle or a vector of handles, `find_system` returns a vector of handles to the objects that it finds. If `sys` is omitted, `find_system` searches all open systems and returns a cell array of pathnames.

Case is ignored for parameter names. Value strings are case sensitive by default (see the `'CaseSensitive'` search constraint for more information). Any parameters that correspond to dialog box entries have string values. See Chapter 10, "Model and Block Parameters" for a list of model and block parameters.

You can specify any of the following search constraints.

Name	Value Type	Description
'SearchDepth'	scalar	Restricts the search depth to the specified level (0 for open systems only, 1 for blocks and subsystems of the top-level system, 2 for the top-level system and its children, etc.). The default is all levels.
'LookUnderMasks'	'none'	Search skips masked blocks.
	{'graphical'}	Search includes masked blocks that have no workspaces and no dialogs. This is the default.
	'functional'	Search includes masked blocks that do not have dialogs.
	'all'	Search includes all masked blocks.
'FollowLinks'	'on'| {'off'}	If 'on', search follows links into library blocks. The default is 'off'.
'FindAll'	'on'| {'off'}	If 'on', search extends to lines, ports, and annotations within systems. The default is 'off'. Note that find_system returns a vector of handles when this option is 'on', regardless of the array type of sys.
'CaseSensitive'	{'on'}| 'off'	If 'on', search considers case when matching search strings. The default is 'on'.
'RegExp'	'on'| {'off'}	If 'on', search treats search expressions as regular expressions. The default is'off'.

The table encloses default constraint values in brackets. If a `'constraint'` is omitted, `find_system` uses the default constraint value.

By default, `find_system` attempts to load any partially loaded models. When a `PreLoadFcn` callback invokes `find_system`, `find_system` tries to load the calling model, causing recursive load warnings. To prevent this warning, disable the model loading property of `find_system`. Turn off the `LoadFullyIfNeeded` property, as follows:

```
find_system(gcs,'LoadFullyIfNeeded','off','PropertyName','PropertyValue')
```

Examples

This command returns a cell array containing the names of all open systems and blocks.

```
find_system
```

This command returns the names of all open block diagrams.

```
open_bd = find_system('type', 'block_diagram')
```

This command returns the names of all Goto blocks that are children of the `Unlocked` subsystem in the `clutch` system.

```
find_system('clutch/
Unlocked','SearchDepth',1,'BlockType','Goto')
```

These commands return the names of all Gain blocks in the `vdp` system having a `Gain` parameter value of 1.

```
gb = find_system('vdp', 'BlockType', 'Gain')
find_system(gb, 'Gain', '1')
```

The preceding commands are equivalent to this command:

```
find_system('vdp', 'BlockType', 'Gain', 'Gain', '1')
```

These commands obtain the handles of all lines and annotations in the vdp system.

```
sys = get_param('vdp', 'Handle');
l = find_system(sys, 'FindAll', 'on', 'type', 'line');
a = find_system(sys, 'FindAll', 'on', 'type', 'annotation');
```

Searching with Regular Expressions

If you specify the 'RegExp' constraint as 'on', find_system treats search value strings as regular expressions. A regular expression is a string of characters in which some characters have special pattern-matching significance. For example, a period (.) in a regular expression matches not only itself but any other character.

Regular expressions greatly expand the types of searches you can perform with find_system. For example, regular expressions allow you to do partial-word searches. You can search for all objects that have a specified parameter that contains or begins or ends with a specified string of characters.

To use regular expressions effectively, you need to learn the meanings of the special characters that regular expressions can contain. The following table lists the special characters supported by find_subystem and explains their usage.

Expression	Usage
.	Matches any character. For example, the string 'a.' matches 'aa', 'ab', 'ac', etc.
*	Matches zero or more of preceding character. For example, 'ab*' matches 'a', 'ab', 'abb', etc. The expression '.*' matches any string, including the empty string.
+	Matches one or more of preceding character. For example, 'ab+' matches 'ab', 'abb', etc.
^	Matches start of string. For example, '^a.*' matches any string that starts with 'a'.

Expression	Usage
`$`	Matches end of string. For example, `'.*a$'` matches any string that ends with `'a'`.
`\`	Causes the next character to be treated as an ordinary character. This escape character lets regular expressions match expressions that contain special characters. For example, the search string `'\\'` matches any string containing a `\` character.
`[]`	Matches any one of a specified set of characters. For example, `'f[oa]r'` matches `'for'` and `'far'`. Some characters have special meaning within brackets. A hyphen (`-`) indicates a range of characters to match. For example, `'[a-zA-Z1-9]'` matches any alphanumeric character. A circumflex (`^`) indicates characters that should not produce a match. For example, `'f[^i]r'` matches `'far'` and `'for'` but not `'fir'`.
`\w`	Matches a word character. (This is a shorthand expression for `[a-z_A-Z0-9]`.) For example, `'^\w'` matches `'mu'` but not `'&mu'`.
`\d`	Matches any digit (shorthand for `[0-9]`). For example, `'\d+'` matches any integer.
`\D`	Matches any nondigit (shorthand for `[^0-9]`).
`\s`	Matches a white space (shorthand for `[ \t\r\n\f]`).
`\S`	Matches a non white-space (shorthand for `[^ \t\r\n\f]`).
`\<WORD\>`	Matches `WORD` exactly, where `WORD` is a string of characters separated by white space from other words. For example, `'\<to\>'` matches `'to'` but not `'today'`.

To use regular expressions to search Simulink systems, specify the `'regexp'` search constraint as `'on'` in a `find_system` command and use a regular expression anywhere you would use an ordinary search value string.

For example, the following command finds all the inport and outport blocks in the `clutch` model demo provided with Simulink.

```
find_system('clutch', 'regexp', 'on', 'blocktype', 'port')
```

See Also

`get_param`, `set_param`

gcb

Purpose

Get the pathname of the current block

Syntax

```
gcb
gcb('sys')
```

Description

`gcb` returns the full block pathname of the current block in the current system.

`gcb('sys')` returns the full block pathname of the current block in the specified system.

The current block is one of these:

- During editing, the current block is the block most recently clicked.
- During simulation of a system that contains S-Function blocks, the current block is the S-Function block currently executing its corresponding MATLAB function.
- During callbacks, the current block is the block whose callback routine is being executed.
- During evaluation of the `MaskInitialization` string, the current block is the block whose mask is being evaluated.

Examples

This command returns the path of the most recently selected block.

```
gcb
ans =
   clutch/Locked/Inertia
```

This command gets the value of the `Gain` parameter of the current block.

```
get_param(gcb,'Gain')
ans =
   1/(Iv+Ie)
```

See Also

`gcbh`, `gcs`

Purpose

Get the handle of the current block

Syntax

```
gcbh
```

Description

`gcbh` returns the handle of the current block in the current system.

You can use this command to identify or address blocks that have no parent system. The command should be most useful to blockset authors.

Examples

This command returns the handle of the most recently selected block.

```
gcbh

ans =

   281.0001
```

See Also

`gcb`

gcs

Purpose

Get the pathname of the current system

Syntax

```
gcs
```

Description

`gcs` returns the full pathname of the current system.

The current system is one of these:

- During editing, the current system is the system or subsystem most recently clicked.
- During simulation of a system that contains S-Function blocks, the current system is the system or subsystem containing the S-Function block that is currently being evaluated.
- During callbacks, the current system is the system containing any block whose callback routine is being executed.
- During evaluation of the `MaskInitialization` string, the current system is the system containing the block whose mask is being evaluated.

The current system is always the current model or a subsystem of the current model. Use `bdroot` to get the current model.

Examples

This example returns the path of the system that contains the most recently selected block.

```
gcs
ans =
   clutch/Locked
```

See Also

`bdroot` , `gcb`

Purpose

Get a model's active configuration set.

Syntax

```
getActiveConfigSet('model')
```

Description

`getActiveConfigSet('model')` returns a `Simulink.ConfigSet` object representing the active configuration set of `'model'`, where `'model'` is the name of an open model.

Examples

The following command

```
cs = getActiveConfigSet(gcs);
```

returns the active configuration set of the currently selected model.

See Also

`attachConfigSet`, `setActiveConfigSet`

getCallbackAnnotation

Purpose

Get information about an annotation

Syntax

```
getCallbackAnnotation
```

Description

`getCallbackAnnotation` is intended to be invoked by annotation callback functions. If it is invoked from an annotation callback function, it returns an instance of `Simulink.Annotation` class that represents the annotation associated with the callback function. The callback function can then use the instance to get and set the annotation's properties, such as its text, font and color. If this function is not invoked from an annotation callback function, it returns nothing, i.e., `[]`.

Purpose

Get one of a model's configuration sets

Syntax

```
getConfigSet('model', configSetName)
```

Description

`getConfigSets('model', configSetName)` returns a `Simulink.ConfigSet` object representing the configuration set named `configSetName` whose owner is `'model'`, where model is a string specifying the name of a model.

Note Use `getConfigSets` to get the names of the configuration sets owned by a model.

Examples

The following command gets the configuration set of the currently selected model whose name is `'DevelopmentConfiguration'`.

```
hCs = getConfigSet(gcs, 'DevelomentConfiguration');
```

See Also

`attachConfigSet`, `getActiveConfigSet`, `getConfigSets`, `setActiveConfigSet`

getConfigSets

Purpose

Get the names of a model's configuration sets

Syntax

```
getConfigSets('model')
```

Description

`getConfigSets('model')` returns a cell array of strings specifying the names of the configuration sets owned by `'model'`, where `'model'` is the name of a model.

Examples

The following command displays the names of the configuration sets owned by the current selected model at the MATLAB command line.

```
getConfigSets(gcs)
```

See Also

`attachConfigSet`, `setActiveConfigSet`

Purpose

Get system and block parameter values

Syntax

```
get_param('obj', 'parameter')
get_param( { objects }, 'parameter')
get_param(handle, 'parameter')
get_param(0, 'parameter')
get_param('obj', 'ObjectParameters')
get_param('obj', 'DialogParameters')
```

Description

`get_param('obj', 'parameter')`, where `'obj'` is a system or block pathname, returns the value of the specified parameter. Case is ignored for parameter names.

`get_param( { objects }, 'parameter')` accepts a cell array of full path specifiers, enabling you to get the values of a parameter common to all objects specified in the cell array.

`get_param(handle, 'parameter')` returns the specified parameter of the object whose handle is `handle`.

`get_param(0, 'parameter')` returns the current value of a Simulink session parameter or the default value of a model or block parameter.

`get_param('obj', 'ObjectParameters')` returns a structure that describes `obj`'s parameters. Each field of the returned structure corresponds to a particular parameter and has the parameter's name. For example, the Name field corresponds to the object's `Name` parameter. Each parameter field itself contains three fields, Name, Type, and Attributes, that specify the parameter's name (for example, `'Gain'`), data type (for example, string), and attributes (for example, read-only), respectively.

`get_param('obj', 'DialogParameters')` returns a cell array containing the names of the dialog parameters of the specified block.

Chapter 10, "Model and Block Parameters" contains lists of model and block parameters.

Examples

This command returns the value of the `Gain` parameter for the Inertia block in the `Requisite Friction` subsystem of the `clutch` system.

```
get_param('clutch/Requisite Friction/Inertia','Gain')
ans =
   1/(Iv+Ie)
```

These commands display the block types of all blocks in the `mx + b` system (the current system), described in "Masked Subsystem Example" in "Using Simulink®".

```
blks = find_system(gcs, 'Type', 'block');
listblks = get_param(blks, 'BlockType')

listblks =

    'SubSystem'
    'Inport'
    'Constant'
    'Gain'
    'Sum'
    'Outport'
```

This command returns the name of the currently selected block.

```
get_param(gcb, 'Name')
```

The following commands get the attributes of the currently selected block's `Name` parameter.

```
p = get_param(gcb, 'ObjectParameters');
a = p.Name.Attributes

ans =
    'read-write'    'always-save'
```

The following command gets the dialog parameters of a Sine Wave block.

```
p = get_param('untitled/Sine Wave', 'DialogParameters')
p =
    'Amplitude'
    'Frequency'
    'Phase'
    'SampleTime'
```

See Also

`find_system`, `set_param`

libinfo

Purpose

Get information about the library blocks referenced by a model

Syntax

```
libdata = libinfo('sys')
```

Description

`libdata = libinfo('sys')` returns information about library blocks referenced by `sys` and all of the systems underneath it. The command returns an array of structures that describes each library block referenced by the model. Each structure has the following fields:

- `Block`

  Path of the link to the library block.

- `Library`

  Name of the library containing the referenced block.

- `ReferenceBlock`

  Path of the library block.

- `LinkStatus`

  Value of the `LinkStatus` parameter for the link to the library block.

This command also accepts search constraints as additional arguments. For instance:

```
libdata=libinfo(Sys,'FollowLinks','off')
```

See `find_system` for more information.

Purpose

Invisibly load a Simulink model

Syntax

```
load_system('sys')
```

Description

`load_system('sys')` loads `'sys'`, where `sys` is the name of a Simulink model, into memory without making its model window visible.

Examples

The command

```
load_system('vdp')
```

loads the `vdp` sample model into memory.

See Also

`close_system`, `open_system`

modeladvisor

Purpose

Open the Model Advisor

Syntax

```
modeladvisor(model)
```

Description

`modeladvisor(model)` opens the Model Advisor (see the "Model Advisor" in "Using Simulink®") on the model or subsystem specified by model, where `model` is a path or handle to the `model` or subsystem. If the specified model or subsystem is not open, this command opens it.

Examples

The command

```
modeladvisor('vdp')
```

opens the Model Advisor on the `vdp` demo model.

The command

```
modeladvisor('f14/Aircraft Dynamics Model')
```

opens the Model Advisor on the Aircraft Dynamics Model subsystem of the `f14` demo model. The command

```
modeladvisor(gcs)
```

opens the Model Advisor on the currently selected subsystem.

The command

```
modeladvisor(bdroot)
```

opens the Model Advisor on the currently selected model.

Purpose

Create an empty Simulink system

Syntax

```
new_system('sys')
new_system('sys', 'Model')
new_system('sys', 'Model', 'subsystem_path')
new_system('sys', 'Model', 'ErrorIfShadowed')
new_system('sys', 'Library')
```

Description

`new_system('sys')` or `new_system('sys', 'Model')` creates an empty system where *'sys'* is the name of the new system. This command displays an error if '*sys*' is a MATLAB keyword, `'simulink'`, or more than 63 characters long.

`new_system('sys', 'Model', 'subsystem_path')` creates a system from a subsystem where `'subsystem_path'` is the full path of the subsystem. The model that contains the subsystem must be open when this command is executed.

`new_system('sys', 'Model', 'ErrorIfShadowed')` creates an empty system having the specified name. This command generates an error if another model, M-file, or variable of the same name exists on the MATLAB path or workspace.

`new_system('sys', 'Library')` creates an empty library.

Note The `new_system` command does not open the window of the system or library that it creates.

See Chapter 10, "Model and Block Parameters" for a list of the default parameter values for the new system.

Examples

This command creates a new system named `'mysys'`.

```
new_system('mysys')
```

The command

```
new_system('mysys','Library')
```

creates, but does not open, a new library named `'sys'`.

The command

```
new_system('vdp','Model','ErrorIfShadowed')
```

returns an error because `'vdp'` is the name of a model on the MATLAB path.

The commands

```
load_system('f14')
new_system('mycontroller','Model','f14/Controller')
```

create a new model named `mycontroller` that has the same contents as does the subsystem named `Controller` in the `f14` demo model.

See Also

`close_system`, `open_system`, `save_system`

Purpose

Open a Simulink system window or a block dialog box

Syntax

```
open_system('sys')
open_system('blk')
open_system('blk', 'force')
open_system('blk', 'parameter')
open_system('blk', 'mask')
open_system('blk', 'OpenFcn')
open_system('sys', 'replace')
open_system('sys', 'reuse')
```

Description

`open_system('sys')` opens the specified system or subsystem window, where `'sys'` is the name of a model on the MATLAB path, the fully qualified pathname of a model, or the relative pathname of a subsystem of an already open system (for example, `engine/Combustion`). On UNIX, the fully qualified pathname of a model can start with a tilde (~), signifying your home directory.

`open_system('blk')`, where `'blk'` is a full block pathname, opens the dialog box associated with the specified block. If the block's `OpenFcn` callback parameter is defined, the routine is evaluated.

`open_system('blk', 'force')`, where `'blk'` is a full pathname or a masked system, looks under the mask of the specified system. This command is equivalent to using the **Look Under Mask** menu item.

`open_system('blk', 'parameter')` opens this block's parameter dialog box.

`open_system('sys', 'mask')` opens this block's mask.

`open_system('blk', 'OpenFcn')` runs this block's open function.

`open_system('sys', 'replace')` the window of the parent system with the window of the subsystem opened by this command.

`open_system('sys', 'reuse')` uses the parent system's window to display the subsystem opened by this command.

Examples

This command opens the `controller` system in its default screen location.

```
open_system('controller')
```

This command opens the block dialog box for the Gain block in the `controller` system.

```
open_system('controller/Gain')
```

Suppose that `mymodel` contains a masked subsystem, `A`, and a block, `B`, whose `OpenFcn` contains the following lines:

```
open_system('mymodel/B', 'parameter');
open_system('mymodel/A', 'mask');
```

Then opening block `B` causes both the parameter dialog box for B and the mask dialog box for `A` to appear.

See Also

`close_system`, `load_system`, `new_system`, `save_system`

Purpose

Replace blocks in a Simulink model

Syntax

```
replace_block('sys', 'blk1', 'blk2', 'noprompt')
replace_block('sys', 'Parameter', 'value', 'blk', ...)
```

Description

`replace_block('sys', 'blk1', 'blk2')` replaces all blocks in `'sys'` having the block or mask type `'blk1'` with `'blk2'`. If `'blk2'` is a Simulink built-in block, only the block name is necessary. If `'blk'` is in another system, its full block pathname is required. If `'noprompt'` is omitted, Simulink displays a dialog box that asks you to select matching blocks before making the replacement. Specifying the `'noprompt'` argument suppresses the dialog box from being displayed. If a return variable is specified, the paths of the replaced blocks are stored in that variable.

`replace_block('sys', 'Parameter', 'value', ..., 'blk')` replaces all blocks in `'sys'` having the specified values for the specified parameters with `'blk'`. You can specify any number of parameter name/value pairs.

Note Because it may be difficult to undo the changes this command makes, it is a good idea to save your system first.

Examples

This command replaces all Gain blocks in the `f14` system with Integrator blocks and stores the paths of the replaced blocks in `RepNames`. Simulink lists the matching blocks in a dialog box before making the replacement.

```
RepNames = replace_block('f14','Gain','Integrator')
```

This command replaces all blocks in the `Unlocked` subsystem in the `clutch` system having a `Gain` of `'bv'` with the Integrator block. Simulink displays a dialog box listing the matching blocks before making the replacement.

```
replace_block('clutch/Unlocked','Gain','bv','Integrator')
```

This command replaces the Gain blocks in the `f14` system with Integrator blocks but does not display the dialog box.

```
replace_block('f14','Gain','Integrator','noprompt')
```

See Also

`find_system`, `set_param`

Purpose

Save a Simulink system

Syntax

```
save_system
save_system('sys')
save_system('sys', 'newname')
save_system('sys', 'newname', 'BreakLinks')
save_system('sys', 'newname', 'SaveModelWorkspace')
save_system('sys', 'newname', 'BreakLinks', 'SaveModelWorkspace')
save_system('sys', 'newname', 'ErrorIfShadowed')
save_system('sys', 'newname', '', 'version')
save_system('sys', 'newname', 'BreakLinks', 'version')
```

Description

`save_system` saves the current top-level system to a file with its current name.

`save_system('sys')` saves the specified top-level system to a file with its current name. The system must be open.

`save_system('sys', 'newname')` saves the specified top-level system to a file with the specified new name. The system to be saved must be open. The new name can be a fil name, in which case Simulink saves the system in the working directory, or a fully qualified pathname. On UNIX, the fully qualified pathname can start with a tilde (~), signifying your home directory. This command displays an error if you enter any of the following as the new model name:

A MATLAB keyword
`'simulink'`
More than 63 characters

`save_system('sys', 'newname', 'BreakLinks')` saves the specified top-level system to a file with the specified new name, replacing links to library blocks with copies of the library blocks in the saved file.

`save_system('sys', 'newname', 'SaveModelWorkspace')` saves the specified top-level system to a file with the specified new name. If the model workspace `DataSource` is a MAT-file, this command also saves

the contents of the model workspace. 'SaveModelWorkspace' is most useful when DataSource is a MAT-file.

`save_system('sys', 'newname', 'BreakLinks','SaveModelWorkspace')` saves the specified top-level system to a file with the specified new name, replacing links to library blocks with copies of the library blocks in the saved file. If the model workspace DataSource is a MAT-file, this command also saves the contents of the model workspace. 'SaveModelWorkspace' is most useful when DataSource is a MAT-file. The positions of 'BreakLinks' and 'SaveModelWorkspace' are interchangeable.

`save_system('sys', 'newname', 'ErrorIfShadowed',)` saves the specified top-level system to a file with the specified new name. This command generates an error if the specified new name already exists on the MATLAB path or workspace.

`save_system('sys', 'newname', '', 'version')` saves the specified top-level system in a form that can be loaded by a specified version of Simulink. Valid values for 'version' include 'R14', 'R13SP1', 'R13', 'R12P1', and 'R12'. If the system to be saved contains blocks not supported by the specified Simulink version, the command replaces the unsupported blocks with empty masked subsystem blocks colored yellow. As a result, the converted system may generate incorrect results.

`save_system('sys', 'newname', 'BreakLinks', 'version')` saves the specified top-level system with broken library links and in a form compatible with a specified version of Simulink.

Examples

This command saves the current system.

```
save_system
```

This command saves the vdp system.

```
save_system('vdp')
```

This command saves the vdp system to a file with the name 'myvdp'.

```
save_system('vdp', 'myvdp')
```

This command saves the vdp system to a file with the name 'myvdp' and replaces links to library blocks with copies of the library blocks in the saved file.

```
save_system('vdp','myvdp','BreakLinks')
```

This command tries to save the vdp system to a file with the name 'max', but returns an error because 'max' is the name of a MATLAB function.

```
save_system('vdp', 'max', 'ErrorIfShadowed')
```

This command saves the vdp system to Simulink Version R13SP1 with the name 'myvdp'. It does not replace links to library blocks with copies of the library blocks.

```
save_system('vdp','myvdp','','R13SP1)
```

See Also

close_system, new_system, open_system

setActiveConfigSet

Purpose

Sets a model's active configuration set

Syntax

```
setActiveConfigSet('model', 'configSetName')
```

Description

`setActiveConfigSet('model', 'configSetName')` sets the active configuration set of `model`, where `model` is the name of an open model, to *`configSetName`*, where *`configSetName`* specifies the name of one of the model's configuration sets.

Examples

The following example

```
setActiveConfigSet(gcs, 'develConfigSet');
```

makes it the active configuration set of the currently selected model.

See Also

`attachConfigSet`, `getActiveConfigSet`

Purpose

Set Simulink system and block parameters

Syntax

```
set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)
set_param(0, 'modelparm1', value1, 'modelparm2', value2, ...)
```

Description

`set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)`, where `'obj'` is a system or block path, sets the specified parameters to the specified values. Value strings are case sensitive. Case is ignored for parameter names. Any parameters that correspond to dialog box entries have string values. Model and block parameters are listed in Chapter 10, "Model and Block Parameters".

`set_param(0, 'modelparm1', value1, 'modelparm2', value2, ...)` sets the specified model parameters to default values, i.e., to values that Simulink assigns to the parameters when it creates a model. You can use this form of `set_param` in your MATLAB startup file to specify your own default values for Simulink model parameters.

You can change block parameter values in the workspace during a simulation and update the block diagram with these changes. To do this, make the changes in the command window, then make the model window the active window, then choose **Update Diagram** from the **Edit** menu.

Note Most block parameter values must be specified as strings. Two exceptions are the `Position` and `UserData` parameters, common to all blocks.

Examples

This command sets the `Solver` and `StopTime` parameters of the `vdp` system.

```
set_param('vdp', 'Solver', 'ode15s', 'StopTime', '3000')
```

This command sets the `Gain` parameter of block Mu in the `vdp` system to 1000 (stiff).

```
set_param('vdp/Mu', 'Gain', '1000')
```

s

This command sets the position of the Fcn block in the `vdp` system.

```
set_param('vdp/Fcn', 'Position', [50 100 110 120])
```

This command sets the `Zeros` and `Poles` parameters for the Zero-Pole block in the `mymodel` system.

```
set_param('mymodel/Zero-Pole','Zeros','[2 4]','Poles','[1 2 3]')
```

This command sets the `Gain` parameter for a block in a masked subsystem. The variable `k` is associated with the `Gain` parameter.

```
set_param('mymodel/Subsystem', 'k', '10')
```

This command sets the `OpenFcn` callback parameter of the block named `Compute` in system `mymodel`. The function `'my_open_fcn'` executes when you double-click on the Compute block (see "Using Callback Functions").

```
set_param('mymodel/Compute', 'OpenFcn', 'my_open_fcn')
```

See Also

`find_system`, `get_param`

Purpose

Create and access Signal Builder blocks

Syntax

```
[time, data] = signalbuilder(block)
[time, data, siglabels] = signalbuilder(block)
[time, data, siglabels, grouplabels] = signalbuilder(block)
block = signalbuilder([],'create', time, data, siglabels, grouplabels)
block = signalbuilder(block,'append', time, data, siglabels, grouplabels)
[time, data] = signalbuilder(block,'get', signal, group)
signalbuilder(block, 'set', signal, group, time, data)
index = signalbuilder(block, 'activegroup')
signalbuilder(block, 'activegroup', index)
```

Description

`[time, data] = signalbuilder(block)` returns the time (x coordinate) and amplitude (y coordinate) data of the Signal Builder `block`.

The output arguments, `time` and `data`, take different formats depending on the block configuration:

Configuration	Time/Data Format
1 signal, 1 group	Row vector of break points.
>1 signal, 1 group	Column cell vector where each element corresponds to a separate signal and contains a row vector of breakpoints.
1 signal, >1 group	Row cell vector where each element corresponds to a separate group and contains a row vector of breakpoints.
>1 signal, >1 group	Cell matrix where each element (i,j) corresponds to signal i and group j.

`[time, data, siglabels] = signalbuilder(block)` returns the signal labels, *`siglabels`*, in a string or a cell array of strings.

`[time, data, siglabels, grouplabels] = signalbuilder(block)` returns the group labels, *`grouplabels`*, in a string or a cell array of strings.

block = signalbuilder([],'create', *time*, *data*, *siglabels*, *grouplabels*) creates a Signal Builder block in a new Simulink model using the specified values. If *data* is a cell array and *time* is a vector, the *time* values are duplicated for each element of data. Each vector in *time* and *data* must be the same length and have at least two elements. If *time* is a cell array, all elements in a column must have the same initial and final value. Signal labels, *siglabels*, and group labels, *grouplabels*, can be omitted to use default values. The function returns the path to the new block, *block*.

block = signalbuilder(*block*,'append', *time*, *data*, *siglabels*, *grouplabels*) appends new groups to the Signal Builder block, *block*. The *time* and *data* arguments must have the same number of signals as the existing block.

Get/Set Methods for Specific Signals and Groups

[*time*, *data*] = signalbuilder(*block*,'get', *signal*, *group*) gets the time and data values for the specified signal(s) and group(s). The signal argument can be the name of a signal, a scalar index of a signal, or an array of signal indices. The group argument can be a group label, a scalar index, or an array of indices.

signalbuilder(*block*,'set', *signal*, *group*, *time*, *data*) sets the time and data values for the specified signal(s) and group(s). Use empty values of *time* and *data* to remove groups and signals.

Query and Set the Active Group

index = signalbuilder(*block*, 'activegroup') gets the index of the active group.

signalbuilder(*block*, 'activegroup', *index*) sets the active group index to *index*.

Purpose

Open the Simulink block library

Syntax

```
simulink
simulink('open')
simulink('close')
```

Description

On Microsoft Windows, `simulink` or `simulink('open')` opens the Simulink block library browser. On UNIX, the command opens the Simulink library window. `simulink('close')` closes the library window.

Simulink.Bus.cellToObject

Purpose

Convert a cell array containing bus information to bus objects

Syntax

```
Simulink.Bus.cellToObject(busCell)
```

Description

Simulink.Bus.cellToObject(busCell) creates a set of bus objects in the MATLAB base workspace from a cell array of bus information.

See Also

`Simulink.Bus.save`

Purpose

Creates bus objects for blocks

Syntax

```
busInfo = Simulink.Bus.createObject(model, blks)
busInfo = Simulink.Bus.createObject(model, blks, 'fileName')
busInfo = Simulink.Bus.createObject(model, blks, 'fileName',
 'format')
```

Description

`Simulink.Bus.createObject(`*`model`*`, `*`blks`*`, `*`'fileName'`*`, `*`'format'`*`)` creates bus objects, i.e., instances of `Simulink.Bus` class, in the MATLAB workspace for specified blocks and optionally saves the bus objects in an M-file. The function accepts the following arguments:

- *`model`* — Name or handle of a model
- *`blks`* — List of subsystem-level Inport blocks, root-level or subsystem-level Outport blocks or Bus Creator blocks in the specified model. If only one block needs to be specified, this argument can be the full pathname of the block. Otherwise, this argument can be either a cell array containing block pathnames or a vector of block handles.
- *`'fileName'`* — Name of the file in which to save the bus objects created by this function. If this argument is omitted, this function does not save the created bus objects in a file.
- *`'format'`* — Format used to store the bus objects. May be `'cell'` or `'object'` or omitted in which case `'cell'` is assumed. Use cell array format to save the objects in a compact form.

This function returns a structure array containing bus information for the specified blocks. Each element of the structure array corresponds to one of the specified blocks and contains the following fields:

- `block` — Handle of the block
- `busName` — Name of the bus object associated with the block

See Also

`Simulink.Bus.cellToObject`, `Simulink.Bus.save`

Simulink.Bus.save

Purpose

Save bus objects in an M-file

Syntax

```
Simulink.Bus.save('fileName')
Simulink.Bus.save('fileName', 'format')
Simulink.Bus.save('fileName', 'format', busNames)
```

Description

`Simulink.Bus.save('fileName', 'format', busNames)` saves bus objects, i.e., instances of `Simulink.Bus` class, residing in the MATLAB workspace in an M-file. Executing the M-file restores the objects to the workspace. This function takes the following arguments:

- *`'fileName'`* — Name of the file in which to store the bus objects
- *`'format'`* — Format used to store the bus objects. May be `'cell'` or `'object'` or omitted in which case `'cell'` is assumed. Use cell array format to save the objects in a compact form.
- *`busNames`* — A cell array containing names of bus objects to be saved. If the cell array is empty or omitted, this function saves all bus objects in the MATLAB workspace.

See also

`Simulink.Bus.cellToObject`

Purpose

Converts an atomic subsystem to a model reference

Syntax

```
[success,mdlRefBlkH] =
Simulink.SubSystem.convertToModelReference(subsys,
mdlRef, 'opt1', 'val1', 'opt2', 'val2', ...)
```

Description

`[success,mdlRefBlkH] = Simulink.SubSystem.convertToModelReference(subsys, mdlRef, 'opt1', 'val1', 'opt2', 'val2', ...)` converts an atomic subsystem to a model reference. It does this by creating a model, copying the contents of the subsystem into the model, and reconfiguring the root level `Inport` and `Outport` blocks and configuration parameters of the new model. Then, based on its input arguments, this function either replaces the subsystem block with a Model block that references the new model, or it creates another, temporary model containing a Model block that references the model derived from the subsystem block.

Note Execute

```
sldemo_mdlref_conversion
```

at the MATLAB command line for a demonstration of this command's usage.

To be converted, your model must specify the following configuration parameter settings:

- The **Inline parameters** option in the **Optimization** pane must be on.
- The **Signal resolution** option in the **Data Validity** diagnostics pane must be set to `Explicit only`.
- The **Mux blocks used to create bus signals** diagnostic in the **Connectivity** diagnostics pane must be set to `Error`.

You can use the following commands to set these parameters to the values required by this function:

```
set_param(mdlName, 'InlineParams', 'on');
set_param(mdlName, 'SignalResolutionControl', 'UseLocalSettings');
set_param(mdlName, 'StrictBusMsg', 'ErrorLevel1');
```

Note This function produces error or warning messages for models and subsystems that it cannot handle. Even if conversion is successful, you may still need to reconfigure the resulting model to meet your requirements.

This function accepts the following arguments:

- *subsys* — Full name or handle of the atomic subsystem block to be converted
- *mdlRef* — Name of the model to which the subsystem is to be converted
- *'opt1'*, *'val1'*, *'opt2'*, *'val2'*... — parameter/value pairs that specify various conversion options. This function support the following option pairs:
  - `'Replace Subsystem', [true|{false}]` — If the option value is true, this function replaces the subsystem block with a Model block that references the model created from the subsystem. If you do not specify this option or specify its value as false, this function creates and opens a model containing a Model block that references the model derived from the subsystem block.
  - `'BusSaveFormat', ['Cell' | 'Object']` — If this option is specified, the function saves the bus objects that it creates in an M-file. See `Simulink.Bus.save` for more information.
  - `'BuildTarget', ['Sim' | 'RTW']` — If you specify this option, this function generates a model reference Sim or RTW target for the new model.

- `'Force', [true|{false}]` — If this parameter is true, this function reports some errors that would halt the conversion process as warnings and continues with the conversion. This allows you to use this function to do the initial steps of conversion and then complete the conversion process yourself. If you do not specify this option or specify it as false, this function halts the conversion if an error occurs.

This function returns the following outputs:

- *`success`* — True if this function is successful; otherwise, false.
- *`mdlRefBlkH`* — Handle of the `Model` block that references the new model

See Also

`Simulink.Bus.save`

slCharacterEncoding

Purpose

Change the MATLAB character set encoding

Syntax

```
slCharacterEncoding()
slCharacterEncoding(encoding)
```

Description

This command allows you to change the current MATLAB character set encoding to be compatible with the encoding of a model that you want to open.

`slCharacterEncoding()` returns the current MATLAB character set encoding.

`slCharacterEncoding(encoding)` change the MATLAB character set encoding to the specified encoding. Valid values include:

- `'US-ASCII'`
- `'UTF-8'`
- `'Shift_JIS'`
- `'ISO-8859-1'`

To display a complete list of the names of character set encodings supported by MATLAB and the characters supported by the encodings, use the ICU Converter Explorer. The first column of the ICU Converter Explorer lists the primary names of the character sets supported by MATLAB. The remaining columns list aliases for the character sets.

The `slCharacterEncoding` command accepts the aliases as well as the primary names of character sets. To display a table listing the characters supported by a character set and the encodings for the characters, click the character set's primary name in the ICU Converter Explorer.

Note You must close all open models or libraries before changing the MATLAB character set encoding except when changing from `'US-ASCII'` to another encoding.

Purpose

Discretize a Simulink model containing continuous blocks

Syntax

```
sldiscmdl('sys',sampletime)
sldiscmdl('sys',sampletime,'method')
sldiscmdl('sys',sampletime,{options})
sldiscmdl('sys',sampletime,'method',cf)
sldiscmdl('sys',sampletime,'method',{options})
sldiscmdl('sys',sampletime,'method',cf,{options})
```

Description

`sldiscmdl('sys',sampletime)` discretizes the model specified by `'sys'` and `sampletime`. You can enter a sample time and an offset as a two-element vector for `sampletime`. The units for `sampletime` are seconds.

`sldiscmdl('sys',sampletime,'method')` discretizes the model with the transform method specified by `'method'`. Available values for `'method'` are shown below:

Value	Description
`'zoh'`	Zero-order hold on the inputs
`'foh'`	First-order hold on the inputs
`'tustin'`	Bilinear (Tustin) approximation
`'prewarp'`	Tustin approximation with frequency prewarping
`'matched'`	Matched pole-zero method (for SISO systems only)

`sldiscmdl('sys',sampletime,{options})` discretizes the model with the criteria specified by `{options}`, where `{options}` is a cell array containing the following string elements:

```
{'target','ReplaceWith','PutInto','prompt'}
```

Available values for `'target'` are shown below:

Value	Description
`'all'`	Discretize all continuous blocks
`'selected'`	Discretize selected blocks only
`'<full path name of block>'`	Discretize specified block

Available values for `'ReplaceWith'` are shown below:

Value	Description
`'parammask'`	Create discrete blocks whose parameters are retained from the corresponding continuous block
`'hardcoded'`	Create discrete blocks whose parameters are “hard_coded” values placed directly into the block’s dialog box.

Available values for `'PutInto'` are shown below:

Value	Description
`'current'`	Apply discretization to current model
`'configurable'`	Create discretization candidate in a configurable subsystem
`'untitled'`	Create discretization in a new untitled window
`'copy'`	Create discretization in copy of the original model

Available values for `'prompt'` are shown below:

Value	Description
'on'	Show the discretization information
'off'	Do not show the discretization information

`sldiscmdl('sys',sampletime,'method',cf)` discretizes the model with the critical frequency specified by `cf`. The units for `cf` are Hz. This is only used when the transform method is `'prewarp'`.

Examples

This command discretizes all of the continuous blocks in the `f14` model with a 1 second sample time.

```
sldiscmdl('f14',1.0)
```

This command discretizes the `Controller` subsystem in the `f14` model using a first-order hold transform method with a 1–second sample time and a 0.1–second sample time offset. The discretized block has "hard-coded" parameters that are placed directly into the block's dialog box.

```
sldiscmdl('f14',[1.0 0.1],'foh',{'f14/Controller',...
'hardcoded','copy','on'})
```

This command discretizes the `Controller` subsystem in the `f14` model using a zero-order hold transform method with a 1–second sample time and a 0.1–second sample time offset. It returns to the command window a cell array for the original continuous blocks in the system and a cell array for the discretized blocks in the system.

```
[a, b] = sldiscmdl('f14',[1.0 0.1],'zoh', {'f14/Controller',...
'hardcoded', 'copy', 'on'})
a =

    [1x43 char]    [1x37 char]    [1x53 char]    [1x30 char]


b =

    [1x43 char]    [1x37 char]    [1x53 char]    [1x30 char]
```

You can index into the cell arrays to get the new names of the discretized blocks and the original names of the continuous blocks.

For example, this command returns the name of the second discretized block.

```
b{2}

ans =

f14_disc_copy/Controller/Pitch Rate
Lead Filter
```

slmdldiscui

Purpose

Open the Model Discretizer GUI

Syntax

```
slmdldiscui('name')
```

Description

`slmdldiscui('name')` opens the Model Discretizer with the library or model specified by `'name'`.

Examples

This command opens the Model Discretizer with the `f14` model.

```
slmdldiscui('f14')
```

This command opens the Model Discretizer with the library named Test.

```
slmdldiscui('Test')
```

Purpose

Replace Mux blocks used to create buses with Bus Creator blocks

Syntax

```
[muxes, uniqueMuxes, uniqueBds] = slreplace_mux(model, reportonly)
```

Description

`slreplace_mux(model)` or `slreplace_mux(model, true)` reports all Mux blocks that create buses in `model` and in libraries referenced by `model`.

A signal created by a Mux block is a bus if the signal meets either or both of the following conditions:

- A Bus Selector block individually selects one or more of the signal's elements (as opposed to the entire signal).
- The signal's components have different data types, numeric types (complex or real), dimensionality, storage classes, or sampling modes.

Note Before running this command, you should set the **Mux blocks used to create bus signals** connectivity diagnostic to `warning` or `none`. See "Connectivity Diagnostics" for more information.

`slreplace_mux(model, false)` replaces all Mux blocks in `model` that create buses, including Mux blocks in libraries, with Bus Creator blocks. This command saves the model, if changed, and saves and closes any library that it modifies.

Note You should make a backup copy of your model and libraries before using this form of the command because it is difficult to undo its effects.

`[muxes, uniqueMuxes, uniqueBds] = slreplace_mux(model)` returns the following output variables:

- `muxes`

  All Mux blocks used as Bus Creators in the model and in libraries referenced by the model

- `uniqueMuxes`

  All Mux blocks used as Bus Creators in the model and in libraries referenced by the model except blocks in the model that are copies of blocks in libraries

- `uniqueBds`

  Models and libraries that use Mux blocks as Bus Creators

Purpose

Replace blocks from previous releases with the latest versions

Syntax

```
slupdate('sys')
slupdate('sys', prompt)
```

Description

`slupdate('sys')` replaces blocks in model `sys` from a previous release of Simulink with the latest versions. `slupdate('sys', prompt)` specifies whether to prompt you before replacing a block. If *`prompt`* equals `1`, the command prompts you before replacing the block. The prompt asks whether you want to replace the block. Valid responses are

- `y`

  Replace the block (the default).

- `n`

  Do not replace the block.

- `a`

  Replace this and all subsequent obsolete blocks without further prompting

  .

If `prompt` equals `0`, the command replaces all obsolete blocks without prompting you.

Note The model to be updated must have been created by a version of Simulink earlier than Simulink 5 and must be open when you call `slupdate`.

In addition to replacing obsolete blocks, `slupdate`

- Reconnects broken links to masked blocks in libraries provided by the MathWorks to ensure that the model reflects changes made to the

blocks in this release. This will overwrite any customizations that you have made to the masks of these blocks.

- Update obsolete configuration settings for the model

Purpose

Display a graph of model reference dependencies

Syntax

```
view_mdlrefs('model_name')
```

Description

`view_mdlrefs('model_name')` displays a graph of model reference dependencies for the model specified by `model_name`. The nodes in the graph represent Simulink models. The directed lines indicate model dependencies. For more information, see the `sldemo_mdlref_depgraph` demo.

See Also

`find_mdlrefs`

5

Simulation Commands

The following section describes commands that you can use to run simulations manually.

Task-Oriented Command List (p. 5-2)	Simulation commands listed by the tasks they perform.
Simulation Commands — Alphabetical List (p. 5-3)	Simulation commands listed in alphabetical order.

Task-Oriented Command List

This table lists the tasks performed by commands described in this section. An alphabetical list follows.

Task	Command
Simulate a dynamic system represented by a Simulink model.	`sim`
Get simulation options.	`simget`
Set simulation options.	`simset`
Plot simulation output.	`simplot`
Execute a particular phase of the simulation of a model.	`model`
Build simulation targets for models referenced by this model.	`slbuild`
Unpack a signal log.	`unpack`
List the names of signal logging objects in a signal log container object.	`who`
List the names and types of signal logging objects in a signal log container object.	`whos`

Simulation Commands — Alphabetical List

add_exec_event_listener
model
sim
simget
simplot
simset
slbuild
unpack
who
whos

add_exec_event_listener

Purpose

Register a listener for a block method execution event

Syntax

```
h = add_exec_event_listener(blk, event, listener);
```

Description

`h = add_exec_event_listener(blk, event, listener)` registers a listener for a block method execution event where the listener is an M-file program that performs some task, such as logging runtime data for a block, when the event occurs (see "Listening for Method Execution Events" in "Using Simulink®" for more information). Simulink invokes the registered listener whenever the specified event occurs during simulation of the model.

Note Simulink can register a listener only while a simulation is running. Invoking this function when no simulation is running results in an error message. To ensure that a listener catches all relevant events triggered by a model's simulation, you should register the listener in the model's StartFcn callback function (see "Model Callback Functions").

Arguments

`blk`

Specifies the block whose method execution event the listener is intended to handle. May be one of the following:

- Full pathname of a block
- A block handle
- A block runtime object (see "Accessing Block Data During Simulation" in "Using Simulink®".)

`event`

Specifies the type of event for which the listener listens. It may be any of the following:

Event	Occurs...
`'PreDerivatives'`	Before a block's Derivatives method executes
`'PostDerivatives'`	After a block's Derivatives method executes
`'PreOutputs'`	Before a block's Outputs method executes.
`'PostOutputs'`	After a block's Outputs method executes
`'PreUpdate'`	Before a block's Update method executes
`'PostUpdate'`	After a block's Update method executes

listener

Specifies the listener to be registered. It may be either a string specifying a MATLAB expression, e.g., `'disp(''here'')'` or a handle to a MATLAB function that accepts two arguments. The first argument is the block runtime object of the block that triggered the event. The second argument is an instance of `EventData` class that specifies the runtime object and the name of the event that just occurred.

Return Value

`add_exec_event_listener` returns a handle to the listener that it registered. To stop listening for an event, use the MATLAB `clear` command to clear the listener handle from the workspace in which the listener was registered.

model

Purpose

Execute a particular phase of the simulation of a model

Syntax

```
[sys,x0,str,ts] = model([],[],[],'sizes');
[sys,x0,str,ts] = model([],[],[],'compile');
outputs = model(t,x,u,'outputs');
derivs = model(t,x,u,'derivs');
dstates = model(t,x,u,'update');
model([],[],[],'term');
```

Description

The `model` command executes a specific phase of the simulation of a Simulink model whose name is `model`. The command's last (`flag`) argument specifies the phase of the simulation to be executed. See "Simulating Dynamic Systems" for a description of the steps that Simulink uses to simulate a model.

This command is intended to allow linear analysis and other M-file program-based tools to run a simulation step by step, gathering information about the model's states and outputs at each step. It is not intended to be used to run a model step by step, for example, to debug a model. Use the Simulink debugger if you need to examine intermediate results to debug a model.

Arguments

`sys`	Vector of model size data: • `sys(1)` = number of continuous states • `sys(2)` = number of discrete states • `sys(3)` = number of outputs • `sys(4)` = number of inputs • `sys(5)` = reserved • `sys(6)` = direct-feedthrough flag (1 = yes, 0 = no) • `sys(7)` = number of sample times (= number of rows in `ts`)
`x0`	Vector containing the initial conditions of the system's states
`str`	Vector of names of the blocks associated with the model's states. The state names and initial conditions appear in the same order in `str` and `x0`, respectively.
`ts`	An *m*-by-2 matrix containing the sample time (period, offset) information
outputs	Outputs of the model at time step `t`.
derivs	Derivatives of the continuous states of the model at time `t`.
dstates	Discrete states of the model at time `t`.
`t`	Time step
`x`	State vector

u	Inputs
flag	String that indicates the simulation phase to be executed: • 'sizes' executes the size computation phase of the simulation. This phase determines the sizes of the model's inputs, outputs, state vector, etc. • 'compile' executes the compilation phase of the simulation. The compilation phase propagates signal and sample time attributes. It is equivalent to selecting the **Update Diagram** (**Ctrl-D**) option from the Simulink **Edit** menu. • 'update' computes the next values of the model's discrete states. • 'outputs' computes the outputs of the model's blocks at time t. • 'derivs'computes the derivatives of the model's continuous states at time step t. • 'term' causes Simulink to terminate simulation of the model.

Examples

This command executes the compilation phase of the vdp model that comes with Simulink.

```
vdp([], [], [], 'compile')
```

The following command terminates the simulation initiated in the previous example.

```
vdp([], [], [], 'term')
```

Note You must always terminate simulation of the model by invoking the model command with the `'term'` command. Simulink does not let you close the model until you have terminated the simulation.

See Also

`sim`

Purpose

Simulate a dynamic system

Syntax

```
[t,x,y] = sim(model,timespan,options,ut);
[t,x,y1, y2, ..., yn] = sim(model,timespan,options,ut);
```

Description

The `sim` command executes a Simulink model, using all **Configuration Parameters** dialog box settings, including the options specified on the **Data Import/Export** pane.

You can supply a null ([ ]) matrix for any right-side argument except the first (the model name). The `sim` command uses default values for unspecified arguments and arguments specified as null matrices. The default values are the values specified by the model. You can set optional simulation parameters, using the `sim` command's `options` argument. Parameters set in this way override parameters specified by the model.

If you do not specify the left side arguments, the command logs the simulation data specified by the **Data Import/Export** pane of the **Configuration Parameters** dialog box (see "Data Import/Export Pane" in "Using Simulink®").

If you want to simulate a continuous system, you must specify the `solver` parameter, using `simset`. The solver defaults to `VariableStepDiscrete` for purely discrete models.

Note The base workspace for a simulation launched by the `sim` command is the MATLAB workspace by default, with one exception. The exception is that the default workspace for To Workspace blocks is the current workspace, i.e., the workspace of the function that invoked the `sim` command. You can use the `DstWorkspace` and `SrcWorkspace` options of the `sim` command (see `simset`) to change these defaults. For example, suppose that you want to use the workspace of the function that invokes the `sim` command as the base workspace of the simulation. To do this, specify `current` as the value of the `DstWorkspace` and `SrcWorkspace` options.

Arguments

`t`	Returns the simulation's time vector.
`x`	Returns the simulation's state matrix consisting of continuous states followed by discrete states.
`y`	Returns the simulation's output matrix. Each column contains the output of a root-level Outport block, in port number order. If any Outport block has a vector input, its output takes the appropriate number of columns.
`y1,...,yn`	Each y_i returns the output of the corresponding root-level Outport block for a model that has `n` such blocks.
`model`	Name of a block diagram.
`timespan`	Simulation start and stop time. Specify as one of these: `tFinal` to specify the stop time. The start time is 0. `[tStart tFinal]` to specify the start and stop times. `[tStart OutputTimes tFinal]` to specify the start and stop times and time points to be returned in `t`. Generally, `t` will include more time points. `OutputTimes` is equivalent to choosing **Produce additional output** on the dialog box.

`options`	Optional simulation parameters specified as a structure created by the `simset` command (see `simset`).
`ut`	Optional external inputs to top-level Inport blocks. `ut` can be a MATLAB function (expressed as a string) that specifies the input `u = UT(t)` at each simulation time step, a table of input values versus time for all input ports, or a comma-separated list of tables, `ut1`, `ut2`, ..., each of which corresponds to a specific port. Tabular input for all ports can be in the form of a MATLAB array or a structure. Tabular input for individual ports must be in the form of a structure. See "Importing Data from the MATLAB Workspace" in the online documentation for a description of the array and structure input formats.

Examples

This command simulates the Van der Pol equations, using the `vdp` model that comes with Simulink. The command uses all default parameters.

```
[t,x,y] = sim('vdp')
```

This command simulates the Van der Pol equations, using the parameter values associated with the `vdp` model, but defines a value for the `Refine` parameter.

```
[t,x,y] = sim('vdp', [], simset('Refine',2));
```

This command simulates the Van der Pol equations for 1,000 seconds, saving the last 100 rows of the return variables. The simulation outputs values for `t` and `y` only, but saves the final state vector in a variable called `xFinal`.

```
[t,x,y] = sim('vdp', 1000, simset('MaxRows', 100,
          'OutputVariables', 'ty', 'FinalStateName', 'xFinal'));
```

See Also

simset, simget

simplot

Purpose

Plot simulation data in a figure window

Syntax

```
simplot(data);
simplot(time, data);
```

Description

The `simplot` command plots output from a simulation in a Handle Graphics® figure window. The plot looks like the display on the screen of a Scope block. Plotting the output on a figure window allows you to annotate and print the output.

Arguments

`data`	Data produced by one of the Simulink output blocks (for example, a root-level Outport block or a To Workspace block) or in one of the output formats used by those blocks: **Array**, **Structure**, **Structure with time** (see "Data Import/Export Pane" in "Using Simulink®").
`time`	The vector of sample times produced by an output block when you have selected **Array** or **Structure** as the simulation's output format. The `simplot` command ignores this argument if the format of the data is **Structure with time**.

Examples

The following sequence of commands

```
vdp
set_param(gcs, 'SaveOutput', 'on')
set_param(gcs, `SaveFormat', `StructureWithTime')
sim(gcs)
simplot(yout)
```

plots the output of the `vdp` demo model on a figure window as follows.

See Also

`sim`, `set_param`

Purpose

Draw a graph connecting a series of points

Syntax

```
plot(Y)
plot(X1,Y1,X2,Y2,...)
```

Description

`plot(Y)` plots, for a vector *Y*, each element against its index. If *Y* is a matrix, it plots each column of the matrix as though it were a vector.

`plot(X1,Y1,X2,Y2,...)` plots the vectors *Y1* against *X1*, *Y2* against *X2*, and so on. Vector pairs must be the same length and the list must consist of an even number of vectors.

Plot commands can include `NaN` and `inf` values. When `NaN`s or `inf`s are encountered, Simulink stops drawing, then begins redrawing at the next numbers that are not `NaN` or `inf`.

The appearance of the plot on the icon depends on the value of the **Drawing coordinates** parameter. For more information, see "Icon options" in the "Using Simulink®".

Simulink displays three question marks (`? ? ?`) in the block icon and issues warnings in these situations:

- When the values for the parameters used in the drawing commands are not yet defined (for example, when the mask is first created and values have not yet been entered in the mask dialog box)
- When a masked block parameter or drawing command is entered incorrectly

Examples

This command

```
plot([0 1 5], [0 0 4])
```

generates the plot that appears on the icon for the Ramp block, in the Sources library.

See Also

`image`

Purpose

Draw a port label on the icon of a masked subsystem

Syntax

```
port_label('port_type', port_number, 'label')
port_label('port_type', port_number, 'label','texmode','on')
```

Description

`port_label('port_type', port_number, 'label')` draws a label on a port where *`port_type`* is either `'input'` or `'output'`, *`port_number`* is an integer, and *`label`* is a string specifying the port's label.

`port_label('port_type', port_number, 'label','texmode','on')` lets you use TeX formatting commands in `label`. The TeX formatting commands allow you to include symbols and Greek letters in the port label. See "Mathematical Symbols, Greek Letters, and Tex Characters" in the MATLAB documentation for information on the TeX formatting commands supported by Simulink.

Examples

The command

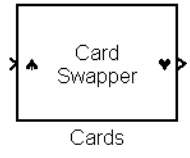
```
port_label('input', 1, 'a')
```

defines a as the label of input port 1.

The commands

```
disp('Card\nSwapper');
port_label('input',1,'\spadesuit','texmode','on');
port_label('output',1,'\heartsuit','texmode','on');
```

draw playing card symbols as the labels of the ports on a masked subsystem.



See Also `disp`, `fprintf`, `text`

Purpose

Display text at a specific location on the icon of a masked subsystem

Syntax

```
text(x, y, 'text')
text(x, y, 'text', 'horizontalAlignment', 'halign',
  'verticalAlignment', 'valign')
text(x, y, 'text', 'texmode', 'on')
```

Description

The `text` command places a character string at a location specified by the point `(x,y)`. The units depend on the **Drawing coordinates** parameter. For more information, see "Icon options".

`text(x,y, text, 'texmode', 'on')` allows you to use TeX formatting commands in *`text`*. The TeX formatting commands in turn allow you to include symbols and Greek letters in icon text. See in the MATLAB documentation for information on the TeX formatting commands supported by Simulink.

You can optionally specify the horizontal and/or vertical alignment of the text relative to the point (x, y) in the `text` command.

The `text` command offers the following horizontal alignment options.

Option	Aligns
`'left'`	The left end of the text at the specified point
`'right'`	The right end of the text at the specified point
`'center'`	The center of the text at the specified point

The `text` command offers the following vertical alignment options.

Option	Aligns
`'base'`	The baseline of the text at the specified point
`'bottom'`	The bottom line of the text at the specified point
`'middle'`	The midline of the text at the specified point

Option	Aligns
'cap'	The capitals line of the text at the specified point
'top'	The top of the text at the specified point

Note While this command is identical in name to its corresponding MATLAB function, it provides only the functionality described above.

Examples

The command

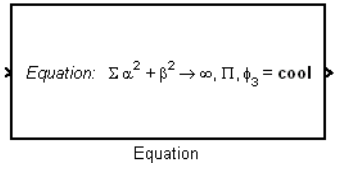
```
text(0.5, 0.5, 'foobar', 'horizontalAlignment', 'center')
```

centers `foobar` in the icon.

The command

```
text(.05,.5,'{\itEquation:}  \Sigma \alpha^2 +
\beta^2 \rightarrow \infty, \Pi, \phi_3 = {\bfcool}',
'hor','left','texmode','on')
```

draws a left-aligned equation on the icon.



See Also

`disp`, `fprintf`, `port_label`

7

Simulink Debugger Commands

The following sections describe commands that you can use to create programs that create or modify models.

Command Summary (p. 7-2)	Brief descriptions of commands.
Simulink Debugger Commands — Alphabetical List (p. 7-5)	

Command Summary

The following table lists the debugger commands. The table's Repeat column specifies whether pressing the **Enter** key at the command line repeats the command. Detailed descriptions of the commands follow the table.

Command	Short Form	Repeat	Description
`animate`	`ani`	No	Enable/disable animation mode.
`ashow`	`as`	No	Show an algebraic loop.
`atrace`	`at`	No	Set algebraic loop trace level.
`bafter`	`ba`	No	Insert a breakpoint after a method.
`break`	`b`	No	Insert a breakpoint before a method.
`bshow`	`bs`	No	Show a specified block.
`clear`	`cl`	No	Clear breakpoints from a model.
`continue`	`c`	Yes	Continue the simulation.
`disp`	`d`	Yes	Display a block's I/O when the simulation stops.
`ebreak`	eb	No	Break at recoverable solver errors.
`elist`	el	No	Display method execution order.
`emode`	em	No	Toggle between accelerated and normal mode.
`etrace`	et	No	Enable or disable method tracing.
`help`	`?` or `h`	No	Display help for debugger commands.

Command	Short Form	Repeat	Description
`minor`	`m`	No	Enable or disable breakpoints in minor time steps.
`nanbreak`	`na`	No	Set or clear nonfinite value break mode.
`next`	`n`	Yes	Go to start of the next time step.
`probe`	`p`	No	Display block data.
`quit`	`q`	No	Abort simulation.
`rbreak`	`rb`	No	Toggle solver reset breakpoint.
`run`	`r`	No	Run the simulation to completion.
`stimes`	`sti`	No	Display a model's sample times.
`slist`	`sli`	No	Display a model's sorted lists.
`states`	`state`	No	Display current state values.
`status`	`stat`	No	Display debugging options in effect.
`step`	`s`	Yes	Advance the simulation by one or more methods.
`stop`	`sto`	No	Stop the simulation.
`strace`	`i`	No	Set solver trace level.
`systems`	`sys`	No	List a model's nonvirtual systems.
`tbreak`	`tb`	No	Set or clear a time breakpoint.
`trace`	`tr`	Yes	Display a block's I/O each time the block executes.

Command	Short Form	Repeat	Description
`undisp`	`und`	Yes	Remove a block from the debugger's list of display points.
`untrace`	`unt`	Yes	Remove a block from the debugger's list of trace points.
`where`	w	No	Display the current location of the simulation in the simulation loop.
`xbreak`	`x`	No	Break when the debugger encounters a step-size-limiting state.
`zcbreak`	`zcb`	No	Toggle breaking at nonsampled zero-crossing events.
`zclist`	`zcl`	No	List blocks containing nonsampled zero crossings.

Simulink Debugger Commands — Alphabetical List

animate
ashow
atrace
bafter
break
bshow
clear
continue
disp
ebreak
elist
emode
etrace
help
minor
nanbreak
next
probe
quit
rbreak
run
slist
states
status
step
stimes
stop
strace
systems
tbreak
trace
undisp
untrace
where
xbreak

zcbreak
zclist

Purpose

Enable or disable animation mode

Syntax

```
animate [delay | stop]
```

Arguments

`delay`	Length in seconds between method calls (1 second by default).
`stop`	Disable animation mode.

Description

`animate` without any arguments enables animation mode. `animate delay` enables animation mode and specifies delay as the time delay in seconds between method calls. `animate` **`stop`** disables animation mode.

See Also

`continue`

ashow

Purpose

Show an algebraic loop

Syntax

```
ashow <gcb | s:b | s#n | clear>
```

Arguments

`gcb`	Current block.
`s:b`	The block whose system index is `s` and block index is `b`.
`s#n`	The algebraic loop numbered `n` in system `s`.
`clear`	Switch that clears loop coloring.

Description

`ashow` without any arguments lists all of a model's algebraic loops in the MATLAB Command Window. `ashow gcb` or `ashow s:b` highlights the algebraic loop that contains the specified block. `ashow s#n` highlights the `n`th algebraic loop in system `s`. The `ashow` **`clear`** command removes algebraic loop highlights from the model diagram.

See Also

`atrace`, `slist`

Purpose

Set algebraic loop trace level

Syntax

```
atrace level
```

Arguments

`level` Trace level (0 = none, 4 = everything).

Description

The `atrace` command sets the algebraic loop trace level for a simulation.

Command	Displays for Each Algebraic Loop
`atrace 0`	No information
`atrace 1`	The loop variable solution, the number of iterations required to solve the loop, and the estimated solution error
`atrace 2`	Same as level 1
`atrace 3`	Level 2 plus Jacobian matrix used to solve loop
`atrace 4`	Level 3 plus intermediate solutions of the loop variable

See Also

`states`, `systems`

bafter

Purpose

Insert a breakpoint after a specified method

Syntax

```
bafter
bafter m:mid
bafter <sid:bid | gcb> [mth] [tid:TID]
bafter <s:sid | gcs> [mth] [tid:TID]
bafter mdl [mth] [tid:TID]
```

Arguments

mid	Method ID
sid:*bid*	Block ID
gcb	Currently selected block
sid	System ID
gcs	Currently selected system
mdl	Currently selected model
mth	A method name, e.g., `Outputs.Major`
TID	Task ID

Description

`bafter` inserts a breakpoint after the current method.

`bafter m:`*mid* inserts a breakpoint after the method specified by *mid* (see "Method ID").

`bafter` *sid*:*bid* inserts a breakpoint after each invocation of the method of the block specified by *sid*:*bid* (see "Block ID") in major time steps. `bafter gcb` inserts a breakpoint after each invocation of a method of the currently selected block (see `gcb`) in major times steps. To break at invocation of a block's methods in minor time steps, execute `minor` before or after executing this form of the `bafter` command.

`bafter s:`*sid* inserts a breakpoint after each method of the root system or nonvirtual subsystem specified by the system ID: *sid*.

Note The `systems` command displays the system IDs for all nonvirtual systems in the currently selected model.

`bafter` **`gcs`** inserts a breakpoint after each method of the currently selected nonvirtual system.

`bafter` **`mdl`** inserts a breakpoint after each method of the currently selected model.

The optional *`mth`* parameter allow you to set a breakpoint after a particular block, system, or model method and task. For example, `bafter` **`gcb`** `Outputs` sets a breakpoint after the Outputs method of the currently selected block.

The optional TID parameter allows you to set a breakpoint after invocation of a method by a particular task. For example, suppose that the currently selected nonvirtual subsystem operates on task 2 and 3. Then `bafter` **`gcs`** `Outputs` **`tid:`**`2` sets a breakpoint after the invocation of the subsystem's Outputs method that occurs when task 2 is active.

See Also

`break`, `ebreak`, `tbreak`, `xbreak`, `nanbreak`, `zcbreak`, `rbreak`, `clear`, `where`, `slist`, `systems`

break

Purpose

Insert a breakpoint before a specified method

Syntax

```
break
break m:mid
break <sid:bid | gcb> [mth] [tid:TID]
break <s:sid | gcs> [mth] [tid:TID]
break mdl [mth] [tid:TID]
```

Arguments

mid	Method ID
sid:*bid*	Block ID
gcb	Currently selected block
sid	System ID
gcs	Currently selected system
mdl	Currently selected model
mth	A method name, e.g., `Outputs.Major`
TID	task ID

Description

`break` inserts a breakpoint before the current method.

`break` **m:***mid* inserts a breakpoint before the method specified by *mid* (see "Method ID").

`break` *sid*:*bid* inserts a breakpoint before each invocation of the method of the block specified by *sid*:*bid* (see "Block ID") in major time steps. `break` gcb inserts a breakpoint before each invocation of a method of the currently selected block (see `gcb`) in major times steps. To break before invocation of a block's methods in minor time steps, execute `minor` before or after executing this form of the `bafter` command.

`break` **s:***sid* inserts a breakpoint at each method of the root system or nonvirtual subsystem specified by the system ID: *sid*.

Note The `systems` command displays the system IDs for all nonvirtual systems in the currently selected model.

`break gcs` inserts a breakpoint at each method of the currently selected nonvirtual system.

`break mdl` inserts a breakpoint at each method of the currently selected model.

The optional *`mth`* parameter allow you to set a breakpoint at a particular block, system, or model method. For example, `break gcb Outputs` sets a breakpoint at the Outputs method of the currently selected block.

The optional TID parameter allows you to set a breakpoint at the invocation of a method by a particular task. For example, suppose that the currently selected nonvirtual subsystem operates on task 2 and 3. Then `break gcs Outputs tid:2` sets a breakpoint at the invocation of the subsystem's Outputs method that occurs when task 2 is active.

See Also

`bafter`, `clear`, `ebreak`, `nanbreak`, `rbreak`, `systems`, `tbreak`, `where`, `xbreak`, `zcbreak`, `slist`

bshow

Purpose

Show a specified block

Syntax

```
bshow s:b
```

Arguments

`s:b`	The block whose system index is `s` and block index is `b`.

Description

The `bshow` command opens the model window containing the specified block and selects the block.

See Also

`slist`

Purpose

Clear breakpoints from a model

Syntax

```
clear
clear m:mid
clear id
clear <sid:bid | gcb>
```

Arguments

mid	Method ID
id	Breakpoint ID
sid:*bid*	Block ID
gcb	Currently selected block

Description

`clear` clears a breakpoint from the current method.

`clear m:`*`mid`* clears a breakpoint from the method specified by *`mid`*.

`clear` *`id`* clears the breakpoint specified by the breakpoint ID *`id`*.

`clear` *`sid`*:*`bid`* clears any breakpoints set on the methods of the block specified by *`sid:bid`*.

`clear` **`gcb`** clears any breakpoints set on the methods of the currently selected block.

See Also

`break`, `bafter`, `slist`

continue

Purpose

Continue the simulation

Syntax

```
continue
```

Description

The `continue` command continues the simulation from the current breakpoint. If animation mode is not enabled, the simulation continues until it reaches another breakpoint or its final time step. If animation mode is enabled, the simulation continues in animation mode to the first method of the next major time step, ignoring breakpoints.

See Also

`run`, `stop`, `quit`, `animate`

Purpose

Display a block's I/O when the simulation stops

Syntax

```
disp
disp gcb
disp s:b
```

Arguments

`s:b`	The block whose system index is `s` and block index is `b`.
`gcb`	Current block.

Description

The `disp` command registers a block as a display point. The debugger displays the inputs and outputs of all display points in the MATLAB Command Window whenever the simulation halts. Invoking `disp` without arguments shows a list of display points. Use `undisp` to unregister a block.

See Also

`undisp`, `slist`, `probe`, `trace`

ebreak

Purpose

Enable (or disable) a breakpoint on solver errors.

Syntax

```
ebreak
```

Description

This command causes the simulation to stop if the solver detects a recoverable error in the model. If you do not set or disable this breakpoint, the solver recovers from the error and proceeds with the simulation without notifying you.

See Also

`break`, `bafter`, `tbreak`, `xbreak`, `nanbreak`, `zcbreak`, `rbreak`, `clear`, `where`, `slist`, `systems`

Purpose

List simulation methods in the order in which they are executed during a simulation

Syntax

```
elist m:mid [tid:TID]
elist <gcs | s:sid> [mth] [tid:TID]
elist <gcb | sid:bid> [mth] [tid:TID]
```

Description

`elist m:`*`mid`* lists the methods invoked by the system or nonvirtual subsystem method corresponding to the method id *`mid`* (see the `where` command for information on method IDs), e.g.,

The method list specifies the calling method followed by the methods

help

Purpose

Display help for debugger commands

Syntax

```
help
```

Description

The `help` command displays a list of debugger commands in the command window. The list includes the syntax and a brief description of each command.

Purpose

Enable or disable breakpoints in minor time steps

Syntax

```
minor
```

Description

The `minor` command causes the debugger to enable or disable minor step breakpoint mode. By default, executing `break <sid:bid | gcb>` sets breakpoints only on invocations of a block's methods, e.g., Outputs, in major time steps. To enable breakpoints at invocations of the methods in minor time steps, you must execute `minor` before or after executing the break command. To disable breaking in minor steps, execute `minor` again.

See Also

`break`, `bafter`

nanbreak

Purpose

Set or clear nonfinite value break mode

Syntax

```
nanbreak
```

Description

The `nanbreak` command causes the debugger to break whenever the simulation encounters a nonfinite (`NaN` or `Inf`) value. If nonfinite break mode is set, `nanbreak` clears it.

See Also

`break`, `bafter`, `ebreak`, `rbreak`, `tbreak`, `xbreak`, `zcbreak`

Purpose

Advance the simulation to the start of the next method at the current level in the model's execution list

Syntax

```
next
```

Description

The `next` command advances the simulation to the start of the next method at the current level in the model's method execution list.

Note The next command has the same effect as the `step over` command. See `step` for more information.

See Also

`step`

probe

Purpose

Display block data.

Syntax

```
probe [<s:b | gcb>] [level io | (all)]
```

Arguments

`s:b`	The block whose system index is `s` and block index is `b`.
`gcb`	Current block.
level io	Display block's I/O.
level all	Display all information regarding a block's current state, including inputs and outputs, states, and zero crossings.

Description

`probe` causes the debugger to enter or exit probe mode. In probe mode, the debugger displays the I/O of any block you select. To exit probe mode, enter any command. `probe gcb` displays the I/O of the currently selected block. `probe s:b` displays the I/O of the block whose index is `s:b`.

See Also

`disp`, `trace`

Purpose

Abort simulation

Syntax

```
quit
```

Description

The `quit` command terminates the current simulation.

See Also

`stop`

rbreak

Purpose

Break when the simulation requires a solver reset.

Syntax

```
rbreak
```

Description

This command enables (or disables) a solver reset breakpoint if the breakpoint is disabled (or enabled). The breakpoint causes the debugger to halt the simulation whenever an event that requires a solver reset occurs. The halt occurs before the solver is reset.

See Also

`break`, `bafter`, `ebreak`, `nanbreak`, `tbreak`, `xbreak`, `zcbreak`,

Purpose

Run the simulation to completion

Syntax

```
run
```

Description

The `run` command runs the simulation from the current breakpoint to its final time step. It ignores breakpoints and display points.

See Also

`continue`, `stop`, `quit`

slist

Purpose

Display the sorted list of a model's root system and of each of its nonvirtual subsystems

Syntax

```
slist
```

Description

The `slist` command displays the sorted list of a model's root system and each of its nonvirtual subsystems. For example, the sorted list for the `vdp` model's root system is

```
---- Sorted list for 'vdp' [9 nonvirtual blocks, directFeed=0]
  0:0    'vdp/x1' (Integrator)
  0:1    'vdp/Out1' (Outport)
  0:2    'vdp/x2' (Integrator)
  0:3    'vdp/Out2' (Outport)
  0:4    'vdp/Scope' (Scope)
  0:5    'vdp/Fcn' (Fcn)
  0:6    'vdp/Product' (Product)
  0:7    'vdp/Mu' (Gain)
  0:8    'vdp/Sum' (Sum)
```

For each system (root or nonvirtual), the `slist` command displays a title line followed by an entry for each block in the order in which the blocks appear in the sorted list. The title line specifies the name of the system, the number of nonvirtual blocks that the system contains, and the number of blocks in the system that have direct feedthrough ports. Each block entry lists the block's id and the name and type of the block. The block id consists of a system index and a block index separated by a colon (`s:b`). The block index is the position of the block in the sorted list. The system index is the order in which Simulink generated the system's sorted list. The system index has no special significance. It simply allows blocks that appear in the same position in different sorted lists to have unique identifiers.

A sorted list is a list of a root system or nonvirtual subsystem's blocks sorted according to data dependencies and other criteria. Simulink uses sorted lists to create block method execution lists (see `elist`) for root system and nonvirtual subsystem methods that invoke the

corresponding methods of the blocks that the root system or subsystem contains. In general, root system and nonvirtual subsystem methods invoke the block methods in the same order as the blocks appear in the sorted list. However, significant exceptions occur. For example, execution lists for multitask models group all blocks operating at the same rate (i.e., in the same task) together with slower groups appearing later than faster groups. The grouping of methods by task can result in an order of block method execution that differs from the order in which blocks appear in the sorted list. However, within groups, methods execute in the same order as the corresponding blocks appear in the sorted list.

See Also

`systems`, `elist`

states

Purpose

Display current state values

Syntax

```
states
```

Description

The `states` command displays a list of the current states of the model. The display lists the value, index, and name of each state.

Purpose

Display debugging options in effect

Syntax

```
status
```

Description

The `status` command displays a list of the debugging options in effect.

step

Purpose

Advance the simulation by one or more methods

Syntax

```
step [in into]
step over
step out
step top
step blockmth
```

Description

This command advances the simulation

- Into (`step [in into]`), over (`step over`), or out of the method at

Input	Value
`'funcstr'`	Function of `x` `funcstr` is the function for which breakpoints are approximated.
`xmin`	Minimum value of `x`
`xmax`	Maximum value of `x`
`xdt`	Data type of `x`
`xscale`	Scaling for the `x` values
`ydt`	Data type of `y`
`yscale`	Scaling for the `y` values
`rndmeth`	Rounding mode supported by fixed-point Simulink blocks: `'Toward Zero'`, `'Nearest'`, `'Floor'` (default value), `'Ceiling'`

- `xmin` and `xmax` specify the range over which the breakpoints are approximated.
- `xdt`, `xscale`, `ydt`, `yscale`, and `rndmeth` follow conventions used by fixed-point Simulink blocks.
- `rndmeth` has a default value listed in the input table.

In addition to the required parameters, there are three optional inputs, as follows.

Input	Value
`errmax`	Maximum acceptable error
`nptsmax`	Maximum number of points
`errworst`	Spacing: `'even'`, `'pow2'` (even power of 2), `'unrestricted'` (default value)

Of these, you must use at least one of the parameters `errmax` and `nptsmax`. If you omit one of these, use brackets, `[]`, in place of the

omitted parameter. The function will then ignore that requirement for the lookup table.

The outputs of the function are as follows.

Output	Value
`xdata`	The breakpoints for the lookup table
ydata	The ideal function applied to the breakpoints
`errworst`	The worst case error, which is the maximum absolute error between the ideal function and the approximation given by the lookup table

Criteria For Optimizing the Breakpoints: errmax, nptsmax, and spacing

The approximation produced from the lookup table must satisfy the requirements for the maximum acceptable error, `errmax`, the maximum number of points, `nptsmax`, and the spacing, `spacing`. The requirements are

- The maximum absolute error is less than `errmax`.
- The number of points required is less than `nptsmax`.
- The spacing is specified as unrestricted, even or even power of 2.

Modes for errmax and nptsmax

- If both `errmax` and `nptsmax` are specified

  The returned breakpoints will meet both criteria if possible. The `errmax` parameter is given priority, and `nptsmax` is ignored, if both criteria cannot be met with the specified spacing.

- If only `errmax` is specified

  The breakpoints that meet the error criteria, and have the least number of points are returned

- If only `nptsmax` is specified

The breakpoints that require `nptsmax` or fewer, and give the smallest worst case error are returned

Modes for Spacing

If no spacing is specified, and more than one spacing method meets the requirements given by `errmax` and `nptsmax`, power of 2 spacing is chosen over even spacing, which in turn is chosen over uneven spacing. This case occurs when the `errmax` and `nptsmax` are both specified, but typically does not occur when only one is specified:

- If `unrestricted` is entered, the function chooses the spacing that provides the best optimization.
- If `even` is entered, the function chooses an evenly spaced set of points, including the `pow2` spacing.
- If `pow2` spacing is entered, the function chooses an even power of 2 spaced set of points.

Note The global optimum may not be found. The worst case error can depend on fixed-point calculations, which are highly nonlinear. Furthermore, the optimization approach is heuristic.

The spacing you choose depends on the parameters you want to optimize: execution speed, function approximation error, ROM usage, and RAM usage:

- The execution speed depends on the bisection search, and the interpolation method.
- The error depends on how accurately the method approximates the nonuniform curvature of the function.
- The ROM usage depends on the amount of data and command ROM used.
- The RAM usage depends on how much global and stack RAM is used.

When the lookup table has even power of two spacing, division is replaced by a bit shift. As a result, the execution speed is faster than for evenly spaced data.

Using the Approximation Function

1. Choose a function and use the `eval('funcstr');` command to view the function before creating the lookup table.
2. Define the remaining inputs.
3. Run the `fixpt_look1_func_approx` function.
4. Use the `fixpt_look1_func_plot` function to plot the function from the selected breakpoints, and to calculate the error and the number of points used.
5. Vary the inputs to produce sets of breakpoints that generate functions with varying number of points required and worst case error.
6. Compare the number of points required and worst case error from various runs to choose the best set of breakpoints.

Calculating the Output Function

To calculate the function, use the returned breakpoints with

- The `eval` function
- A function lookup table. The `x` values are the breakpoints from the `fixpt_look1_func_approx` function, and the `y` values can be supplied using the `eval` function.

See "Tutorial: Producing Lookup Table Data" in "Simulink Fixed Point" for a tutorial on using `fixpt_look1_func_approx`.

The following table summarizes the effect of spacing on the execution speed, error, and memory used.

Parameter	Even Power of 2 Spaced Data	Evenly Spaced Data	Unevenly Spaced Data
Execution Speed	The execution speed is the fastest. The position search and interpolation are the same as for evenly spaced data. However, to increase the speed more, the position search is replaced by a bit shift, and the interpolation is replaced with a bit mask.	The execution speed is faster than that for unevenly spaced data because the position search is faster and the interpolation requires a simple division.	The execution speed is the slowest of the different spacings because the position search is slower, and the interpolation requires more operations.
Error	The error can be larger than that for unevenly spaced data because approximating a function with nonuniform curvature requires more points to achieve the same accuracy.	The error can be larger than that for unevenly spaced data because approximating a function with nonuniform curvature requires more points to achieve the same accuracy.	The error can be smaller because approximating a function with nonuniform curvature requires fewer points to achieve the same accuracy.
ROM Usage	Uses less command ROM, but more data ROM.	Uses less command ROM, but more data ROM.	Uses more command ROM, and less data ROM.
RAM Usage	Not significant.	Not significant.	Not significant.

Examples

This example produces a lookup table for a sine function. The inputs for the example are as follows:

```
funcstr = 'sin(2*pi*x)';
xmin = 0;
xmax = 0.25;
xdt = ufix(16);
xscale = 2^-16;
ydt = sfix(16);
yscale = 2^-14;
rndmeth = 'Floor';
errmax = 2^-10;
spacing = 'pow2';
```

To create the lookup table, type

```
[xdata, ydata, errWorst]=fixpt_look1_func_approx(funcstr,
xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,errmax,[],spacing);
```

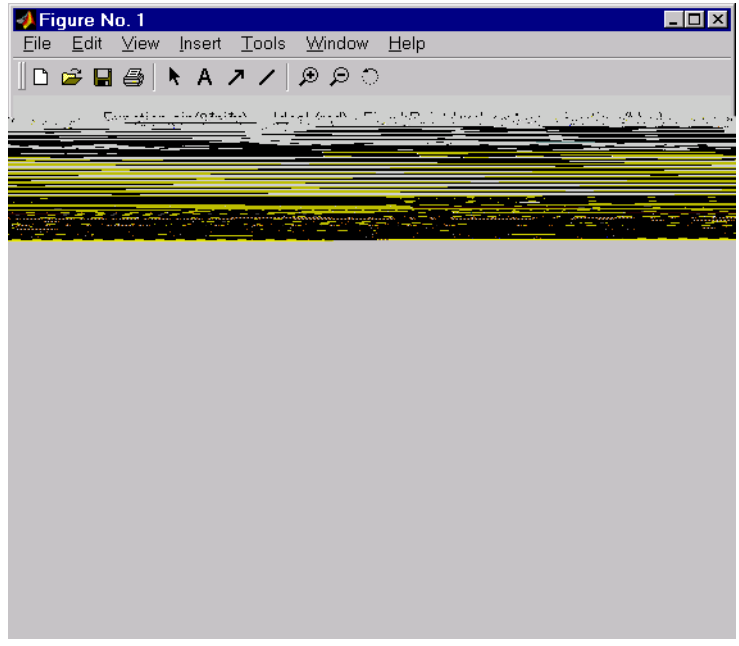
The brackets [ ] are a place holder for the `nptsmax` parameter, which is not used in this example.

You can then plot the ideal function, the approximation, and the errors by typing

```
fixpt_look1_func_plot(xdata,ydata,funcstr,xmin,xmax,xdt,...
xscale,ydt,yscale,rndmeth);
```

The `fixpt_look1_func_plot` function produces a plot of the fixed-point sine function, using these breakpoints, and a plot of the error between the ideal function and the fixed-point function. The maximum absolute error and the number of points required are listed with the plot. The error drops to zero at a breakpoint, and increases between breakpoints due to the difference in curvature of the ideal function and the line drawn between breakpoints.

The resulting plots are shown.



The lookup table requires 33 points to achieve a maximum absolute error of `2^-11.3922`.

See Also

`fixpt_look1_func_plot`

Purpose

Create a MATLAB structure describing an unsigned integer data type

Syntax

```
a = uint(TotalBits)
```

Description

`uint(TotalBits)` returns a MATLAB structure that describes the data type of an unsigned integer with a word size given by `TotalBits`.

`uint` is automatically called when an unsigned integer is specified in a block dialog box.

The default binary point for this data type is assumed to lie to the right of all bits.

Examples

Define a 16-bit unsigned integer:

```
a = uint(16)
a =
       Class: 'INT'
    IsSigned: 0
    MantBits: 16
```

See Also

`fixdt`, `float`, `sfix`, `sfrac`, `sint`, `ufix`, `ufrac`

9

Data Object Classes

The following sections describe the properties and usage of the following classes of Simulink data objects (see “Working with Data Objects” “Using Simulink®” for general information on creating and using Simulink data objects).

Class Summary (p. 9-2)

Classes — Alphabetical List (p. 9-5)

Class Summary

The following table briefly describes the purpose of each Simulink data object class.

Class	Purpose
`Simulink.BlockData`	Provide run-time information about block-related data, such as block parameters.
`EventData`	Provides information about block method execution events.
`Simulink.AliasType`	Specifies an alternate name for an existing data type.
`Simulink.BlockCompDworkData`	Provides postcompilation information about a block's Dwork vector.
`Simulink.BlockCompInputPortData`	Provides postcompilation information about a block input port.
`Simulink.BlockCompOutputPortData`	Provides postcompilation information about a block output port.
`Simulink.BlockData`	Provide run-time information about block-related data, such as block parameters.
`Simulink.BlockPortData`	Describe a block input or output port.
`Simulink.BlockPreCompInputPortData`	Provides precompilation information about a block input port.
`Simulink.BlockPreCompOutputPortData`	Provides precompilation information about a block output port.
`Simulink.Bus`	Describes a signal bus.

Class	Purpose
`Simulink.BusElement`	Describe an element of a signal bus.
`Simulink.ConfigSet`	Access a model configuration set.
`Simulink.ModelDataLogs`	Stores a model's signal logs.
`Simulink.ModelWorkspace`	Accesses a model's workspace.
`Simulink.MSFcnRunTimeBlock`	Get run-time information about a Level-2 M-file S-function block.
`Simulink.NumericType`	Describes a numeric data type.
`Simulink.Parameter`	Describes the value of a block parameter.
`Simulink.ParamRTWInfo`	Specify information needed to generate code for a parameter.
`Simulink.RunTimeBlock`	Allow Level-2 M-file S-function and other M-file programs to get information about a block while a simulation running.
`Simulink.ScopeDataLogs`	Log data displayed by a Scope viewer.
`Simulink.Signal`	Describes the value of a block output.
`Simulink.StructElement`	Describes an element of a data structure.
`Simulink.StructType`	Describes a data structure.
`Simulink.SubsysDataLogs`	Stores a subsystem's signal logs.

Class	Purpose
`Simulink.TimeInfo`	Provide information about the time data in a `Simulink.Timeseries` object.
`Simulink.Timeseries`	Log for an elementary signal.
`Simulink.TsArray`	Log for a composite signal.

Classes — Alphabetical List

EventData
Simulink.AliasType
Simulink.Annotation
Simulink.BlockCompDworkData
Simulink.BlockCompInputPortData
Simulink.BlockCompOutputPortData
Simulink.BlockData
Simulink.BlockPortData
Simulink.BlockPreCompInputPortData
Simulink.BlockPreCompOutputPortData
Simulink.Bus
Simulink.BusElement
Simulink.ConfigSet
Simulink.ModelDataLogs
Simulink.ModelWorkspace
Simulink.MSFcnRunTimeBlock
Simulink.NumericType
Simulink.Parameter
Simulink.ParamRTWInfo
Simulink.RunTimeBlock
Simulink.ScopeDataLogs
Simulink.Signal
Simulink.StructElement
Simulink.StructType
Simulink.SubsysDataLogs
Simulink.TimeInfo
Simulink.Timeseries
Simulink.TsArray

EventData

Purpose

Provides information about block method execution events.

Description

Simulink creates an instance of this class when a block method execution event occurs during simulation and passes it to any listeners registered for the event (see `add_exec_event_listener`). The instance specifies the type of event that occurred and the block whose method execution triggered the event. See "Accessing Block Data During Simulation" in "Using Simulink®" for more information.

Parent

None

Children

None

Property Summary

Name	Description
"Type"	Type of method execution event that occurred.
"Source"	Block that triggered the event.

Properties

`Type`

Description

Type of method execution event that occurred. Possible values are:

Event	Occurs...
`'PreOutputs'`	Before a block's Outputs method executes.
`'PostOutputs'`	After a block's Outputs method executes.
`'PreUpdate'`	Before a block's Update method executes.
`'PostUpdate'`	After a block's Update method executes.
`'PreDerivatives'`	Before a block's Derivatives method executes.
`'PostDerivatives'`	After a block's Derivatives method executes.

Data Type
string

Access
RO

Source

Description
Block that triggered the event

Data Type
Simulink.RunTimeBlock

Access
RO

Simulink.AliasType

Purpose

Create an alias for a signal and/or parameter data type

Description

This class allows you to designate MATLAB variables as aliases for signal and parameter data types. You do this by creating instances of this class and assigning them to variables in the MATLAB or model workspaces (see "Creating a Data Type Alias" on page 9-8). The MATLAB variable to which a `Simulink.AliasType` object is assigned is called a data type alias. The data type to which an alias refers is called its base type. Simulink allows you to set the `BaseType` property of the object that the variable references, thereby designating the data type for which it is an alias.

Simulink lets you use aliases instead of actual type names in dialog boxes and `set_param` commands to specify the data types of Simulink block outputs and parameters. Using aliases to specify signal and parameter data types can greatly simplify global changes to the signal and parameter data types that a model specifies. In particular, changing the data type of all signals and parameters whose data type is specified by an alias requires only changing the base type of the alias. By contrast, changing the data types of signals and parameters whose data types are specified by an actual type name requires respecifying the data type of each signal and parameter individually, a tedious and time-consuming process.

Creating a Data Type Alias

You can use either the Model Explorer or MATLAB commands (see "MATLAB Commands for Creating Data Type Aliases" on page 9-9) to create a data type alias.

To use the Model Explorer to create an alias:

1. Select the MATLAB workspace in the Model Explorer's **Model Hierarchy** pane.

   You must create data type aliases in the MATLAB workspace. If you attempt to create an alias in a model workspace, Simulink displays an error.

2 Select **Simulink.AliasType** from the Model Explorer's **Add** menu.

   Simulink creates an instance of a `Simulink.AliasType` object and assigns it to a variable named `Alias` in the MATLAB workspace.

3 Rename the variable to a more appropriate name, for example, a name that reflects its intended usage.

   To change the name, edit the name displayed in the **Name** field in the Model Explorer's **Contents** pane.

4 Enter the name of the data type that this alias represents in the **Base type** field in the Model Explorer's **Dialog** pane.

   You can specify the name of any existing standard or user-defined data type in this field. Skip this step if the desired base type is `double` (the default).

5 Save the newly created alias in a MAT-file loaded by the models in which it is used.

MATLAB Commands for Creating Data Type Aliases

Use the following syntax to create a data type alias at the MATLAB command line or in a MATLAB program

```
ALIAS = Simulink.AliasType;
```

where `ALIAS` is the name of the variable that you want to serve as the alias. For example, the following line creates an alias names `MyFloat`.

```
MyFloat = Simulink.AliasType;
```

The following notations get and set the properties of a data type alias, respectively,

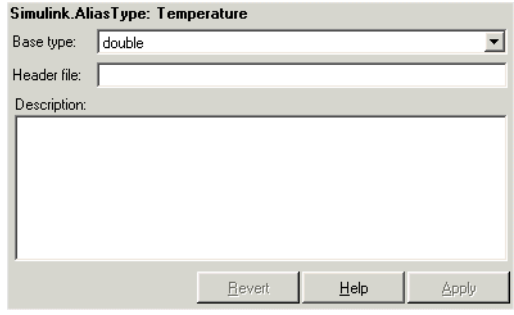
```
PROPVALUE = ALIAS.PROPNAME;
ALIAS.PROPNAME = PROPVALUE;
```

where ALIAS is the name of the alias, PROPNAME is the name of the alias object's properties, and PROPVALUE is the property's value. For example, the following code saves the current value of MyFloat's BaseType property and assigns it a new value.

```
save = MyFloat.BaseType;
MyFloat.BaseType = 'single';
```

See "Properties" on page 9-11 for information on the names, permitted values, and usage of the properties of data type alias objects.

Property Dialog Box



Base type

The data type to which this alias refers. The default is double. To specify another data type, select the data type from the adjacent pull-down list of standard data types or enter the data type's name in the edit field. Note that you can, with one exception, specify a nonstandard data type, e.g, a data type defined by a Simulink.NumericType object, by entering the data type's name in the edit field. The exception is a Simulink.NumericType whose Category is Fixed-point: unspecified scaling.

Header file

Name of a user-supplied C header file that defines a data type having the same name as this alias (i.e., as the MATLAB variable that references this alias object). If this field is not empty, code generated from this model defines the alias type by including the

specified header file. If this field is empty, the generated code defines the alias type itself.

Description
Describes the usage of the data type referenced by this alias.

Properties

Name	Description
`BaseType`	A string specifying the name of a standard or custom data type. (**Base Type**)
`Description`	A string that describes the usage of the data type. May be a null string. (**Description**)
`HeaderFile`	A string that specifies the name of a C header file that defines a data type having the same name as the alias. (**Header File**)

Simulink.Annotation

Purpose

Specify properties of a model annotation

Description

Instances of this class specify the properties of annotations. You can use `getCallbackAnnotation` in an annotation callback function to get the `Simulink.Annotation` instance for the annotation associated with the callback function. You can use `find_system` and `get_param` to get the `Simulink.Annotation` instance associated with any annotation in a model. For example, the following code gets the annotation object for the first annotation in the currently selected model and turns on its drop shadow

```
ah = find_system(gcs, 'FindAll', 'on', 'type', 'annotation');
ao = get_param(ah(1), 'Object');
ao.DropShadow = 'on';
```

Parent

None

Children

None.

Property Summary

Property	Description
`Text`	String specifying text of annotation.
`FontName`	String specifying name of annotation's font. The default value, `'auto'`, specifies use of the model's preferred font.
`FontSize`	Integer specifying size of annotation's font in points. The default value, `-1`, specifies use of the model's preferred font size.
`FontWeight`	String specifying the angle of the annotation's font. The default value, `'auto'`, specifies use of the model's preferred font angle
`Selected`	String specifying whether this annotation is currently selected (`'on'`) or not selected (`'off'`)

Property	Description
`Position`	Two-element vector specifying the x-y coordinates of this annotation relative to the top, left corner of the block diagram, e.g., `[236 83]`.
`HorizontalAlignment`	String specifying the horizontal alignment of this annotation, e.g., `'center'`.
`VerticalAlignment`	String specifying the horizontal alignment of this annotation, e.g., `'middle'`.
`ForegroundColor`	String specifying foreground color of this annotation.
`BackgroundColor`	String specifying background color of this annotation.
`DropShadow`	String specifying whether to display a drop shadow. Options are `'on'` or `'off'`.
`TeXMode`	String specifying whether to render TeX markup. Options are `'on'` or `'off'`.
`LoadFcn`	String specifying M-code to be executed when the model containing this annotation is loaded. See “Annotation Callback Functions” in the online Simulink documentation.
`DeleteFcn`	String specifying M-code to be executed before deleting this annotation. See “Annotation Callback Functions” in the online Simulink documentation.
`UseDisplayTextAsClickCallback`	String specifying whether to use the contents of the `Text` property as this annotation’s click function. Options are `'on'` or `'off'`.
`UserData`	Any data that you want to associate with this annotation.

Simulink.BlockCompDworkData

Purpose

Provides postcompilation information about a block's Dwork vector.

Description

Simulink returns an instance of this class when an M-file program, e.g., a Level-2 M-file S-function, invokes the "Dwork" on page 9-82 method of a block's run-time object after the model containing the block has been compiled.

Parent

`Simulink.BlockData`

Children

None

Property Summary

Name	Description
"UsedAsDiscState"	True if this Dwork vector is being used to store the values of a block's discrete states.

Properties

`UsedAsDiscState`

Description
True if this Dwork vector is being used to store the values of a block's discrete states.

Data Type
Boolean

Access
`RW` for M-file S-function blocks, `RO` for other blocks.

Purpose

Provides postcompilation information about a block input port

Description

Simulink returns an instance of this class when an M-file program, e.g., a Level-2 M-file S-function, invokes the "InputPort" on page 9-83 method of a block's run-time object after the model containing the block has been compiled.

Parent

`Simulink.BlockPortData`

Children

None

Property Summary

Name	Description
"DirectFeedthrough"	True if this port has direct feedthrough.
"Overwritable"	True if this port is overwritable.

Properties

`DirectFeedthrough`

Description
True if this input port has direct feedthrough.

Data Type
Boolean

Access
`RW` for M-file S functions, `RO` for other blocks.

`Overwritable`

Description
True if this input port is overwritable.

Data Type
Boolean

Access
`RW` for M-file S functions, `RO` for other blocks.

Simulink.BlockCompOutputPortData

Purpose

Provides postcompilation information about a block output port.

Description

Simulink returns an instance of this class when an M-file program, e.g., a Level-2 M-file S-function, invokes the "OutputPort" on page 9-84 method of a block's run-time object after the model containing the block has been compiled.

Parent

`Simulink.BlockPortData`

Children

None

Property Summary

Name	Description
"Reusable"	True if this port has direct feedthrough.

Properties

`Reusable`

Description
Specifies whether an output port's memory is reusable. Options are: `NotReusableAndGlobal` and `ReusableAndLocal`.

Data Type
string

Access
`RW` for M-file S functions, `RO` for other blocks.

Purpose

Provide run-time information about block-related data, such as block parameters

Description

This class defines properties that are common to objects that provide run-time information about a block's ports and work vectors.

Parent

None

Children

`Simulink.BlockPortData`, `Simulink.BlockCompDworkData`

Property Summary

Name	Description
"Complexity"	Numeric type (real or complex) of the block data.
"Data"	The block data.
"DataAsDouble"	The block data in `double` form.
"Datatype"	Data type of the block data.
"DatatypeID"	Index of the data type of the block data.
"Dimensions"	Dimensions of the block data.
"Name"	Name of the block data.
"Type"	Type of block data (e.g., a parameter).

Properties

`Complexity`

Description
Numeric type (real or complex) of the block data.

Data Type
string

Access
`RW` for M-file S functions, `RO` for other blocks.

`Data`

Description
The block data.

Data Type
The data type specified by the “Datatype” or “DatatypeID” properties of this object.

Access
`RW`

`DataAsDouble`

Description
The block data’s in `double` form.

Data Type
`double`

Access
`RO`

`Datatype`

Description
Data type of the values of the block-related object.

Data Type
string

Access
`RO`

`DatatypeID`

Description
Index of the data type of the values of the block-related object.

Data Type
integer

Access
`RW` for M-file S functions, `RO` for other blocks

`Dimensions`

Description
Dimensions of the block-related object, e.g., parameter or DWork vector.

Data Type
array

Access
`RW` for M-file S functions, `RO` for other blocks

`Name`

Description
Name of block-related object, e.g., a block parameter or Dwork vector.

Data Type
string

Access
`RW` for M-file S functions, `RO` for other blocks

`Type`

Description
Type of block data. Possible values are:

Type	Description
`'BlockPreCompInputPortData'`	This object contains data for an input port before the model is compiled.
`'BlockPreCompOutputPortData'`	This object contains data for an output port before the model is compiled.

Type	Description
'BlockCompInputPortData'	This object contains data for an input port after the model is compiled.
'BlockCompOutputPortData'	This object contains data for an output port after the model is compiled.
'BlockPreCompDworkData'	This object contains data for a Dwork vector before the model is compiled.
'BlockCompDworkData'	This object contains data for a Dwork vector after the model is compiled.
'BlockDialogPrmData'	This object describes a dialog box parameter of a Level-2 M-file S-function.
'BlockRuntimePrmData'	This object describes a run-time parameter of a Level-2 M-file S-function.
'BlockCompContStatesData'	This object describes the continuous states of the block at the current time step.
'BlockDerivativesData'	This object describes the derivatives of the block's continuous states at the current time step.

Data Type
string

Access
RO

Purpose

Describe a block input or output port

Description

This class defines properties that are common to objects that provide run-time information about a block's ports.

Parent

`Simulink.BlockData`

Children

`Simulink.BlockPreCompInputPortData`,
`Simulink.BlockPreCompOutputPortData`,
`Simulink.BlockCompInputPortData`,
`Simulink.BlockCompOutputPortData`

Property Summary

Name	Description
"IsBus"	True if this port is connected to a bus.
"IsSampleHit"	True if this port produces output or accepts input at the current simulation time step.
"SampleTime"	Sample time of this port.
"SampleTimeIndex"	Sample time index of this port.
"SamplingMode"	Sampling mode of the port.

Properties

`IsBus`

Description
True if this port is connected to a bus.

Data Type
Boolean

Access
`RO`

`IsSampleHit`

Description
True if this port produces output or accepts input at the current simulation time step.

Data Type
Boolean

Access
`RO`

`SampleTime`

Description
Sample time of this port.

Data Type
`[period offset]` where `period` and `offset` are values of type `double`. See "Specifying Sample Time" for more information.

Access
`RW` for M-file S functions, `RO` for other blocks

`SampleTimeIndex`

Description
Sample time index of this port.

Data Type
integer

Access
`RO`

`SamplingMode`

Description
Sampling mode of the port. Valid values are:

Value	Description
`'frame'`	Port accepts or outputs frame-based signals. The use of frame-based signals requires a Signal Processing Blockset license.
`'inherited'`	Sampling mode is inherited from the port to which this port is connected.
`'sample'`	Port accepts or outputs sampled data.

Data Type
string

Access
`RW` for M-file S functions, `RO` for other blocks

Simulink.BlockPreCompInputPortData

Purpose

Provides precompilation information about a block input port

Description

Simulink returns an instance of this class when an M-file program, e.g., a Level-2 M-file S-function, invokes the "InputPort" on page 9-83 method of a block's run-time object before the model containing the block has been compiled.

Parent

`Simulink.BlockPortData`

Children

None

Property Summary

Name	Description
"DirectFeedthrough"	True if this port has direct feedthrough.
"Overwritable"	True if this port is overwritable.

Properties

`DirectFeedthrough`

Description
True if this input port has direct feedthrough.

Data Type
Boolean

Access
`RW` for M-file S functions, `RO` for other blocks

`Overwritable`

Description
True if this input port is overwritable.

Data Type
Boolean

Access
`RW` for M-file S functions, `RO` for other blocks

Purpose

Provide precompilation information about a block output port.

Description

Simulink returns an instance of this class when an M-file program, e.g., a Level-2 M-file S-function, invokes the "OutputPort" on page 9-84 method of a block's run-time object before the model containing the block has been compiled.

Parent

`Simulink.BlockPortData`

Children

none

Property Summary

Name	Description
"Reusable"	True if this port has direct feedthrough.

Properties

`Reusable`

Description
Specifies whether an output port's memory is reusable. Options are: `NotReusableAndGlobal` and `ReusableAndLocal`.

Data Type
string

Access
`RW` for M-file S functions, `RO` for other blocks

Simulink.Bus

Purpose

Specify the properties of a signal bus

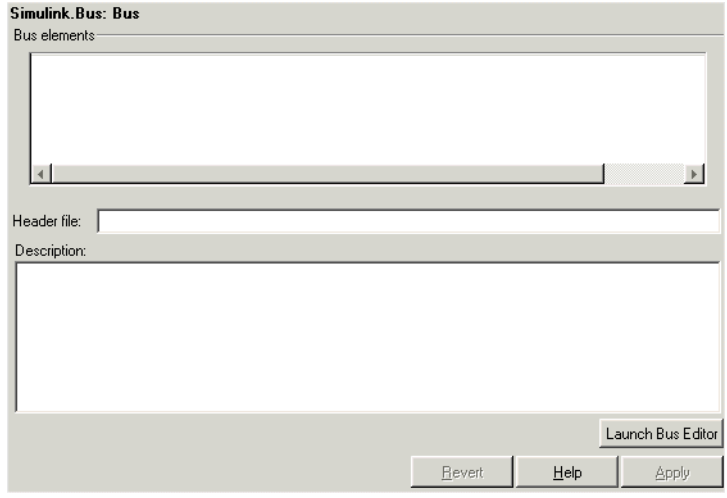
Description

Objects of this class (in conjunction with objects of the `Simulink.BusElement` class) specify the properties of a signal bus. You can use these objects to enable Simulink to check the validity of buses connected to the inputs of blocks in your model. You do this by entering the name of a bus object defining a bus in the **Bus object** field of a block's parameter dialog box. When you update the model's diagram or start a simulation of the model, Simulink checks whether the buses connected to the blocks have the properties specified by the bus objects. If not, Simulink halts and displays an error message.

You can use the Model Explorer (see "The Model Explorer" in "Using Simulink®"), the Simulink Bus Editor (see "Bus Editor"), or MATLAB commands (see "Working with Data Objects") to create bus objects. You must use the Bus Editor or the MATLAB command line to set the properties of a bus object. Simulink also provides a set of utility functions for creating and saving bus objects. See the documentation for the following functions for more information:

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

Property Dialog Box



Bus elements

Table that displays the properties of the bus's elements. You cannot edit this table. You must use either the Simulink **Bus Editor** (see "Bus Editor" in "Using Simulink®") or MATLAB commands to add or delete bus elements or change the properties of existing bus elements. To launch the bus editor, click the **Launch Bus Editor** button at the bottom of this dialog box or select **Bus Editor** from the model editor's **Tools** menu.

Header file

Name of a C header file that declares the structure of this bus. This field is intended for use by Real-Time Workshop (see "Code Generation with User-Defined Data Types" "Real-Time Workshop Embedded Coder User's Guide"). Simulink ignores this field.

Description

Description of this structure. This field is intended for you to use to document this bus. Simulink itself does not use this field.

Properties

Name	Access	Description
`Description`	`RW`	String that describes this bus. This property is intended for user use. Simulink itself does not use it. (Description)
`Elements`	`RW`	An array of `Simulink.BusElement` objects that define the names, data types, dimensions, and other properties of the bus's elements. The elements must have unique names. (Bus elements)
`HeaderFile`	`RW`	String that specifies the name of a C header file that declares the structure of this bus. This property is intended for use by Real-Time Workshop. Simulink does not use it. (Header file)

See Also

`Simulink.BusElement`

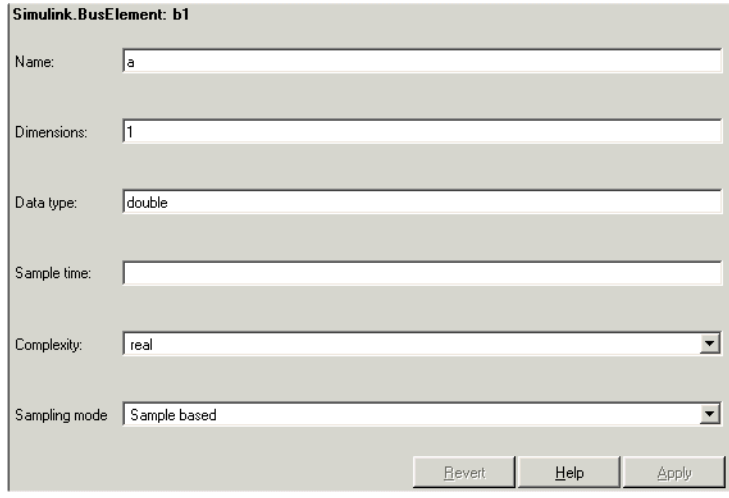
Purpose

Describe an element of a signal bus

Description

Objects of this class describe elements of buses described by objects of the `Simulink.Bus` class.

Property Dialog Box



Name

Name of the element.

Dimensions

A vector specifying the dimensions of this element. Must be -1 if this element is itself a bus.

Data type

Name of the data type of this element. The value of this field can be the name of a

- ,built-in Simulink data type, e.g., `double` or `uint8`
- `Simulink.NumericType` object, with one exception. The exception is a `Simulink.NumericType` whose `Category` is `Fixed-point:  unspecified scaling`.

- `Simulink.Bus` object. This allows you to create bus objects that specify hierarchical buses, i.e., buses that contain other buses.

Sample time
Size of the interval between times when this signal's value must be recomputed. Must be -1 (inherited) if this bus element is itself a bus or if the bus that includes this element passes through a block that changes the bus's sample time, such as a Rate Transition block. See "Specifying Sample Time" for more information.

Complexity
Numeric type (i.e., real or complex) of this element. Must be real if this bus element is itself a bus.

Sampling mode
Sampling mode of this element. Must be sample-based if this element is itself a bus. This field is intended to be used by applications based on Simulink.

Properties

Name	Access	Description
`Complexity`	`RW`	String that specifies the numeric type (`'real'` or `'complex'`) of this element. (Complexity)
`DataType`	`RW`	String that specifies the name of the data type of this element or the name of a bus object to specify an element that is itself a bus. (Data type)
`Dimensions`	`RW`	A vector specifying the dimensions of this element. (Dimensions)
`Name`	`RW`	String specifying the name of this element. (link linkend="bqb37qu">Name)

Name	Access	Description
SampleTime	RW	Scalar or vector that specifies the sample time of this bus element. (Sample time)
SamplingMode	RW	String that specifies the sampling mode of this element. (Sampling mode)

See Also

Simulink.Bus

Simulink.ConfigSet

Purpose

Access a model configuration set.

Description

Instances of this handle class allow you to write programs to create, modify, and attach configuration sets to models. See “Configuration Set API” for more information.

Property Summary

Name	Description
“Components”	Components of the configuration set.
“Description”	Description of the configuration set.
“Name”	Name of the configuration set.
“SimulationMode”	Mode used to simulation this configuration.

Note You can use the **Model Configuration** dialog box to set the `Name`, `Description`, and `SimulationMode` properties of a model’s active configuration set. See “Model Configuration Dialog Box” in “Using Simulink®” for more information.

Method Summary

Name	Description
“attachComponent”	Attach a component to a configuration set.
“copy”	Create a copy of a configuration set.
“getComponent”	Get a component of a configuration set.
“getFullName”	Get the full pathname of a configuration set.
“getModel”	Get the handle of the model that owns a configuration set.
“get_param”	Get the value of a configuration set parameter.

Name	Description
“isActive”	Determine whether a configuration set is the active set of the model that owns it.
“isValidParam”	Determine whether a specified parameter is a valid parameter of a configuration set.
“setPropEnabled”	Prevent or allow a user to change a parameter.
“set_param”	Set the value of a configuration set parameter.

Properties

`Components`

Description
Array of `Simulink.ConfigComponent` objects representing the components of the configuration set, e.g., solver parameters, data import/export parameters, etc.

Data Type
array

Access
`RW`

`Description`

Description
Description of the configuration set. You can use this property to provide additional information about a configuration set, such as its purpose.

Data Type
`string`

Access
`RW`

`Name`

Description
Configuration set’s name.

Data Type
`string`

Access
`RW`

`SimulationMode`

Description
Model's simulation mode. Valid values are `normal`, `accelerator`, or `external`.

Data Type
`string`

Access
`RW`

Methods

`attachComponent`

Purpose
Attach a component to this configuration set.

Syntax
`attachComponent(component)`

Arguments

`component`
Instance of `Simulink.ConfigComponent` class.

Description
This method replaces a component in this configuration set with a component having the same name.

Example
The following example replaces the solver component of the active configuration set of model A with the solver component of the active configuration set of model B.

```
hCs = getActiveConfigSet('B');
```

```
hSolverConfig = hCs.getComponent('Solver');
hSolverConfig = hSolverConfig.copy;
hCs = getActiveConfigSet('A');
hCs.attachComponent(hSolverConfig);
```

`copy`

Purpose

Create a copy of this configuration set.

Syntax

```
copy
```

Description

This method creates a copy of this configuration set.

Note You must use this method to create copies of configuration sets. This is because `Simulink.ConfigSet` is a handle class. See "Handle Versus Value Classes" in "Using Simulink®" for more information.

`getComponent`

Purpose

Get a component of this configuration set.

Syntax

```
getComponent(componentName)
```

Arguments

`componentName`

String specifying the name of the component to be returned.

Description

Returns the specified component. Omit the argument to get a list of the names of the components that this configuration set contains.

Example

The following code gets the solver component of the active configuration set of the currently selected model.

```
hCs = getActiveConfigSet(gcs);
hSolverConfig = hCs.getComponent('Solver');
```

The following code displays the names of the components of the currently selected model's active configuration set at the MATLAB command line.

```
hCs = getActiveConfigSet(gcs);
hCs.getComponent
```

`getFullName`

Purpose
Get the full path name of a configuration set.

Syntax
`getFullName`

Description
This method returns a string specifying the full pathname of a configuration set, e.g., `'vdp/Configuration'`.

`getModel`

Purpose
Get the model that owns this configuration set.

Syntax
`getModel`

Description
Returns a handle to the model that owns this configuration set.

Example
The following command opens the block diagram of the model that owns the configuration set referenced by the MATLAB workspace variable `hCs`.

```
open_system(hCs.getModel);
```

`get_param`

Purpose

Get the value of a configuration set parameter.

Syntax

`get_param(paramName)`

Arguments

`paramName`

String specifying the name of the parameter whose value is to be returned.

Description

This method returns the value of the specified parameter. `config.get_param('ObjectParameters')` returns the names of the valid parameters in the configuration set named `config`.

Example

The following command gets the name of the solver used by the selected model's active configuration.

```
hAcs = getActiveConfigSet(bdroot);
hAcs.get_param('SolverName');
```

Note You can also use the `get_param` model construction command to get the values of parameters of a model's active configuration set, e.g., `get_param(bdroot, 'SolverName')` gets the solver name of the currently selected model.

`isActive`

Purpose

Determine whether this configuration set is its model's active configuration set.

Syntax
`isActive`

Description
Returns `true` if this configuration set is the active configuration set of the model that owns this configuration set.

`isValidParam`

Purpose
Determine whether a specified parameter is a valid parameter of this configuration set. A parameter is valid if it is compatible with other parameters in the configuration set. For example, if SolverType is set to `'variable-step'`, FixedStep is an invalid parameter.

Syntax
`isValidParam(paramName)`

Arguments

`paramName`
String specifying the name of the parameter whose validity is to be determined.

Description
This method returns true if the specified parameter is a valid parameter of this configuration set; otherwise, it returns false.

Example
The following code sets the parameter StopTime only if it is a valid parameter of the currently selected model's active configuration set.

```
hAcs = getActiveConfigSet(gcs);
if hAcs.isValidParam('StopTime')
 set_param('StopTime', '20');
end
```

setPropEnabled

Purpose
Enable a configuration set parameter to be changed.

Syntax
```
setPropEnabled(paramName, isEnabled)
```

Arguments

`paramName`
: Name of the parameter whose value is to be set.

`isEnabled`
: Specify as `true` to enable parameter; as `false`, to disable the parameter.

Description
This method sets the enabled status the parameter specified by `paramName` to the value specified by `isEnabled`. Disabling a parameter prevents the user from changing it.

Example
The following code prevents the user from setting the simulation stop time of the currently selected model.

```
hAcs = getActiveConfigSet(gcs);
hAcs.setPropEnabled('StopTime', false);
```

set_param

Purpose
Set the value of a configuration set parameter.

Syntax
```
set_param(paramName, paramValue)
```

Arguments

`paramName`
: Name of the parameter whose value is to be set.

`paramValue`
: Value to assign to the parameter.

Description

This method sets the configuration set parameter specified by `paramName` to the value specified by `paramValue`.

Example

The following command sets the simulation stop time of the selected model's active configuration.

```
hAcs = getActiveConfigSet(gcs);
hAcs.set_param('StopTime', '20');
```

Note You can also use the `set_param` model construction command to set the parameters of the active configuration set, e.g., `set_param(gcs, 'StopTime', '20')` sets the simulation stop time of the currently selected model.

Purpose

Container for a model's signal data logs

Description

Simulink creates instances of this class to contain signal logs that it creates while simulating a model (see "Logging Signals"). In particular, Simulink creates an instance of this class for a top-level model and for each model referenced by the top-level model that contains signals to be logged. Simulink assigns the `ModelDataLogs` object for the top-level model to a variable in the MATLAB workspace. The name of the variable is the name specified in the **Signal logging name** field on the **Data Import/Export** pane of the model's **Configuration Parameters** dialog box. The default value is `logsout`.

A `ModelDataLogs` object has a variable number of properties. The first property, named `Name`, specifies the name of the model whose signal data the object contains or, if the model is a referenced model, the name of the Model block that references the model. The remaining properties reference objects that contain signal data logged during simulation of the model. The objects may be instances of any of the following types of objects:

- `Simulink.Timeseries`

  Log for a signal in this model.

- `Simulink.TsArray`

  Container for the logs of the elements of a root-level composite signal (e.g., a Mux or Bus Creator signal) in this model.

- `Simulink.ModelDataLogs`

  Container for the logs of a model referenced by this model.

- `Simulink.SubsysDataLogs`

  Container for the signal logs of a subsystem of this model.

- `Simulink.ScopeDataLogs`

  Container for data displayed on Scope signal viewers (see "The Signal & Scope Manager" in "Using Simulink®").

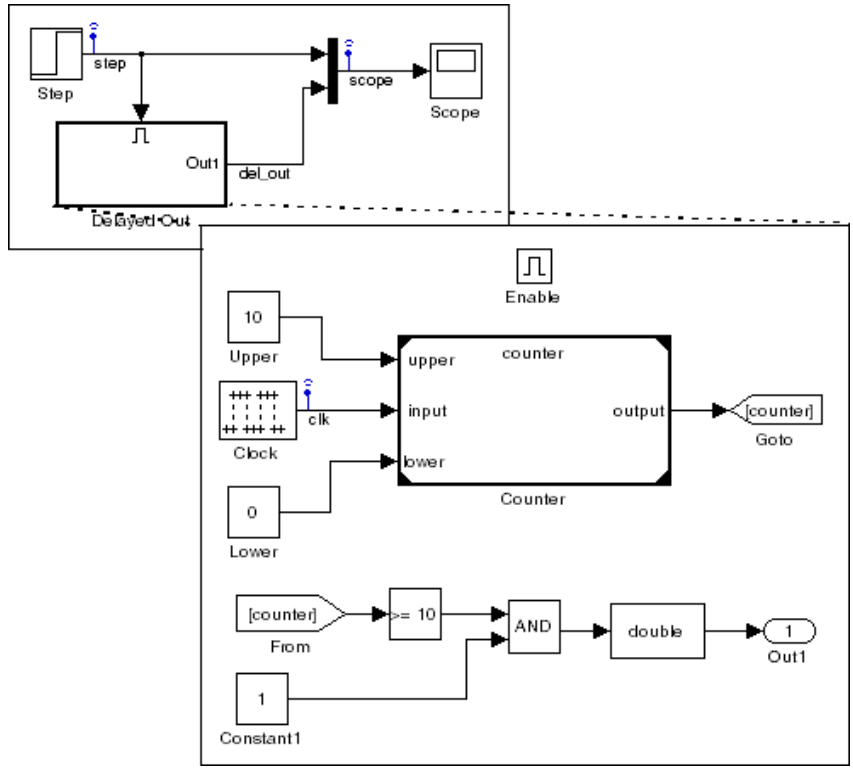
The names of the properties identify the data being logged as follows:

- For signal data logs, the name of the signal
- For a subsystem or model log container, the name of the subsystem or model, respectively
- For a scope viewer data log, the name specified on the viewer's parameter dialog box

---

Note If a name contains spaces, the `ModelDataLogs` objects specifies its name as `('name')` where **`name`** is the actual name, e.g., `('Brake Subsystem')`.

Consider, for example, the following model.



As indicated by the testpoint icons, this model specifies that Simulink should log the signals named `step` and `scope` in the model's root system and the signal named `clk` in the subsystem named Delayed Out. After simulation of this model, the MATLAB workspace contains the following variable:

```
>> logsout

logsout =

Simulink.ModelDataLogs (siglgex):
  Name                    Elements  Simulink Class
```

```
scope                    2      TsArray
step                     1      Timeseries
('Delayed Out')          2      SubsysDataLogs
```

The `logsout` variable contains the signal data logged during the simulation. You can use fully qualified object names or the Simulink `unpack` command to access the signal data stored in `logsout`. For example, to access the amplitudes of the `clk` signal in the Delayed Out subsystem, enter

```
>> data = logsout.('Delayed Out').clk;
```

or

```
>> logsout.unpack('all');
>> data = clk;
```

Note, you can use a custom logging name or signal name when logging a signal. If you choose to use the signal name, and that signal name is a multiline one, as shown in the following:

```
(fixPt)
temp (degC)
```

include an `sprintf('\n')` between the two lines of the signal name when accessing the logged data. For example,

```
logsout.('Boiler Plant model').('digital thermometer').\
(['(fixPt)' sprintf('\n') 'temp (degC)'])
```

See Also

`Simulink.Timeseries`, `Simulink.TsArray`, `Simulink.SubsysDataLogs`, `Simulink.ScopeDataLogs`, `unpack`

Purpose

Describe a model workspace.

Description

Instances of this class describe model workspaces. Simulink creates an instance of this class for each model that you open during a Simulink session. See “Working with Model Workspaces” in “Using Simulink®”for more information.

Property Summary

Name	Access	Description
`DataSource`	`RW`	Specifies the source used to initialize this workspace. Valid values are • `'MDL-File'` • `'MAT-File'` • `'M-Code'`
`FileName`	`RW`	Specifies the name of the MAT-file used to initialize this workspace. Simulink ignores this property if `DataSource` is not `'MAT-File'`.
`MCode`	`RW`	A string specifying M code used to initialize this workspace. Simulink ignores this property if `DataSource` is not `'M-Code'`.

Method Summary

Name	Description
“assignin”	Assign a value to a variable in the model’s workspace.
“clear”	Clear the model’s workspace.
“evalin”	Evaluate an expression in the model’s workspace.

Name	Description
“reload”	Reload the model workspace from the workspace’s data source.
“save”	Save the model’s workspace to a specified MAT-file.
“saveToSource”	Save the workspace to the MAT-file that the workspace designates as its data source.
“whos”	List the variables in the model workspace.

Methods

`assignin`

Purpose
Assign a value to a variable in the model’s workspace.

Syntax
```
assignin('varname', varvalue)
```

Arguments

`varname`
Name of the variable to be assigned a value.

`varvalue`
Value to be assigned the variable.

Description
This method assigns the value specified by `varvalue` to the variable whose name is `varname`.

See also
“evalin”

`clear`

Purpose
Clear the model’s workspace.

Syntax
```
clear
```

Description
This method empties the workspace of its variables.

`evalin`

Purpose
Evaluate an expression in the model's workspace.

Syntax
```
evalin('expression')
```

Arguments

`expression`
A MATLAB expression to be evaluated.

Description
This method evaluates expression in the model workspace.

See also
"assignin"

`reload`

Purpose
Reload the model workspace from the workspace's data source.

Syntax
```
reload
```

Description
This method reloads the model workspace from the data source specified by its DataSource parameter.

See also
"saveToSource"

`save`

Purpose

Save the model's workspace to a specified MAT-file.

Syntax

```
save('filename')
```

Arguments

`filename`

Name of a MAT-file.

Description

This method saves the model's workspace to the MAT-file specified by `filename`.

Note This method allows you to save the workspace to a file other than the file specified by the workspace's `FileName` property. If you want to save the model workspace to the file specified by the file's `FileName` property, it is simpler to use the workspace's `saveToSource` method.

Example

```
hws = get_param('mymodel','modelworkspace')
hws.DataSource = 'MAT-File';
hws.FileName = 'workspace';
hws.assignin('roll', 30);
hws.saveToSource;
hws.assignin('roll', 40);
hws.save('workspace_test.mat');
```

See also

"reload", "saveToSource"

`saveToSource`

Purpose
Save the workspace to the MAT-file that it designates as its data source.

Syntax
`saveToSource`

Description
This method saves the model workspace designated by its `FileName` property.

Example

```
hws = get_param('mymodel','modelworkspace')
hws.DataSource = 'MAT-File';
hws.FileName = 'params';
hws.assignin('roll', 30);
hws.saveToSource;
```

See also
“save”, “reload”

`whos`

Purpose
List the variables in the model workspace.

Syntax
`whos`

Description
This method lists the variables in the model's workspace. The listing includes the size and class of the variables.

Example

```
>> hws = get_param('mymodel','modelworkspace');
>> hws.assignin('k', 2);
>> hws.whos
```

```
  Name      Size                     Bytes  Class

  k         1x1                          8  double array
```

Purpose

Get run-time information about a Level-2 M-file S-function block

Description

This class allows a Level-2 M-file S-function or other M program to obtain information from Simulink and provide information to Simulink about a Level-2 M-file S-function block. Simulink creates an instance of this class for each Level-2 M-file S-function block in a model. Simulink passes the object to the callback methods of Level-2 M-File S-Functions when it updates or simulates a model, allowing the callback methods to get and provide block-related information to Simulink. See "Writing Level-2 M-File S-Functions" in "Writing S-Functions" for more information.

You can also use instances of this class in M-file programs to obtain information about Level-2 M-File S-Function blocks during a simulation. See "Accessing Block Data During Simulation" in "Using Simulink®" for more information.

Parent Class

`Simulink.RunTimeBlock`

Derived Classes

None

Property Summary

Name	Description
"DialogPrmsTunable"	Specifies which of the S-function's dialog parameters are tunable.
"NextTimeHit"	Time of the next sample hit for variable sample time S-functions.

Method Summary

Name	Description
“AutoRegRuntimePrms”	Register this block’s dialog parameters as run-time parameters.
“AutoUpdateRuntimePrms”	Update this block’s run-time parameters.
“IsDoingConstantOutput”	Determine whether the current simulation stage is the constant sample time stage.
“IsMajorTimeStep”	Determine whether the current simulation time step is a major time step.
“IsSampleHit”	Determine whether the current simulation time is one at which a task handled by this block is active.
“IsSpecialSampleHit”	Determine whether the current simulation time is one at which multiple tasks handled by this block are active.
“IsSampleHit”	Register a callback method for this block.
“RegisterDataTypeFxpBinaryPoint”	Register fixed-point data type with binary point-only scaling.

Name	Description
"RegisterDataTypeFxpFSlopeFixExpBias"	Register fixed-point data type with [Slope Bias] scaling specified in terms of fractional slope, fixed exponent, and bias.
"RegisterDataTypeFxpSlopeBias"	Register data type with [Slope Bias] scaling.
"RegisterDataTypeFxpBinaryPoint"	Specify whether to use this block's TLC file to generate the simulation target for the model that uses it.
"SetPreCompPortInfoToDefaults"	Set compiled attributes of this block's ports to default values.
"SetPreCompPortInfoToDynamic"	Set precompiled attributes of this block's ports to be inherited.
"SetSimViewingDevice"	Specify whether block is a viewer.
"WriteRTWParam"	Write custom parameter information to Real-Time Workshop file.

Properties

`DialogPrmsTunable`

Description

Specifies whether a dialog parameter of the S-function is tunable. Tunable parameters are registered as run-time parameters when you

call the “AutoRegRuntimePrms” method. Note that SimOnlyTunable parameters are not registered as run-time parameters.

Data Type
array

Access
RW

`NextTimeHit`

Description
Time of the next sample hit for variable sample-time S-functions.

Data Type
double

Access
RW

Methods

`AutoRegRuntimePrms`

Purpose
Register a block’s tunable dialog parameters as run-time parameters.

Syntax
```
AutoRegRuntimePrms;
```

Description
Register this block’s tunable dialog parameters as run-time parameters.

`AutoUpdateRuntimePrms`

Purpose
Update a block’s run-time parameters.

Syntax
```
AutoRegRuntimePrms;
```

Description
Automatically update the values of the run-time parameters during a call to `mdlProcessParameters`.

`IsDoingConstantOutput`

Purpose

Determine whether this is in the constant sample time stage of a simulation.

Syntax

```
bVal = IsDoingConstantOutput;
```

Description

Returns true if this is the constant sample time stage of a simulation, i.e., the stage at the beginning of a simulation where Simulink computes the values of block outputs that cannot change during the simulation (see "Constant Sample Time" in "Using Simulink®"). Use this method in the `mdlOutputs` method of an S-function with port-based sample times to avoid unnecessarily computing the outputs of ports that have constant sample time, i.e., `[inf, 0]`.

```
function Outputs(block)
.
.
  if block.IsDoingConstantOutput
    ts = block.OutputPort(1).SampleTime;
    if ts(1) == Inf
    %% Compute port's output.
    end
  end
.
.
%% end of Outputs
```

See "Specifying Port-Based Sample Times" in "Writing S-Functions" for more information.

`IsMajorTimeStep`

Purpose.

Determine whether current time step is a major or a minor time step.

Syntax

```
bVal = IsMajorTimeStep;
```

Description

Returns true if the current time step is a major time step; false, if it is a minor time step. This method can be called only from `mdlOutputs` and `mdlUpdate`.

`IsSampleHit`

Purpose

Determine whether the current simulation time is one at which a task handled by this block is active.

Syntax

```
bVal = IsSampleHit(stIdx);
```

Arguments

`stIdx`

Index of sample time to be queried.

Description

Use in Outputs or Update block methods when the M-file S-function has multiple sample times to determine whether a sample hit has occurred at `stIdx` (similar to `ssIsSampleHit` for C-Mex S-functions.)

`IsSpecialSampleHit`

Purpose

Determine whether the current simulation time is one at which multiple tasks implemented by this block are active.

Syntax

```
bVal = IsSpecialSampleHit(stIdx1,stIdx1);
```

Arguments

`stIdx1`

Index of sample time of first task to be queried.

`stIdx2`
Index of sample time of second task to be queried.

Description
Use in Outputs or Update block methods to ensure the validity of data shared by multiple tasks running at different rates. Returns true if a sample hit has occurred at `stIdx1` and a sample hit has also occurred at `stIdx2` in the same time step (similar to `ssIsSpecialSampleHit` for C-Mex S-functions).

`RegBlockMethod`

Purpose
Register a block callback method.

Syntax
```
RegBlockMethod(methName, methHandle);
```

Arguments

`methName`
Name of method to be registered.

`methHandle`
MATLAB function handle of the callback method to be registered.

Description
Registers the block callback method specified by `methName` and `methHandle`. Use this method in a Level-2 M-file S-function to specify the block callback methods that the S-function implements.

`RegisterDataTypeFxpBinaryPoint`

Purpose
Register fixed-point data type with binary point-only scaling.

Syntax
```
dtID = RegisterDataTypeFxpBinaryPoint(isSigned,
wordLength, fractionalSlope, fixedExponent, bias,
obeyDataTypeOverride);
```

Arguments

`isSigned`
: `true` if the data type is signed.

  `false` if the data type is unsigned.

`wordLength`
: Total number of bits in the data type, including any sign bit.

`fractionalLength`
: Number of bits in the data type to the right of the binary point.

`obeyDataTypeOverride`
: `true` indicates that the **Data Type Override** setting for the subsystem is to be obeyed. Depending on the value of **Data Type Override**, the resulting data type could be `True Doubles`, `True Singles`, `ScaledDouble`, or the fixed-point data type specified by the other arguments of the function.

  `false` indicates that the **Data Type Override** setting is to be ignored.

Description

This method registers a fixed-point data type with Simulink and returns a data type ID. The data type ID can be used to specify the data types of input and output ports, run-time parameters, and DWork states. It can also be used with all the standard data type access methods defined for instances of this class, such as “DatatypeSize”.

Use this function if you want to register a fixed-point data type with binary point-only scaling. Alternatively, you can use one of the other fixed-point registration functions:

- Use “RegisterDataTypeFxpFSlopeFixExpBias” to register a data type with [Slope Bias] scaling by specifying the word length, fractional slope, fixed exponent, and bias.
- Use “RegisterDataTypeFxpSlopeBias” to register a data type with [Slope Bias] scaling.

If the registered data type is not one of the Simulink built-in data types, a Simulink Fixed Point license is checked out.

`RegisterDataTypeFxpFSlopeFixExpBias`

Purpose

Register fixed-point data type with [Slope Bias] scaling specified in terms of fractional slope, fixed exponent, and bias

Syntax

```
dtID = RegisterDataTypeFxpFSlopeFixExpBias(isSigned,
wordLength, fractionalSlope, fixedExponent, bias,
obeyDataTypeOverride);
```

Arguments

`isSigned`
: `true` if the data type is signed.

  `false` if the data type is unsigned.

`wordLength`
: Total number of bits in the data type, including any sign bit.

`fractionalSlope`
: Fractional slope of the data type.

`fixedExponent`
: Exponent of the slope of the data type.

`bias`
: Bias of the scaling of the data type.

`obeyDataTypeOverride`
: `true` indicates that the **Data Type Override** setting for the subsystem is to be obeyed. Depending on the value of **Data Type Override**, the resulting data type could be `True Doubles`, `True Singles`, `ScaledDouble`, or the fixed-point data type specified by the other arguments of the function.

`false` indicates that the **Data Type Override** setting is to be ignored.

Description

This method registers a fixed-point data type with Simulink and returns a data type ID. The data type ID can be used to specify the data types of input and output ports, run-time parameters, and DWork states. It can also be used with all the standard data type access methods defined for instances of this class, such as "DatatypeSize".

Use this function if you want to register a fixed-point data type by specifying the word length, fractional slope, fixed exponent, and bias. Alternatively, you can use one of the other fixed-point registration functions:

- Use "RegisterDataTypeFxpBinaryPoint" to register a data type with binary point-only scaling.
- Use "RegisterDataTypeFxpSlopeBias" to register a data type with [Slope Bias] scaling.

If the registered data type is not one of the Simulink built-in data types, a Simulink Fixed Point license is checked out.

`RegisterDataTypeFxpSlopeBias`

Purpose

Register data type with [Slope Bias] scaling.

Syntax

```
dtID = RegisterDataTypeFxpSlopeBias(isSigned, wordLength,
totalSlope, bias, obeyDataTypeOverride);
```

Arguments

`isSigned`

`true` if the data type is signed.

`false` if the data type is unsigned.

`wordLength`
Total number of bits in the data type, including any sign bit.

`totalSlope`
Total slope of the scaling of the data type.

`bias`
Bias of the scaling of the data type.

`obeyDataTypeOverride`
`true` indicates that the **Data Type Override** setting for the subsystem is to be obeyed. Depending on the value of **Data Type Override**, the resulting data type could be `True Doubles`, `True Singles`, `ScaledDouble`, or the fixed-point data type specified by the other arguments of the function.

`false` indicates that the **Data Type Override** setting is to be ignored.

Description

This method registers a fixed-point data type with Simulink and returns a data type ID. The data type ID can be used to specify the data types of input and output ports, run-time parameters, and DWork states. It can also be used with all the standard data type access methods defined for instances of this class, such as "DatatypeSize" on page 9-81.

Use this function if you want to register a fixed-point data type with [Slope Bias] scaling. Alternatively, you can use one of the other fixed-point registration functions:

- Use "RegisterDataTypeFxpBinaryPoint" to register a data type with binary point-only scaling.
- Use "RegisterDataTypeFxpFSlopeFixExpBias" to register a data type by specifying the word length, fractional slope, fixed exponent, and bias

If the registered data type is not one of the Simulink built-in data types, a Simulink Fixed Point license is checked out.

`SetAccelRunOnTLC`

Purpose
Specify whether to use block's TLC file to generate code for the Simulink accelerator.

Syntax
```
SetAccelRunOnTLC(bVal);
```

Arguments

`bVal`
: May be `'true'` (use TLC file) or `'false'` (run block in interpreted mode).

Description
Specify if the block should use its TLC file to generate code that runs with the accelerator. If this option is `'false'`, the block runs in interpreted mode.

`SetPreCompPortInfoToDefaults`

Purpose
Set compiled attributes of this block to default values.

Syntax
```
SetPreCompPortInfoToDefaults;
```

Description
Initialize the compiled information (dimensions, data type, complexity, and sampling mode) of this block's ports to have default attributes (double, real, sample-based scalars).

`SetPreCompPortInfoToDynamic`

Purpose
Set compiled attributes of this block to be inherited.

Syntax
```
SetPreCompPortInfoToDefaults;
```

Description
Initialize the compiled information (dimensions, data type, complexity, and sampling mode) of the block's ports to be inherited.

`SetSimViewingDevice`

Purpose
Specify whether this block is a viewer.

Syntax
```
SetSimViewingDevice(bVal);
```

Arguments

`bVal`
May be `'true'` (is a viewer) or `'false'` (is not a viewer).

Description
Specify if the block is a viewer/scope. If this flag is specified, the block will be used only during simulation and automatically stubbed out in generated code.

`WriteRTWParam`

Purpose
Write a custom parameter to the Real-Time Workshop information file used for code generation.

Syntax
```
WriteRTWParam(pType, pName, pVal)
```

Arguments

`pType`
Type of the parameter to be written. Valid values are `'string'` and `'matrix'`.

`pName`
Name of the parameter to be written.

`pVal`
Value of the parameter to be written.

Description

Use in the `mdlRTW` method of the M-file S-function to write out custom parameters. These parameters are generally settings used to determine how code should be generated in the TLC file for the S-function.

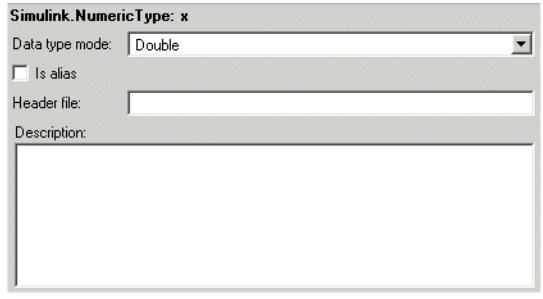
Purpose

Specify a data type

Description

This class lets you specify a data type. To do this, create an instance of this class in the MATLAB workspace and set its properties to the properties of the custom data type. Then assign this data type to all signals and parameters of your model that you want to conform to the data type. Assigning the data type in this way allows you to change the data types of the signals and parameters in your model by changing the properties of the object that describe them. You do not have to change the model itself.

Property Dialog Box



Data type mode

Data type of this numeric type. The options are

Option	Description
`Boolean`	Same as the MATLAB `boolean` type.
`Double`	Same as the MATLAB `double` type.
`Single`	Same as the MATLAB `single` type.
`Fixed-point: unspecified scaling`	A fixed-point data type with unspecified scaling.

Option	Description
`Fixed-point: binary point scaling`	A fixed-point data type with binary-point scaling.
`Fixed-point: slope and bias scaling`	A fixed-point data type with slope and bias scaling.

Selecting a category causes Simulink to disable other controls on the dialog box (see below) that apply to the category and to disable controls that do not apply. Selecting a fixed-point category may, depending on the other dialog box options that you select, cause the model to run only on systems that have a Simulink Fixed Point option installed.

Is alias

If this option is selected, Simulink uses the name of the workspace variable that references this object as the name of the data type. Otherwise, Simulink uses the category of the data type as its name, or, if the category is a fixed-point category, Simulink generates a name that encodes the type's properties, using the encoding specified by the Simulink Fixed Point product.

Header file

Name of a user-supplied C header file that defines a data type having the same name as this numeric type (i.e., as the MATLAB variable that references this object). If this field is not empty, code generated from this model defines the numeric type by including the specified header file. If this field is empty, the generated code defines the numeric type itself.

Description

Description of this data type. This field is intended for use in documenting this data type. Simulink ignores it.

Signed

Specifies whether the data type is signed or unsigned. This option is enabled only for fixed-point data type categories.

Word-Length

Word length in bits of the fixed-point data type. This option is enabled only for fixed-point data type categories.

Data type
Data type of the parameter. Simulink determines the data type from the entry in the parameter's **Value** field. You cannot set this field yourself.

Units
Measurement units in which this value is expressed, e.g., inches. This field is intended for use in documenting this parameter. Simulink ignores it.

Dimensions
Dimensions of the parameter. Simulink determines the dimensions from the entry in the **Value** field of this parameter. You cannot set this field yourself.

Complexity
Numeric type (i.e., real or complex) of the parameter. Simulink determines the numeric type of this parameter from the entry in the **Value** field of this parameter. You cannot set this field yourself.

Minimum
Minimum value that the parameter can have when tuned. This field is intended for use in documenting this parameter. Simulink ignores it.

Maximum
Maximum value that the parameter can have. This field is intended for use in documenting this parameter. Simulink ignores it.

Storage class
Storage class of this parameter. Simulink code generation products use this property to allocate memory for this parameter in generate code. See"Storage Classes of Tunable Parameters" in "Real-Time Workshop User's Guide" for more information.

Alias
Alternate name for this parameter. Simulink ignores this setting.

Description

Description of this parameter. This field is intended for use in documenting this parameter. Simulink ignores it.

Properties

Name	Access	Description
`Value`	`RW`	Value of this parameter. (Value)
`DataType`	`RO`	String specifying the name of the data type of this parameter. This field is read-only. (Data type)
`Dimensions`	`RO`	Vector specifying the dimensions of this parameter. (Dimensions)
`Complexity`	`RO`	String specifying the numeric type of this paramter. Valid values are `'real'` or `'complex'`. (Complexity)
`Min`	`RW`	Minimum value that this parameter can have. Simulink ignores this property. (Minimum)
`Max`	`RW`	Maximum value that this parameter can have. Simulink ignores this property. (Maximum)
`DocUnits`	`RW`	Measurement units in which this parameter's value is expressed. (Units)
`RTWInfo`	`RW`	Information used by Real-Time Workshop for generating code for this parameter. The value of this property is an object of `Simulink.ParamRTWInfo` class.

Purpose

Specify information needed to generate code for a parameter

Description

Simulink creates an instance of this class for each instance of a `Simulink.Parameter` object that it creates. Simulink uses the `Simulink.ParamRTWInfo` object to store information needed to generate code for the parameter specified by the `Simulink.Parameter` object.

You can set the properties of an instance of this class via the `RTWInfo` property or the property dialog box of the `Simulink.Parameter` object that uses it. For example, the following MATLAB expression sets the `StorageClass` property of a `Simulink.ParamRTWInfo` object used by a parameter object name `gain`.

```
gain.RTWInfo.StorageClass = 'ExportedGlobal';
```

Property Dialog Box

Use the `Simulink.Parameter` property dialog box to set the `StorageClass` and `Alias` properties of objects of this class. You must use MATLAB commands to set the other properties.

Properties

Name	Description
`Alias`	Alternate name for this parameter.
`CustomAttributes`	Custom storage class attributes of this parameter. See "Custom Storage Classes" in the Real-Time Workshop Embedded Coder documentation for more information.
`CustomStorageClass`	Custom storage class of this parameter.
`StorageClass`	Storage class of this parameter. See rtw_ug.f1136125 in the Real-Time Workshop documentation for more information.

Simulink.RunTimeBlock

Purpose

Allow Level-2 M-file S-function and other M-file programs to get information about a block while a simulation running.

Description

This class allows a Level-2 M-file S-function or other M program to obtain information about a block. Simulink creates an instance of this class or a derived class for each block in a model. Simulink passes the object to the callback methods of Level-2 M-file S-functions when it updates or simulates a model, allowing the callback methods to get block-related information from and provide such information to Simulink. See "Writing Level-2 M-file S-functions" in Writing S-Functions for more information. You can also use instances of this class in M-file programs to obtain information about blocks during a simulation. See “Accessing Block Data During Simulation” in Using Simulink for more information.

Parent Class

None

Derived Classes

`Simulink.MSFcnRunTimeBlock`

Property Summary

Name	Description
“BlockHandle”	Block’s handle.
“CurrentTime”	Current simulation time.
“NumDworks”	Number of discrete work vectors used by the block.
“NumOutputPorts”	Number of block output ports.
“NumContStates”	Number of block’s continuous states.
“NumDiscStates”	Number of block’s discrete states
“NumDlgParams”	Number of parameters that can be entered on S-function block’s dialog box.

Name	Description
“NumInputPorts”	Number of block’s input ports.
“NumRuntimePrms”	Number of run-time parameters used by block.
“SampleTimes”	Sample times at which block produces outputs.

Method Summary

Name	Description
“ContStates”	Get a block’s continuous states.
“DataTypeIsFixedPoint”	Get a block’s continuous states.
“DatatypeName”	Get name of a data type supported by this block.
“DatatypeSize”	Get size of a data type supported by this block.
“Derivatives”	Get a block’s continuous state derivatives.
“DialogPrm”	Get a parameter entered on an S-function block’s dialog box.
“Dwork”	Get one of a block’s Dwork vectors.
“FixedPointNumericType”	Determine the properties of a fixed-point data type.
“InputPort”	Get one of a block’s input ports.
“OutputPort”	Get one of a block’s output ports.
“RuntimePrm”	Get one of the run-time parameters used by a block.

Properties

`BlockHandle`

Description
Block's handle.

Access
`RO`

`CurrentTime`

Description
Current simulation time.

Access
`RO`

`NumDworks`

Description
Number of data work vectors.

Access
`RW`

See Also
`ssGetNumDWork`

`NumOutputPorts`

Description
Number of output ports.

Access
`RW`

See Also
`ssGetNumOutputPorts`

`NumContStates`

Description

Number of continuous states.

Access
RW

See Also
ssGetNumContStates

NumDiscStates

Description
Number of discrete states. In an M-file S-function, you need to use Dworks to setup discrete states.

Access
RW

See Also
ssGetNumDiscStates

NumDlgParams

Description
Number of parameters declared on the block's dialog. In the case of the S-function, it returns the number of parameters listed as a comma-separated list in the **S-function parameters** dialog field.

Access
RW

See Also
ssGetNumSFcnParams

NumInputPorts

Description
Number of input ports.

Access
RW

See Also
ssGetNumInputPorts

`NumRuntimePrms`

Description
Number of parameters declared on the block's dialog. In the case of the S-function, it returns the number of parameters listed as a comma-separated list in the **S-function parameters** dialog field.

Access
`RW`

See Also
`ssGetNumSFcnParams`

`SampleTimes`

Description
Blocks's sample times.

Access
`RW` for M-file S-functions, `RO` for all other blocks.

Methods

`ContStates`

Purpose
Get a block's continuous states.

Syntax
```
states = ContStates();
```

Description
Get vector of continuous states.

See Also
`ssGetContStates`

`DataTypeIsFixedPoint`

Purpose
Determine whether a data type is fixed point.

Syntax

```
bVal = DataTypeIsFixedPoint(dtID);
```

Arguments

`dtID`
Integer value specifying the ID of a data type.

Description
Returns `true` if the specified data type is a fixed-point data type.

`DatatypeName`

Purpose
Get the name of a data type.

Syntax
```
name = DatatypeName(dtID);
```

Arguments

`dtID`
Integer value specifying ID of a data type.

Description
Returns the name of the data type specified by `dtID`.

See Also
"DatatypeSize"

`DatatypeSize`

Purpose
Get the size of a data type.

Syntax
```
size = DatatypeSize(dtID);
```

Arguments

`dtID`
Integer value specifying the ID of a data type.

Description
Returns the size of the data type specified by `dtID`.

See Also
"DatatypeName"

`Derivatives`

Purpose
Get derivatives of a block's continuous states.

Syntax
```
states = ContStates();
```

Description
Get vector of state derivatives.

See Also
`ssGetdX`

`DialogPrm`

Purpose
Get an S-function's dialog parameters.

Syntax
```
param = DialogPrm(pIdx);
```

Description
Get the specified dialog parameter. In the case of the S-function, each `DialogPrm` corresponds to one of the elements in the comma-separated list of parameters in the **S-function parameters** dialog field.

See Also
`ssGetSFcnParam`, "RuntimePrm"

`Dwork`

Purpose
Get one of a block's Dwork vectors.

Syntax
```
dworkObj = Dwork(pIdx);
```

Description
Get information about the Dwork vector specified by `pIdx` where `pIdx` is the index number of the work vector. This method returns an object of type `Simulink.BlockCompDworkData`.

See Also
`ssGetDWork`

`FixedPointNumericType`

Purpose
Get the properties of a fixed-point data type.

Syntax
```
eno = FixedPointNumericType(dtID);
```

Arguments

`dtID`
Integer value specifying the ID of a fixed-point data type.

Description
Returns an object of `Embedded.Numeric` class that contains the attributes of the specified fixed-point data type.

Note `Embedded.Numeric` is also the class of the `numerictype` objects created by the Fixed-Point Toolbox. For information on the properties defined by `Embedded.Numeric` class, see `numerictype` Object Properties in the "Property Reference" in the “Fixed-Point Toolbox User’s Guide”.

`InputPort`

Purpose
Get an input port of a block.

Syntax
```
port = InputPort(pIdx);
```

Description

Get the input port specified by `pIdx`, where `pIdx` is the index number of the input port. For example,

```
port = rto.InputPort(1)
```

returns the first input port of the block represented by the run-time object `rto`.

This method returns an object of type `Simulink.BlockPreCompInputPortData` or `Simulink.BlockCompInputPortData`, depending on whether the model that contains the port is uncompiled or compiled. You can use this object to get and set the input port's uncompiled or compiled properties, respectively.

See Also

`ssGetInputPortSignalPtrs`, `Simulink.BlockPreCompInputPortData`, `Simulink.BlockCompInputPortData`, "OutputPort"

`OutputPort`

Purpose

Get an output port of a block.

Syntax

```
port = OutputPort(pIdx);
```

Description

Get the output port specified by `pIdx`, where `pIdx` is the index number of the output port. For example,

```
port = rto.InputPort(1)
```

returns the first output port of the block represented by the run-time object `rto`.

This method returns an object of type `Simulink.BlockPreCompOutputPortData` or `Simulink.BlockCompOutputPortData`, depending on whether the

model that contains the port is uncompiled or compiled. You can use this object to get and set the input port's uncompiled or compiled properties, respectively.

See Also
`ssGetInputPortSignalPtrs`,
`Simulink.BlockPreCompOutputPortData`,
`Simulink.BlockCompOutputPortData`

`RuntimePrm`

Purpose
Get an S-function's run-time parameters.

Syntax
```
param = RuntimePrm(pIdx);
```

Description
Get the specified run-time parameter.

See Also
`ssGetRunTimeParamInfo`

Simulink.ScopeDataLogs

Purpose

Log data displayed by a Scope viewer.

Description

Simulink creates instances of this class to log data displayed on Scope viewers (see "The Signal & Scope Manager" in Using Simulink). In particular, if you have enabled data logging for a model, Simulink creates an instance of this class for each scope viewer enabled for logging in the model and assigns it to a property of the model's `Simulink.ModelDataLogs` object. The instance created for each viewer has a Name property whose value is the name specified on the History pane of the viewer's parameter dialog box (see Scope for more information). The instance also has an axes property for each of the scope's axes labeled `Axes1`, `Axes2`, etc. The value of each axes property is itself a `Simulink.ScopeDataLogs` object that contains `Simulink.Timeseries` objects, one for each signal displayed on the axes. The time series objects contain the signal data displayed on the axes.

Consider, for example, the following model.

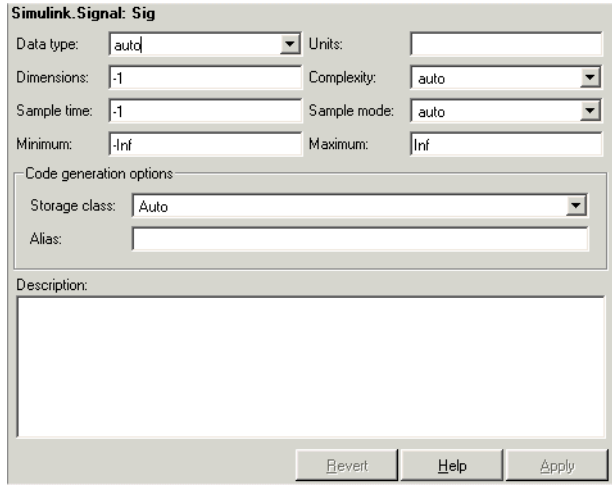
This model displays signals `out1` and `out2` on a single scope viewer that has only one set of axes. The model enables data logging for the scope viewer under the variable name `ScopeData` and for the model as a whole under the default variable name `logsout`. After simulation of the model, the MATLAB workspace contains a `Simulink.ModelDataLogs` object named `logsout` containing a `Simulink.ScopeDataLogs` object that in turn contains a `Simulink.ScopeDataLogs` object that contains `Simulink.Timeseries` objects that contain the times series data for signals `out1` and `out 2`.

In this example, the signal object named `s1` specifies the sample time and data type of the signal emitted by input port `In1` and a Signal Specification block specifies the sample time and data type of the signal emitted by input port `In2`. As this example illustrates, you have to display the signal object in the Model Explorer to determine many of its properties whereas the Signal Specification block displays the property values on the diagram itself. On the other hand, the use of a signal object to specify the sample time and data type properties of signal `s1` allows you to change the sample time or data type without having to edit the model. For example, you could use the Model Explorer, the MATLAB command line, or an M-file program to change these properties.

Using Signal Objects to Check Signal Properties

You can use signal objects to ensure that signal sources assign desired properties to a signal or state. To do this, create a signal object that has the same name as the signal or state to be validated and that specifies the desired properties. Then, whenever you update the diagram containing the signal or state, Simulink checks the properties of the signal specified by the signal's or state's source against the properties specified by the signal object. For most properties, if the source specifies a value other than inherited or auto for the property and the values specified by the source and the signal object differ, Simulink displays an error message. This enables you to quickly determine whether the actual attributes of your model's signals are the attributes you intend them to have.

Property Dialog Box



Data type

Data type of the signal. The default entry, `auto`, specifies that Simulink, should determine the data type. Use the adjacent pull-down list to specify built-in data types (e.g., `uint8`). To specify a custom data type, enter a MATLAB expression that specifies the type, e.g., a base workspace variable that references a `Simulink.NumericType` object.

Units

Measurement units in which the value of this signal is expressed, e.g., inches. This field is intended for use in documenting this signal. Simulink ignores it.

Dimensions

Dimensions of this signal. Valid values are `-1` (the default) specifying any dimensions, `N` specifying a vector signal of size `N`, or `[M N]` specifying an `MxN` matrix signal.

Complexity

Numeric type of the signal. Valid values are `auto` (determined by Simulink), `real`, or `complex`.

Sample time
Rate at which the value of this signal should be computed. See "Specifying Sample Time" in Using Simulink for information on how to specify the sample time.

Sample mode
Sample mode of this signal. Simulink ignores the setting of this field.

Minimum
Minimum value that the signal can have. This field is intended for use in documenting this signal.

Maximum
Maximum value that the signal can have. This field is intended for use in documenting this signal.

Storage class
Storage class of this signal. See "Storage Classes of Tunable Parameters" in the Real-Time Workshop User's Guide for more information.

Alias
Alternate name for this signal. Simulink ignores this setting.

Description
Description of this signal. This field is intended for use in documenting this signal.

Properties

Name	Access	Description
`DataType`	`RW`	String specifying the name of the data type of this signal. (Data type)
`Dimensions`	`RW`	Scalar or vector specifying the dimensions of this signal. (Dimensions)
`Complexity`	`RW`	String specifying the numeric type of this signal. Valid values are `'auto'`, `'real'`, or `'complex'`. (Complexity)

Name	Access	Description
Min	RW	Minimum value that this signal can have. (Minimum)
Max	RW	Maximum value that this signal can have. (Maximum)
DocUnits	RW	Measurement units in which this signal's value is expressed. (Units)
RTWInfo	RW	Information used by Real-Time Workshop for generating code for this signal. The value of this property is an object of Simulink.ParamRTWInfo class.
SampleTime	RW	Rate at which this signal should be updated. ()
Sampling Mode	RW	Sampling mode of this signal. (Sample mode)

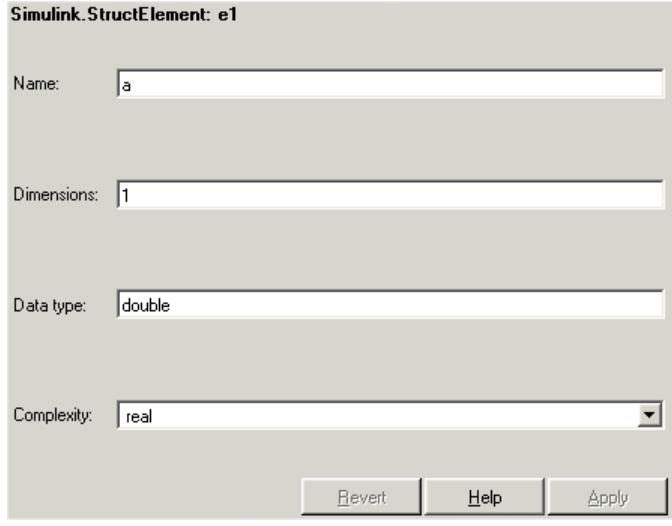
Purpose

Describe an element of a data structure

Description

Objects of this class describe elements of structures described by objects of the `Simulink.StructType` class.

Property Dialog Box



Name
Name of the element.

Dimensions
A vector specifying the dimensions of the element.

Data type
Name of the data type of this element.

Complexity
Numeric type (i.e., real or complex) of this element.

Properties

Name	Access	Description
`Name`	`RW`	String specifying the name of this element. (Name)
`Dimensions`	`RW`	A vector specifying the dimensions of this element. (Dimensions)
`DataType`	`RW`	String that specifies the name of the data type of this element. (Data type)
`Complexity`	`RW`	String that specifies the numeric type (`'real'` or `'complex'`) of this element. (Complexity)

See Also

`Simulink.StructType`

Purpose

Describe a data structure used as the value of a signal or parameter

Description

An object of this class describes a signal whose values are data structures (i.e., aggregates of data of different types as opposed to arrays of values of the same type). This class is intended to support development and use of custom blocks (e.g., S-Function blocks) that accept or output data structures. The class allows users of such blocks to determine the structure of the signals connected to them.

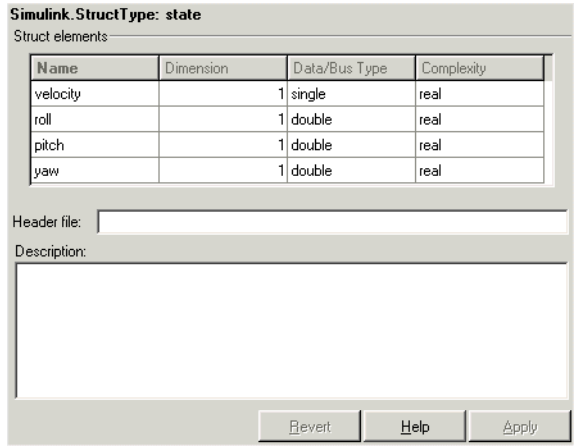
You can use either the Model Explorer or the MATLAB command line to create an instance of this class. To define the elements of a structure, create an array of instances of `Simulink.StructElement` at the MATLAB command line and assign the array as the value of the structure's `Elements` property. For example, the following commands define a structure that contains a floating point and an integer element.

```
v = Simulink.StructElement;
v.Name = 'v';
v.DataType = 'single';
n = Simulink.StructElement;
n.Name = 'n';
n.DataType = 'uint8';

s = Simulink.StructType;
s.Elements = [v n];
```

You can use a structure type object to specify the data type of Inport and Signal Specification blocks. To do this, enter the name of the variable that references the structure type object as the data type in the block's parameter dialog box.

The Simulink S-function API lets you create S-functions capable of generating and manipulating signal structures (see the `simstruct.h` header file for more information). You can connect signal structures created by S-function blocks to any standard Simulink block that accepts any data type. This includes virtual blocks and the Switch block configured to require the same data type on all its data inputs.

Property Dialog Box



Struct elements

Table that displays the properties of the structure's elements. You cannot edit this table. To add or delete this structure's elements or change the properties of elements, you must use MATLAB commands, e.g.,

```
state.Elements(1).DataType = 'double';
```

Header file

Name of a C header file that declares this structure. This field is intended for use by Real-Time Workshop. Simulink ignores it.

Description

Description of this structure. This field is intended for you to use to document this structure. Simulink itself does not use this field.

Properties

Name	Access	Description
`Elements`	`RW`	An array of `Simulink.StructElement` objects that define the names, data types, dimensions, and numeric types of the structure's elements. The elements must have unique names. (Struct elements)
`Description`	`RW`	String that describes this structure. This property is intended for user use. Simulink itself does not use it. (Description)
`HeaderFile`	`RW`	String that specifies the name of a C header file that declares this structure. (Header file)

See Also

`Simulink.StructElement`

Simulink.SubsysDataLogs

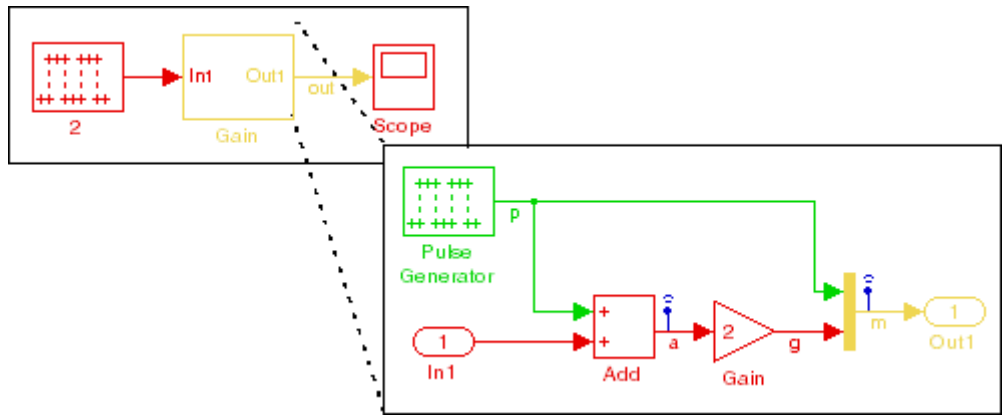
Purpose

Log signals in a subsystem

Description

Simulink creates instances of this class to contain logs for signals belonging to a subsystem (see "Logging Signals" in Using Simulink). Objects of this class have a variable number of properties. The first property, named `Name`, is the name of the subsystem whose log data this object contains. The remaining properties are signal log or signal log container objects containing the data logged for the subsystem specified by this object's `Name` property.

Consider, for example, the following model.



After simulation of this model, the MATLAB workspace contains a `Simulink.ModelDataLogs` object, named `logsout`, that contains a `Simulink.SubsysDataLogs` object, named `Gain`, that contains the log data for signals `a` and `g` in the subsystem named Gain.

```
>> logsout.Gain

ans =

Simulink.SubsysDataLogs (Gain):
  Name                    Elements  Simulink Class
```

```
a                               1       Timeseries
m                               2       TsArray
```

You can use either fully qualified log names or the `unpack` command to access the signal logs contained by a `SubsysDataLogs` object. For example, to access the amplitudes logged for signal `a` in the preceding example, you could enter the following at the MATLAB command line:

```
>> data = logsout.Gain.a.Data;
```

or

```
>> logsout.unpack('all');
data = a.Data;
```

See Also

`Simulink.ModelDataLogs`, `Simulink.Timeseries`, `Simulink.TsArray`, `unpack`

Simulink.TimeInfo

Purpose

Provide information about the time data in a `Simulink.Timeseries` object

Description

Simulink creates instances of these objects to describe the time data that it includes in `Simulink.Timeseries` objects.

Properties

Name	Access	Description
`Units`	`RW`	The units, e.g., `'secs'`, in which the time series data are expressed in the associated `Simulink.Timeseries` object.
`Start`	`RW`	If the associated signal is not in a conditionally executed subsystem, this field contains the simulation time of the first signal value recorded in the associated `Simulink.Timeseries` object. If the signal is in a conditionally executed subsystem, this field contains an array of times when the system became active.
`End`	`RW`	If the associated signal is not in a conditionally executed subsystem, this field contains the simulation time of the last signal value recorded in the associated `Simulink.Timeseries` object. If the signal is in a conditionally executed subsystem, this field contains an array of times when the system became inactive.

Name	Access	Description
`Increment`	`RW`	The interval between simulation times at which signal data is logged in the associated `Simulink.Timeseries` object.
`Length`	`W`	The number of signal samples recorded in the associated `Simulink.Timeseries` object, i.e., the length of the arrays referenced by the object's `Time` and `Data` properties.

See Also

`Simulink.Timeseries`

Simulink.Timeseries

Purpose

Log signal data

Description

Simulink creates instances of this class to store signal data that it logs while simulating a model (see "Logging Signals" in Using Simulink).

Note The MATLAB Time Series Tools can import and manipulate instances of this class. See Time Series Tools in the MATLAB Data Analysis documentation for further details.

Properties

Name	Access	Description
`Name`	`RW`	Name of this signal log.
`BlockPath`	`RW`	Path of the block that output the signal logged in this signal log.
`PortIndex`	`RW`	Index of the output port that emitted the signal logged in this signal log.
`SignalName`	`RW`	Name of the signal logged in this signal log.
`ParentName`	`RW`	Name of the parent of the signal recorded in this log, if the signal is an element of a composite signal; otherwise, the same as `SignalName`.
`TimeInfo`	`RW`	An object of `Simulink.TimeInfo` class that describes the time data in this log.
`Time`	`RW`	An array containing the simulation times at which signal data was logged.
`Data`	`RW`	An array containing the signal data.

See Also

`Simulink.ModelDataLogs`, `Simulink.TimeInfo`, `unpack`

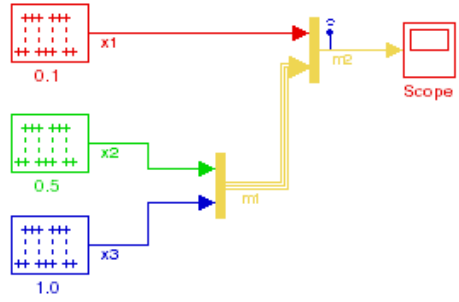
Purpose

Log composite virtual signals

Description

Simulink creates instances of this class to contain the data that it logs for a composite virtual signal, e.g., the output of a Mux or of a virtual Bus Creator block (see "Logging Signals"). Objects of the `Simulink.TsArray` class have a variable number of properties. The first property, called `Name`, specifies the log name of the composite signal. The remaining properties reference logs (i.e., `Simulink.Timeseries` objects) for the elements of the composite signal. The name of each property is the log name of the corresponding signal.

Consider, for example, the following model.



This model specifies that Simulink should log the values of the composite signal `m2` during simulation. After simulation of this model, the MATLAB workspace contains a `Simulink.ModelDataLogs` object, named `logsout`, that contains a `Simulink.TsArray object`, named `m2`, that contains the logs for the elements of `m2`, i.e., for the elementary signals `x1`, `x2`, and `x3`. Entering the fully qualified name of the `Simulink.TsArray` object, i.e., `logsout.m2`, at the MATLAB command line reveals the structure of the signal log for this model.

```
>> logsout.m2
Simulink.TsArray:
  Name                Simulink Class
```

```
x1                  Timeseries
x2                  Timeseries
x3                  Timeseries
```

You can use either fully qualified log names or the `unpack` command to access the signal logs contained by a `Simulink.TsArray` object. For example, to access the amplitudes logged for signal `x1` in the preceding example, you could enter the following at the MATLAB command line:

```
>> data = logsout.m2.x1.Data;
```

or

```
>> logsout.unpack('all');
data = x1.Data;
```

See Also

`Simulink.ModelDataLogs`, `Simulink.Timeseries`, `unpack`

Model and Block Parameters

The following sections list parameters that you can set for Simulink models and blocks, using the `set_param` command.

Model Parameters (p. 10-2)	Parameters specific to models.
Common Block Parameters (p. 10-41)	Parameters that all blocks have.
Block-Specific Parameters (p. 10-51)	Parameters that a specific block has.
Mask Parameters (p. 10-145)	Parameters of a masked subsystem.

Model Parameters

This table lists and describes parameters that describe a model. The parameters appear in the order they are defined in the model file, as described in Chapter 11, "Model File Format". The table also includes model callback parameters (see "Using Callback Functions"). The **Description** column indicates where you can set the value on the **Simulation Parameters** dialog box. Model parameters that are simulation parameters are described in more detail in . Examples showing how to change parameters follow the table (see "Examples of Setting Model Parameters" on page 10-40).

Parameter values must be specified as quoted strings. The string contents depend on the parameter and can be numeric (scalar, vector, or matrix), a variable name, a filename, or a particular value. The **Values** column shows the type of value required, the possible values (separated with a vertical line), and the default value, enclosed in braces.

Model Parameters

Parameter	Description	Values
`AbsTol`	Absolute error tolerance. Setting for the **Absolute tolerance** on the **Solver** pane of the **Configuration Parameters** dialog box.	string `{'auto'}`
`AccelMakeCommand`	Program that builds the Simulink Accelerator target for this model.	string `{'make_rtw'}`
`AccelSystemTargetFile`	TLC file used to build the Simulink Accelerator target for this model.	string `{'accel.tlc'}`
`AccelTemplateMakefile`	Template for the makefile used to build the Simulink Accelerator target for this model.	string `{'accel_default_tmf'}`

Model Parameters (Continued)

Parameter	Description	Values
`AlgebraicLoopMsg`	Specifies diagnostic action to take when there is an algebraic loop. Set by the **Algebraic loop** option on the **Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | {'warning'} | 'error'`
`AnalyticLinearization`	For internal use.	
`ArrayBoundsChecking`	Setting for the **Array bounds exceeded** diagnostic on the **Data Integrity Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'none'} | 'warning' | 'error'`
`ArtificialAlgebraic LoopMsg`	Setting for the **Minimize algebraic loop** diagnostic on the **Solver Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | {'warning'} | 'error'`
`AssertionControl`	Setting for the **Model Verification block enabling** control on the **Data Integrity Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'UseLocalSettings'} | 'EnableAll' | 'DisableAll'`
`BlockDescription StringDataTip`	Specifies whether to display the user description string for a block as a data tip. Set by the **User Description String** command on the model editor's **View->Block Data Tips Options** menu.	`'on' | {'off'}`
`BlockDiagramType`	Type of block diagram.	`'model' | 'library'`

Model Parameters (Continued)

Parameter	Description	Values
`BlockNameDataTip`	Specifies whether to display the user description string for a block as a data tip. Set by the **Block Name** command on the model editor's **View->Block Data Tips Options** menu.	`'on' | {'off'}`
`BlockParametersData Tip`	Specifies whether to display the user description string for a block as a data tip. Set by the **Parameter Names and Values** command on the model editor's **View->Block Data Tips Options** menu.	`'on' | {'off'}`
`BlockPriority ViolationMsg`	Setting for the **Block priority violation** diagnostic on the **Solver Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | {'warning'} | 'error'`
`BlockReductionOpt`	Enables block reduction optimization. Set by the **Block reduction optimization** option on the **Optimization** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`
`Blocks`	Names of the blocks that this model contains.	`cell array {{}}`
`BooleanDataType`	Enable Boolean mode. Set by the **Implement logic signals as boolean data (vs. double)** option on the **Optimization** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`
`Browser`	Deprecated.	

Model Parameters (Continued)

Parameter	Description	Values
`BrowserHandle`	Deprecated.	
`BrowserLookUnder Masks`	Show masked subsystems in the Model Browser. Set by the **Show Masked Subsystems** command on the model editor's **View->Model Browser Options** menu.	`'on' | {'off'}`
`BrowserShowLibraryLinks`	Show library links in the Model Browser. Set by the **Show Library Links** command on the model editor's **View->Model Browser Options** menu.	`'on' | {'off'}`
`BusObjectLabelMismatch`	Set by the **Element name mismatch** option on the **Connectivity** panel of the **Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'none'} | 'warning'| 'error'`
`BufferReuse`	Enable reuse of block I/O buffers. Set by the **Reuse block outputs** option on the **Optimization** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`
`CharacterEncoding`	Specifies the character set used to encode this model. See the `slCharacterEncoding` command for more information.	`string`
`CheckExecutionContext RuntimeOutputMsg`	Set by the **Check runtime output of execution context** option on the **Compatibility Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`

Model Parameters (Continued)

Parameter	Description	Values
`CheckExecutionContext PreStartOutputMsg`	Set by the **Check preactivation output of execution context** option on the **Compatibility Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`
`CheckMatrix SingularityMsg`	Set by the **Attempted division by singular matrix** option on the **Data Integrity Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'none'} | 'warning' | 'error'`
`CheckModelReference TargetMessage`	Message behavior when the **Rebuild options** is set to never in the **Model Referencing Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | 'warning' | {'error'}`
`CheckSSInitial OutputMsg`	Enable checking for undefined initial subsystem output. Set by the **Check undefined subsystem initial output** option on the **Compatibility Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`
`CloseFcn`	Close callback.	command or variable
`ConditionallyExecute Inputs`	Enable conditional input branch execution optimization. Set by the **Conditional input branch execution** control on the **Optimization** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`

Model Parameters (Continued)

Parameter	Description	Values
`ConfigurationManager`	Configuration manager for this model.	string `{'None'}`
`ConsistencyChecking`	Consistency checking. Set by the **Solver data inconsistency** option on the **Solver Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'none'} | 'warning' | 'error'`
`CovCompData`	If `CovHTMLOptions` is set to `on`, and `CovCumulativeReport` is set to `off`, this parameter specifies cvdata objects containing additional model coverage data to include in the model coverage report. Specified by the **Additional data to include in report (cvdata objects)** field in the **Report** pane of the **Coverage Settings** dialog box.	string

Model Parameters (Continued)

Parameter	Description	Values
`CovCumulativeReport`	If `CovHTMLReporting` is set to `on`, this parameter allows the `CovCumulativeReport` and `CovCompData` parameters to specify the number of coverage results displayed in the model coverage report. If set to `on`, display the coverage results for the last simulation in the report. If set to `off`, display the coverage results from successive simulations in the report. Set by the radial buttons **Cumulative runs** (on)/**Last runs** (off) in the **Report** pane of the **Coverage Settings** dialog box.	`'on' | {'off'}`
`CovCumulativeVarName`	If `covSaveCumulativeToWorkSpaceVar` is set to `on`, model coverage saves the results of successive simulations in the workspace variable specified by this property. Entered in the field below the selected **Save cumulative results in workspace variable** check box on the **Results** pane of the **Coverage Settings** dialog box.	string `{'covCumulativeData'}`

Model Parameters (Continued)

Parameter	Description	Values
`CovHTMLOptions`	If `CovHTMLReporting` is set to `on`, use this parameter to select from a set of display options for the resulting model coverage report. In the **Reporting** pane of the **Coverage Settings** dialog box, select **Settings** to receive a dialog box for selecting these options.	String of appended character sets separated by a space. HTML options are enabled or disabled through a value of 1 or 0, respectively, in the following character sets (default values shown): • `'-aTS=1'` Include each test in the model summary • `'-bRG=1'` Produce bar graphs in the model summary • `'-bTC=0'` Use two color bar graphs (red, blue) • `'-hTR=0'` Display hit/count ratio in the model summary • `'-nFC=0'` Do not report fully covered model objects • `'-scm=1'` Include cyclomatic complexity numbers in summary • `'-bcm=1'` Include cyclomatic complexity numbers in block details

Model Parameters (Continued)

Parameter	Description	Values
`CovHTMLReporting`	Set to `on` to tell Simulink to create an HTML report containing the coverage data in the MATLAB Help browser at the end of the simulation. Set by the **Generate HTML report** check box on the **Report** pane of the **Coverage Settings** dialog box.	`{'on'} | 'off'`

Model Parameters (Continued)

Parameter	Description	Values
`CovMetricSettings`	Selects coverage metrics for coverage report. Coverage metrics are enabled by selecting the check boxes for individual coverages in the **Coverage Metrics** section of the **Coverage** pane of the **Coverage Settings** dialog box. Options `'s'` and `'w'` are enabled by selecting the check boxes **Treat Simulink logic blocks as short-circuited** and **Warn when unsupported blocks exist in model**, respectively, in the **Options** pane of the **Coverage Settings** dialog box. Option `'e'` is disabled by selecting the check box **Display coverage results using model coloring** in the **Results** pane of the **Coverage Settings** dialog box.	string `{'dw'}` Each order-independent character in the string enables a coverage metric or option as follows: • `'d'` Enable decision coverage • `'c'` Enable condition coverage • `'m'` Enable MCDC coverage • `'t'` Enable lookup table coverage • `'r'` Enable signal range coverage • `'s'` Treat Simulink logic blocks as short-circuited • `'w'` Warn when unsupported blocks exist in model • `'e'` Eliminate model coloring for coverage results

Model Parameters (Continued)

Parameter	Description	Values
`CovNameIncrementing`	If `CovSaveSingleToWorkspaceVar` is set to `on`, setting this parameter to `on` tells Model Coverage to increment the workspace variable specified in `CovSaveName` to store the results succeeding simulations. Entered in the **Increment variable name with each simulation** check box below the selected **Save last run in workspace variable** check box on the **Results** pane of the **Coverage Settings**dialog box.	`'on' | {'off'}`
`CovPath`	Model path of the subsystem for which Simulink gathers and reports coverage data. Set by browsing for the path in **Coverage Instrumentation Path** on the **Coverage** pane of the **Coverage Settings** dialog box.	`string {'/'}`
`CovReportOnPause`	Specifies that when you pause during simulation the model coverage report appears in updated form with coverage results up to the current pause or stop time. Set by selecting the **Update results on pause** check box on the **Results** pane of the **Coverage Settings** dialog box.	`{'on'} | 'off'`

Model Parameters (Continued)

Parameter	Description	Values
`covSaveCumulativeToWorkspaceVar`	If set to `on`, causes Model Coverage to accumulate and save the results of successive simulations in the workspace variable in `CovCumulativeVarName`. Set by selecting the **Save cumulative results in workspace variable** check box on the **Results** pane of the **Coverage Settings** dialog box.	`{'on'} | 'off'`
`CovSaveName`	If `CovSaveSingleToWorkspaceVar` is set to `on`, Model Coverage saves the results of the last simulation run in the workspace variable specified by this property. Entered in the field below the selected **Save last run in workspace variable** check box on the **Results** pane of the **Coverage Settings** dialog box.	string `{'covdata'}`
`CovSaveSingleToWorkspaceVar`	If enabled, tells Model Coverage to save the results of the last simulation run in the workspace variable specified by the `CovSaveName` property. Set by selecting the **Save last run in workspace variable** check box on the **Results** pane of the **Coverage Settings** dialog box.	`{'on'} | 'off'`
`Created`	Date and time model was created.	`string`
`Creator`	Name of model creator.	string `{''}`
`CurrentBlock`	For internal use.	
`CurrentOutputPort`	For internal use.	

Model Parameters (Continued)

Parameter	Description	Values
`DataTypeOverride`	Specifies data type used to override fixed-point data types. Set by the **Data type override** control on the **Fixed-Point Settings** dialog box.	`{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`
`Decimation`	Decimation factor. Set by the **Decimation** field on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	string `{'1'}`
`DeleteChildFcn`	Delete child callback.	string `{''}`
`Description`	Description of this model.	string
`Dirty`	If the parameter is `on`, the model has unsaved changes.	`'on' | {'off'}`
`DiscreteInherit ContinuousMsg`	Specifies diagnostic action to take when a Unit Delay block inherits a continuous sample time. Set by the **Discrete used as continuous** control on the **Sample Time Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | {'warning'} | 'error'`
`DisplayBdSearch Results`	For internal use.	
`DisplayBlockIO`	For internal use.	
`DisplayCallGraph Dominators`	For internal use	
`DisplayCompileStats`	For internal use.	
`DisplayCondInputTree`	For internal use.	
`DisplayCondStIdTree`	For internal use.	

Model Parameters (Continued)

Parameter	Description	Values
`DisplayError Directions`	For internal use.	
`DisplayInvisible Sources`	For internal use.	
`DisplaySortedLists`	For internal use.	
`DisplayVectorAnd FunctionCounts`	For internal use.	
`DisplayVect PropagationResults`	For internal use.	
`ExecutionContextIcon`	Toggles display of execution context icons on this model's block diagram.	`'on' | {'off'}`
`ExternalInput`	Names of MATLAB workspace variables used to designate data and times to be loaded from the workspace. Set by the **Input** option on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	scalar or vector `{'[t, u]'}`
`ExpressionFolding`	Enables expression folding. Set by the **Eliminate superfluous temporary variables** option on the **Optimization** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`
`ExtMode...`	Parameters whose names start with `ExtMode` apply to Simulink External Mode. See External Mode in the Real-Time Workshop User's Guide for more information.	

Model Parameters (Continued)

Parameter	Description	Values
`ExtrapolationOrder`	Extrapolation order of the `ode14x` implicit fixed-step solver. Set by the **Extrapolation order** control on the **Solver** pane of the **Configuration Parameters** dialog box.	`1 | 2 | 3 | {4}`
`FileName`	For internal use.	
`FinalStateName`	Names of final states to be saved to the workspace. Set by the **Final states** option on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	string `{'xFinal'}`
`FixedStep`	Fixed step size. Set by the **Fixed step size (fundamental sample time)** field on the **Solver** pane of the **Configuration Parameters** dialog box.	string `{'auto'}`
`FixPtInfo`	For internal use.	
`ForceArrayBounds Checking`	For internal use.	
`ForceConsistency Checking`	For internal use.	
`ForceModelCoverage`	For internal use.	
`ForwardingTable`	Specifies the forwarding table for this library. See in Using Simulink for more information.	`{{'old_path_1', 'new_path_1'} ... {'old_path_n', 'new_path_n'}}`
`ForwardingTable String`	For internal use.	

Model Parameters (Continued)

Parameter	Description	Values
`GridSpacing`	Spacing of model editor grid in pixels.	`double`
`Handle`	Handle of this model's block diagram.	double
`HiliteAncestors`	For internal use.	
`IgnoreBidirectional Lines`	For internal use.	
`InheritedTsInSrcMsg`	Message behavior when the sample time is inherited. Set by the **Source block specifies -1 sample time** control on the **Sample Time Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | {'warning'} | 'error'`
`InitFcn`	Function that is called when this model is first compiled for simulation.	string `{''}`
`InitialState`	Initial state name or values. Set by the **Initial state** field on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	variable or vector `{'xInitial'}`
`InitialStep`	Initial step size. Set by the **Initial step size** field on the **Solver** pane of the **Configuration Parameters** dialog box.	string `{'auto'}`
`InlineParams`	Enable inline of parameters in generated code. Set by the **Inline parameters** check box on the **Optimization** pane of the **Configuration Parameters** dialog box.	`'on' | {'off'}`

Model Parameters (Continued)

Parameter	Description	Values
`Int32ToFloatConvMsg`	Message behavior when a 32-bit integer is converted to a single-precision float. Set by the **32-bit integer to single precision float conversion** control on the **Data Integrity Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | {'warning'}`
`IntegerOverflowMsg`	Message behavior when there is an integer overflow. Set by the **Data overflow** control on the **Data Integrity Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | {'warning'} | 'error'`
`InvalidFcnCallConnMsg`	Message behavior when there is an invalid function call connection. Set by the **Invalid function call connection** control on the **Connectivity Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | 'warning' | {'error'}`
`InvariantConstants`	Invariant constant setting.	`'on' | {'off'}`
`Jacobian`	For internal use.	
`LastModifiedBy`	Name used for version control.	string
`LastModifiedDate`	Date used for version control.	string

Model Parameters (Continued)

Parameter	Description	Values
`LibraryLinkDisplay`	Shows which blocks in the model are linked or have disabled or modified links. Set by the **Library Link Display** option under the **Format** menu.	`{'none'} | 'user' | 'all'`
`LibraryType`	For internal use.	`{'none'} | 'BlockLibrary' | 'IOLibrary'`
`LimitDataPoints`	Limit output. Set by the **Limit data points to last** check box on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`
`LinearizationMsg`	For internal use.	
`Lines`	For internal use.	
`LoadExternalInput`	Load input from workspace. Set by the **Input** check box on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	`'on' | {'off'}`
`LoadInitialState`	Load initial state from workspace. Set by the **Initial state** check box on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	`'on' | {'off'}`
`Location`	For internal use.	
`Lock`	Lock/unlock a block library. Setting this parameter on prevents a user from inadvertently changing a library.	`'on' | {'off'}`

Model Parameters (Continued)

Parameter	Description	Values
`MaxDataPoints`	Maximum number of output data points to save. Set by the **Limit data points to last** field on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	string `{'1000'}`
`MaxNumMinSteps`	Maximum number of times the solver uses the minimum step size.	string `{'-1'}`
`MaxOrder`	Maximum order for `ode15s`. Set by the **Maximum order** option on the **Solver** pane of the **Configuration Parameters** dialog box.	`1 | 2 | 3 | 4 | {5}`
`MaxStep`	Maximum step size. Set by the **Max step size** field on the **Solver** pane of the **Configuration Parameters** dialog box.	string `{'auto'}`
`MdlSubVersion`	For internal use	
`MinMaxOverflowArchiveData`	For internal use	
`MinMaxOverflowArchiveMode`	Logging type for fixed-point logging. Set by the **Logging type** option in the **Fixed-Point Settings** dialog box.	`{'Overwrite'} | 'Merge'`
`MinMaxOverflowLogging`	Setting for fixed-point logging. Set by the **Logging mode** option in the **Fixed-Point Settings** dialog box.	`{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`
`MinStep`	Minimum step size for the solver. Set by the **Min step size** field on the **Solver** pane of the **Configuration Parameters** dialog box.	string `{'auto'}`

Model Parameters (Continued)

Parameter	Description	Values
`MinStepSizeMsg`	Message shown when minimum step size is violated. Set by the **Min step size violation** option on the **Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'warning'} | 'error'`
`ModelBrowser Visibility`	Show the Model Browser. Set by the **Model Browser** command of the model's **View->Model Browser Options** menu.	`'on' | {'off'}`
`ModelDataFile`	For internal use.	string `{''}`
`ModelDependencies`	List of model dependencies. Set by the **Model dependencies** field on the **Model Referencing** pane of the **Configuration Parameters** dialog box.	string `{''}`
`ModelReferenceCS MismatchMessage`	Message shown when there is a model configuration mismatch. Set by the **Model configuration mismatch** option on the **Model Referencing Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'none'} | 'warning' | 'error'`
`ModelReferenceData LoggingMessage`	Message shown when there is unsupported data logging. Set by the **Unsupported data logging** option on the **Model Referencing Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | {'warning'} | 'error'`

Model Parameters (Continued)

Parameter	Description	Values
`ModelReferenceIO MismatchMessage`	Message shown when there is a port and parameter mismatch. Set by the **Port and parameter mismatch** option on the **Model Referencing Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'none'} | 'warning' | 'error'`
`ModelReferenceIOMsg`	Message shown when there is an invalid root Inport/Outport block connection. Set by the **Invalid root Inport/Outport block connection** option on the **Model Referencing Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'none'} | 'warning' | 'error'`
`ModelReferenceNum InstancesAllowed`	Total number of instances allowed per top model. Set by the **Total number of instances allowed per top model** option on the **Model Referencing** pane of the **Configuration Parameters** dialog box.	`'Zero' | 'Single' | {'Multi'}`
`ModelReferenceSim TargetVerbose`	Print detailed information when generating simulation targets for models referenced by a top-level model.	`'on' | {'off'}`
`ModelReferenceTarget Type`	For internal use.	

Model Parameters (Continued)

Parameter	Description	Values
`ModelReferenceVersion MismatchMessage`	Message shown when there is a model block version mismatch. Set by the **Model block version mismatch** option on the **Model Referencing Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'none'} | 'warning' | 'error'`
`ModelrefMinAlgLoop Occurrences`	Toggles the minimization of algebraic loop occurrences. Set by the **Pass scalar root inputs by value** check box on the **Model Referencing** pane of the **Configuration Parameters** dialog box.	`'on' | {'off'}`
`ModelrefPassRoot InputsByReference`	Toggles the passing of scalar root inputs by value. Set by the **Minimize algebraic loop occurrences** check box on the **Model Referencing** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`
`ModelVersion`	Version number of model.	string `{'1.1'}`
`ModelVersionFormat`	Format of model's version number.	string `{'1.%<AutoIncrement: 0>'}`
`ModelWorkspace`	References this model's model workspace object.	an instance of the `Simulink.ModelWorkspace` class
`ModifiedBy`	Last modifier of this model.	string
`ModifiedByFormat`	Format for the display of last modifier.	string `{'%<Auto>'}`
`ModifiedComment`	Field for user comments.	string `{''}`

Model Parameters (Continued)

Parameter	Description	Values
`ModifiedDate`	Date of last model modification.	string
`ModifiedDateFormat`	Format of modified date.	string `{'%<Auto>'}`
`ModifiedHistory`	Area for keeping notes about the history of the model. This is set by the **Model history** field on the **History** pane of the **Model Explorer** dialog box.	string `{''}`
`MultiTaskRateTransMsg`	Specifies diagnostic action to take when an invalid rate transition takes place between two blocks operating in single-tasking mode. Set by the **Multitask rate transition** control on the **Sample Time Diagnostics** pane of the **Configuration Parameters** dialog box.	`'warning' | {'error'}`
`Name`	Model name.	string
`NumberNewton Iterations`	Number of Newton's Method iterations performed by the `ode14x` implicit fixed-step solver. Set by the **Number Newton's iterations** control on the **Solver** pane of the pane of the **Configuration Parameters** dialog box.	integer `{'1'}`
`ObjectParameters`	Names/attributes of model parameters.	structure
`Open`	For internal use.	

Model Parameters (Continued)

Parameter	Description	Values
`OptimizeBlockIO Storage`	Enables signal storage reuse optimization. Set by the **Signal storage reuse** control on the **Optimization** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`
`OutputOption`	Time step output options for variable-step solvers. Set by the **Output options** option on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	`'AdditionalOutputTimes' | {'RefineOutputTimes'} | 'SpecifiedOutputTimes'`
`OutputSaveName`	Workspace variable to store the model outputs. Set by the **Output** field on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	`variable {'yout'}`
`OutputTimes`	Output times set when **Output options** on the **Data Import/Export** pane of the **Configuration Parameters** dialog box is set to **Produce additional output**. Set by the **Output times** option on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	string `{'[]'}`
`PaperOrientation`	Printing paper orientation.	`'portrait' | {'landscape'} | 'rotated'`
`PaperPosition`	Position of diagram on paper.	`[left, bottom, width, height]`
`PaperPositionMode`	Paper position mode.	`{'auto'} | 'manual'`

Model Parameters (Continued)

Parameter	Description	Values
`PaperSize`	Size of `PaperType` in `PaperUnits`.	`[width height]` (read only)
`PaperType`	Printing paper type.	`'usletter' | 'uslegal' | 'a0' | 'a1' | 'a2' | 'a3' | 'a4' | 'a5' | 'b0' | 'b1' | 'b2' | 'b3' | 'b4' | 'b5' | 'arch-A' | 'arch-B' | 'arch-C' | 'arch-D' | 'arch-E' | 'A' | 'B' | 'C' | 'D' | 'E' | 'tabloid'`
`PaperUnits`	Printing paper size units.	`'normalized' | {'inches'} | 'centimeters' | 'points'`
`ParameterArgument Names`	List of parameters used as arguments when this model is called as a reference. Set in the **Model arguments (for referencing this model)** field in the **Model Workspace** pane of the **Model Explorer**.	string `{''}`
`ParameterDowncastMsg`	Specifies diagnostic action to take when a parameter downcast occurs during simulation. Set by the **Parameter downcast** control on the **Data Integrity Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | 'warning' | {'error'}`

Model Parameters (Continued)

Parameter	Description	Values
`ParameterOverflowMsg`	Specifies diagnostic action to take when a parameter overflow occurs during simulation. Set by the **Parameter overflow** control on the **Data Integrity Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | 'warning' | {'error'}`
`ParameterPooling`	Toggles pooling of parameters in Real-Time Workshop generated code.	`{'on'} | 'off'`
`ParameterPrecision LossMsg`	Specifies diagnostic action to take when parameter precision loss occurs during simulation. Set by the **Parameter precision loss** control on the **Data Integrity Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | {'warning'} | 'error'`
`ParamWorkspaceSource`	For internal use.	
`Parent`	Name of the model or subsystem that owns this object. The value of this parameter for a model is an empty string.	string `{''}`
PostLoadFcn	Function invoked just after this model is loaded.	string `{''}`
`PostSaveFcn`	Function invoked just after this model is saved to disk.	string `{''}`
`PreLoadFcn`	Preload callback.	command or variable `{''}`
`PreSaveFcn`	Function invoked just before this model is saved to disk.	string `{''}`

Model Parameters (Continued)

Parameter	Description	Values
`ProdHWDeviceType`	Predefined hardware device to specify the C language constraints for your microprocessor. Set by the **Device type** option on the **Hardware Implementation** pane of the **Configuration Parameters** dialog box.	string `{'32-bit Generic'}`
`ProdHWWordLengths`	Number of bits used for char, short, int, and long, respectively (set by the hardware device type).	string `{'8,16,32,32'}`
`Profile`	Enables the simulation profiler for this model.	`'on' | {'off'}`

Model Parameters (Continued)

Parameter	Description	Values

`RecordCoverage`	A value of on causes Simulink to gather and report model coverage data during simulation. The format of this report is controlled by the values of the following parameters:
	`CovCompData`
	`CovCumulativeReport`
	`CovCumulativeVarName`
	`CovHTMLOptions`
	`CovHTMLReporting`
	`CovMetricSettings`
	`CovNameIncrementing`
	`CovPath`
	`CovReportOnPause`
	`covSaveCumulativeToWorkSpaceVar`
	`CovSaveName`
	`CovSaveSingleToWorkspaceVar`
	If the value is `off`, no model coverage data is collected or reported and the preceding coverage report parameters have no effect.
		`'on' | {'off'}`

`Refine`	Refine factor. Set by the **Refine factor** field on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	string `{'1'}`

Model Parameters (Continued)

Parameter	Description	Values
`RelTol`	Relative error tolerance. Set by the **Relative tolerance** field on the **Solver** pane of the **Configuration Parameters** dialog box.	string `{'1e-3'}`
`ReportName`	Name of the associated file for the Report Generator	string `{'simulink-default.rpt'}`
`RequirementInfo`	For internal use.	
`RTW...`	See the Real-Time Workshop documentation for more information on parameters whose names begin with `RTW`.	
`SampleTimeColors`	Set by the Sample Time Colors option under the **Format** menu.	`'on' | {'off'}`
`SampleTimeConstraint`	Set by the Periodic Sample Time Constraint option on the **Configuration Parameters** dialog box.	`{'unconstrained'} | 'STIndependent' | 'Specified'`
`SaveDefaultBlock Params`	For internal use.	
`SaveFinalState`	Save final states to workspace. Set by the **Final states** check box on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	`'on' | {'off'}`
`SaveFormat`	Format used to save data to the MATLAB workspace. Set by the **Format** option on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	`{'Array'} | 'Structure' | 'StructureWithTime'`

Model Parameters (Continued)

Parameter	Description	Values
`SaveOutput`	Save simulation output to workspace. Set by the **Output** check box on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`
`SaveState`	Save states to workspace. Set by the **States** check box on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	`'on' | {'off'}`
`SaveTime`	Save simulation time to workspace. Set by the **Time** check box on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`
`ScreenColor`	Background color of the model window. Set by the Screen color option under the **Format** menu.	`'black' | {'white'} | 'red' | 'green' | 'blue' | 'cyan' | 'magenta' | 'yellow' | 'gray' | 'lightBlue' | 'orange' | 'darkGreen' | [r,g,b,a]` where `r`, `g`, `b`, and `a` are the red, green, blue, and alpha values of the color normalized to the range 0.0 to 1.0. The alpha value is ignored.
`ScrollbarOffset`	For internal use.	

Model Parameters (Continued)

Parameter	Description	Values
SfunCompatibility CheckMsg	Specifies diagnostic action to take when S-function upgrades are needed. Set by the **S-function upgrades needed** option on the **Compatibility Diagnostics** pane of the **Configuration Parameters** dialog box.	{'none'} | 'warning' | 'error'
ShowGrid	Show the Model Editor grid.	'on' | {'off'}
ShowLinearization Annotations	Toggles linearization icons in the model.	{'on'} | 'off'
ShowLineDimensions	Show signal dimensions on this model's block diagram. Set by the **Show Signal Dimensions** command on the **Format->Port/Signal Displays** menu.	'on' | {'off'}
ShowLineDimensions OnError	For internal use.	
ShowLineWidths	Show **Line Widths** menu option.	'on' | {'off'}
ShowLoopsOnError	Highlight invalid loops graphically.	{'on'} | 'off'
ShowModelReference BlockIO	Toggles display of version on block.	'on' | {'off'}
Shown	For internal use.	
ShowPortDataTypes	Show data types of ports on this model's block diagram. Set by the **Port Datatypes** command on the **Format->Port/Signal Displays** menu.	'on' | {'off'}
ShowPortDataTypesOn Error	For internal use.	

Model Parameters (Continued)

Parameter	Description	Values
`ShowStorageClass`	Show storage classes of signals on this model's block diagram. Set by the **Storage Class** command on the **Format->Port/Signal Displays** menu.	`'on' | {'off'}`
`ShowTestPointIcons`	Show test point icons on this model's block diagram. Set by the **Testpoint Icons** command on the **Format->Port/Signal Displays** menu.	`'on' | {'off'}`
`ShowViewerIcons`	Show viewer icons on this model's block diagram. Set by the **Viewer Icons** command on the **Format->Port/Signal Displays** menu.	`'on' | {'off'}`
`SignalLabelMismatchMsg`	Specifies diagnostic action to take when there is a signal label mismatch. Set by the **Signal label mismatch** option on the **Connectivity Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'none'} | 'warning' | 'error'`
`SignalLogging`	Globally enable signal logging for this model. Set by the **Signal logging** check box on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	`{'on'} | 'off'`
`SignalLoggingName`	Name for saving signal logging data to the MATLAB workspace. Set by the **Signal logging** field on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	`string {'logsOut'}`

Model Parameters (Continued)

Parameter	Description	Values
`SignalResolutionControl`	Control which named states and signals get resolved to Simulink signal objects. Set by the **Signal resolution control** check box on the **Data Integrity Diagnostics** pane of the **Configuration Parameters** dialog box.	`'UseLocalSettings' | 'TryResolveAll' | {'TryResolveAllWithWarning'}`
`SimulationCommand`	Executes a simulation command.	`'start' | 'stop' | 'pause' | 'continue' | 'step' | 'update' | 'SimParamDialog' | 'connect' | 'disconnect' | 'WriteExtModeParamVect' | 'AccelBuild'`
`SimulationMode`	Indicates whether Simulink should run in normal, accelerated, or external mode.	`{'normal'} | 'accelerator' | 'external'`
`SimulationStatus`	Indicates simulation status.	`{'stopped'} | 'updating' | 'initializing' | 'running' | 'paused' | 'terminating' | 'external' |`
`SimulationTime`	Current time value for the simulation.	double `{0}`
`SingleTaskRateTransMsg`	Specifies diagnostic action to take when a rate transition takes place between two blocks operating in single-tasking mode. Set by the **Single task rate transition** control on the **Sample Time Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'none'} | 'warning' | 'error'`

Model Parameters (Continued)

Parameter	Description	Values
`Solver`	Solver used for the simulation. Set by the **Solver** check box on the **Solver** pane of the **Configuration Parameters** dialog box.	`'VariableStepDiscrete' | {'ode45'} | 'ode23' | 'ode113' | 'ode15s' | 'ode23s' | 'ode23t' | 'ode23tb' | 'FixedStepDiscrete' | 'ode5' | 'ode4' | 'ode3' | 'ode2' | 'ode1' | 'ode14x'`
`SolverMode`	Solver mode for this model. Set by the **Tasking mode for periodic sample times** option on the **Solver** pane of the **Configuration Parameters** dialog box.	`{'Auto'} | 'SingleTasking' | 'MultiTasking'`
`SolverPrmCheckMsg`	Enables diagnostics to control when Simulink automatically selects solver parameters. Set by the **Automatic solver parameter selection** option on the **Diagnostics** pane of the **Configuration Parameters** dialog box. 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**	`'none' | {'warning'} | 'error'`

Model Parameters (Continued)

Parameter	Description	Values
`SolverResetMethod`	Set by the **Solver reset method** option on the **Solver** pane of the **Configuration Parameters** dialog box.	`{'Fast'} | 'Robust'`
`SortedOrder`	Show the sorted order of this model's blocks on the block diagram. Set by the **SortedOrder** command on the model editor's **Format->Block Displays** menu.	`'on' | {'off'}`
`StartFcn`	Start simulation callback.	command or variable `{''}`
`StartTime`	Simulation start time. Set by the **Start time** field on the **Solver** pane of the **Configuration Parameters** dialog box.	string `{'0.0'}`
`StateSaveName`	State output name to be saved to workspace. Set by the **States** field on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	variable `{'xout'}`
`StatusBar`	Show/hide the status bar on the model editor window. Set by the **Status Bar** command on the model editor's **View** menu.	`{'on'} | 'off'`
`StopFcn`	Stop simulation callback.	command or variable `{''}`
`StopTime`	Simulation stop time. Set by the **Stop time** field on the **Solver** pane of the **Configuration Parameters** dialog box.	string `{'10.0'}`
`TargetFcnLib`	For internal use.	

Model Parameters (Continued)

Parameter	Description	Values
`TimeSaveName`	Simulation time name. Set by the **Time** field on the **Data Import/Export** pane of the **Configuration Parameters** dialog box.	variable `{'tout'}`
`TLC...`	Parameters whose names begin with TLC are used for code generation. See the Real-Time Workshop documentation for more information.	
`Toolbar`	Show/hide the toolbar on the Model Editor window. Set by the **Toolbar** command on the model editor's **View** menu.	`{'on'} | 'off'`
`TryForcingSFcnDF`	This flag is used for backward compatibility with user S-functions that were written prior to R12.	`'on' | {'off'}`
`TunableVars`	List of global (tunable) parameters. Set in the **Model Parameter Configuration** dialog box.	string `{''}`
`TunableVarsStorageClass`	List of storage classes for their respective tunable parameters. Set in the **Model Parameter Configuration** dialog box.	string `{''}`
`TunableVarsTypeQualifier`	List of storage type qualifiers for their respective tunable parameters. Set in the **Model Parameter Configuration** dialog box.	string `{''}`

Model Parameters (Continued)

Parameter	Description	Values
`UnconnectedInputMsg`	Unconnected input ports diagnostic. Set by the **Unconnected block input ports** option on the **Connectivity Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | {'warning'} | 'error'`
`UnconnectedLineMsg`	Unconnected lines diagnostic. Set by the **Unconnected line** option on the **Connectivity Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | {'warning'} | 'error'`
`UnconnectedOutputMsg`	Unconnected block output ports diagnostic. Set by the **Unconnected block output ports** option on the **Connectivity Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | {'warning'} | 'error'`
`UnderSpecifiedDataTypeMsg`	Detect usage of heuristics to assign signal data types. Set by the **Underspecified data types** option on the **Data Integrity Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'none'} | 'warning' | 'error'`
`UnknownTsInhSupMsg`	Detect blocks that have not set whether they allow the model containing them to inherit a sample time. Set by the **Unspecified inheritability of sample time** option on the **Solver Diagnostics** pane of the **Configuration Parameters** dialog box.	`'none' | {'warning'} | 'error'`

Model Parameters (Continued)

Parameter	Description	Values
`UnnecessaryDataType ConvMsg`	Detect unnecessary data type conversion blocks. Set by the **Unnecessary type conversions** option on the **Conversion Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'none'} | 'warning'`
`UpdateHistory`	Specifies when to prompt the user about updating the model history. This is set by the **Prompt to update model history** option on the **History** pane of the **Model Explorer** dialog box.	`{'UpdateHistoryNever'} | 'UpdateHistoryWhenSave'`
`UpdateModelReference Targets`	Rebuilding options. Set on the **Model Referencing** pane of the **Configuration Parameters** dialog box.	`'IfOutOfDate' | 'Force' | 'AssumeUpToDate' | {'IfOutOfDateOr Structural Change'}`
`UseAnalysisPorts`	For internal use.	
`VectorMatrix ConversionMsg`	Detect vector-to-matrix or matrix-to-vector conversions. Set by the **Vector/matrix block input conversion** option on the **Conversion Diagnostics** pane of the **Configuration Parameters** dialog box.	`{'none'} | 'warning' | 'error'`
`Version`	Simulink version used to modify the model (read only).	release version number
`WideLines`	For internal use.	
`WideVectorLines`	Wide Vector Lines menu option.	`'on' | {'off'}`
`ZeroCross`	For internal use.	

Model Parameters (Continued)

Parameter	Description	Values
`ZeroCrossControl`	Enable zero-crossing detection. Set by the **Zero crossing control** control on the **Solver** pane of the **Configuration Parameters** dialog box.	`{'UseLocalSettings'} | 'EnableAll' | 'DisableAll'`
`ZoomFactor`	Zoom factor of the model editor window expressed as a percentage of normal (100%) or by the keywords `FitSystem` or `FitSelection`. Set by the zoom commands on the model editor's **View** menu.	`string {'100'} | 'FitSystem' | 'FitSelection'`

Examples of Setting Model Parameters

These examples show how to set model parameters for the `mymodel` system.

This command sets the simulation start and stop times.

```
set_param('mymodel','StartTime','5','StopTime','100')
```

This command sets the solver to `ode15s` and changes the maximum order.

```
set_param('mymodel','Solver','ode15s','MaxOrder','3')
```

This command associates a `SaveFcn` callback.

```
set_param('mymodel','SaveFcn','my_save_cb')
```

Common Block Parameters

This table lists the parameters common to all Simulink blocks, including block callback parameters (see "Using Callback Routines"). Examples of commands that change these parameters follow this table (see “Examples of Setting Block Parameters” on page 10-50).

Common Block Parameters

Parameter	Description	Values
`AncestorBlock`	Name of the library block that the block is linked to (for blocks with a disabled link).	string
`AttributesFormat String`	String format specified for block annotations in the **Block Properties** dialog box.	string
`BackgroundColor`	Block background color.	RGB value array string | `[r,g,b,a]` where `r, g, b,` and `a` are the red, green, blue, and alpha values of the color normalized to the range 0.0 to 1.0. The alpha value is ignored.
`BlockDescription`	Block description shown in the **Block Properties** dialog box.	string
`BlockType`	Block type (read only).	string
`ClipboardFcn`	Function called when block is copied to the clipboard (**Ctrl+C**)	string
`CloseFcn`	Function called when `close_system` is run on block.	string

Common Block Parameters (Continued)

Parameter	Description	Values
`CompiledPort ComplexSignals`	Complexity of port signals after updating diagram.	
`CompiledPortData Types`	Data types of port signals after updating diagram.	
`CompiledPort Dimensions`	Dimensions of port signals after updating diagram.	
`CompiledPortFrame Data`	Frame mode of port signals after updating diagram.	
`CompiledPortWidths`	Structure of port widths after updating diagram.	
`CompiledSampleTime`	Block sample time after updating diagram.	
`CopyFcn`	Function called when block is copied.	string
`DataTypeOverride_ Compiled`	For internal use.	
`DeleteFcn`	Function called when block is deleted.	MATLAB expression
`DestroyFcn`	Function called when block is destroyed.	MATLAB expression
`Description`	Description of block. Set by the **Description** field in the **General** pane of the **Block Properties** dialog box.	text and tokens
`Diagnostics`		text and tokens
`DialogParameters`	Names/attributes of parameters in block's parameter dialog box.	structure
`DropShadow`	Display drop shadow.	`{'off'} | 'on'`

Common Block Parameters (Continued)

Parameter	Description	Values
`ExtModeUpload Option`		`{'none'} | 'log' | 'monitor'`
`ExtModeLogging Supported`		`{'off'} | 'on'`
`ExtModeLogging Trig`		`{'off'} | 'on'`
`FontAngle`	Font angle.	`'normal' | 'italic' | 'oblique' | {'auto'}`
`FontName`	Font.	string
`FontSize`	Font size. A value of -1 specifies that this block inherits the font size specified by the `DefaultBlockFontSize` model parameter.	real `{'-1'}`
`FontWeight`	Font weight.	`'light' | 'normal' | 'demi' | 'bold' | {'auto'}`
`ForegroundColor`	Foreground color of block's icon.	string `{'black'} | [r,g,b,a]` where `r`, `g`, `b`, and `a` are the red, green, blue, and alpha values of the color normalized to the range 0.0 to 1.0. The alpha value is ignored.
`Handle`	Block handle.	real

Common Block Parameters (Continued)

Parameter	Description	Values
`HiliteAncestors`		`{'none'} | 'error' | 'redWhite' | 'blueWhite' | 'greenWhite' | 'orangeWhite' | 'blackWhite' | 'find' | 'debug' | 'unique' | 'different' | 'lineTrace' | 'fade' | 'default' | 'user1' | 'user2' | 'user3' | 'user4' | 'user5'`
`InitFcn`	Initialization function for a masked block.	MATLAB expression
`InputSignalNames`	Names of input signals.	cell array
`IOSignalStrings`		list
`IOType`		`{'none'} | 'viewer' | 'siggen'`
`LineHandles`	Handles of lines connected to block.	struct
`LinkStatus`	Link status of block.	`{'none'} | 'resolved' | 'unresolved' | 'implicit' | 'inactive' | 'restore' | 'propagate'`
`LoadFcn`	Function called when block is loaded.	MATLAB expression
`MinMaxOverflow Logging_Compiled`	For internal use.	
`ModelCloseFcn`	Function called when model is closed.	MATLAB expression
`ModelParamTable Info`	For internal use.	

Common Block Parameters (Continued)

Parameter	Description	Values
`MoveFcn`	Function called when block is moved.	MATLAB expression
`Name`	Block name.	string
`NameChangeFcn`	Function called when block name is changed.	MATLAB expression
`NamePlacement`	Position of block name.	`{'normal'} | 'alternate'`
`ObjectParameters`	Names/attributes of block's parameters.	structure
`OpenFcn`	Function called when this block's **Block Parameters** dialog box is opened.	MATLAB expression
`Orientation`	Where block faces.	`{'right'} | 'left' | 'up' | 'down'`
`OutputSignalNames`	Names of output signals.	cell array
`Parent`	Name of the system that owns the block.	string `{'untitled'}`
`ParentCloseFcn`	Function called when parent subsystem is closed.	MATLAB expression
`PortConnectivity`	The value of this parameter is an array of structures, each of which describes one of the block's input or output ports. Each port structure has the following fields: • `Type` Specifies the port's type and/or number. The value of this field can be: ▪ *n*, where *n* is the umber of the port for data ports	structure array

Common Block Parameters (Continued)

Parameter	Description	Values
	- `'enable'` if the port is an enable port - `'trigger'` if the port is a trigger port - `'state'` for state ports - `'ifaction'` for action ports - `'LConn#'` for a left connection port where # is the port's number - `'RConn#'` for a right connection port where # is the port's number • `Position` The value of this field is a two-element vector, [*x* *y*], that specifies the port's position. • `SrcBlock` Handle of the block connected to this port. This field is null for output ports. • `SrcPort` Number of the port connected to this port. This field is null for output ports. • `DstBlock`	

Common Block Parameters (Continued)

Parameter	Description	Values
	Handle of the block to which this port is connected. This field is null for input ports. • `DstPort` Number of the port to which this port is connected. This field is null for input ports.	
`PortHandles`	The value of this parameter is a structure that specifies the handles of the block's ports. The structure has the following fields: • `Inport` Handles of the block's input ports. • `Outport` Handles of the block's output ports. • `Enable` Handle of the block's enable port. • `Trigger` Handle of the block's trigger port. • `State` Handle of the block's state port. • `LConn`	structure array

Common Block Parameters (Continued)

Parameter	Description	Values
	Handles of the block's left connection ports. • `RConn` Handles of the block's right connection ports. • `Ifaction` Handle of the block's action port.	
`Ports`	The value of this parameter is a vector that specifies the numbers of each kind of port. The order of the vector's elements corresponds to the following port types: • `Inport` • `Outport` • `Enable` • `Trigger` • `State` • `LConn` • `RConn` • `Ifaction`	vector
`Position`	Position of block in model window.	vector `[left top right bottom]` not enclosed in quotation marks. The maximum value for a coordinate is 32767.

Common Block Parameters (Continued)

Parameter	Description	Values
`PostSaveFcn`	Function called after the block is saved.	MATLAB expression
`PreSaveFcn`	Function called before the block is saved.	MATLAB expression
`Priority`	Specifies the block's order of execution relative to other blocks in the same model. Set by the **Priority** field on the **General** pane of the **Block Properties** dialog box.	string `{''}`
`ReferenceBlock`	Name of the library block that this block is linked to.	string `{''}`
`RequirementInfo`	For internal use.	
`RTWData`	User specified data, used by Real-Time Workshop.	
`SampleTime`	Value of the sample time parameter.	
`Selected`	Status of whether or not block is selected.	`{'on'} | 'off'`
`ShowName`	Display block name.	`{'on'} | 'off'`
`StartFcn`	Function called at the start of a simulation.	MATLAB expression
`StatePerturbation ForJacobian`	See the link Control documentation for details.	
`StopFcn`	Function called at the termination of a simulation.	MATLAB expression

Common Block Parameters (Continued)

Parameter	Description	Values
`Tag`	Text that appears in the block label that Simulink generates. Set by the **Tag** field on the **General** pane of the **Block Properties** dialog box.	string `{''}`
`Type`	Simulink object type (read only).	`'block'`
`UndoDeleteFcn`	Function called when block deletion is undone.	MATLAB expression
`UserData`	User-specified data that can have any MATLAB data type.	`{'[]'}`
`UserDataPersistent`	Status of whether or not `UserData` will be saved in the model file.	`'on' | {'off'}`

Examples of Setting Block Parameters

These examples illustrate how to change common block parameters.

This command changes the orientation of the Gain block in the `mymodel` system so it faces the opposite direction (right to left).

```
set_param('mymodel/Gain','Orientation','left')
```

This command associates an `OpenFcn` callback with the Gain block in the `mymodel` system.

```
set_param('mymodel/Gain','OpenFcn','my_open_cb')
```

This command sets the `Position` parameter of the Gain block in the `mymodel` system. The block is 75 pixels wide by 25 pixels high. The position vector is *not* enclosed in quotation marks.

```
set_param('mymodel/Gain','Position',[50 250 125 275])
```

Block-Specific Parameters

These tables list block-specific parameters for all Simulink blocks. The type of the block appears in parentheses after the block name. Some Simulink blocks are implemented as masked subsystems. The tables indicate masked blocks by adding the designation "masked" after the block type.

Note The type listed for nonmasked blocks is the value of the block's `BlockType` parameter; the type listed for masked blocks is the value of the block's `MaskType` parameter. For more information, see "Mask Parameters" on page 10-145.

The **Dialog Box Prompt** column indicates the text of the prompt for the parameter on the block's dialog box. The **Values** column shows the type of value required (scalar, vector, variable), the possible values (separated with a vertical line), and the default value (enclosed in braces).

Continuous Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Derivative (`Derivative`)		
`LinearizePole`	Linearization Time Constant s/(Ns+1)	string `{'inf'}`
Integrator (`Integrator`)		
`ExternalReset`	External reset	`{'none'} | 'rising' | 'falling' | 'either' | 'level'`
`InitialConditionSource`	Initial condition source	`{'internal'} | 'external'`
`InitialCondition`	Initial condition	scalar or vector {'0'}
`LimitOutput`	Limit output	`{'off'} | 'on'`
`UpperSaturationLimit`	Upper saturation limit	scalar or vector `{'inf'}`
`LowerSaturationLimit`	Lower saturation limit	scalar or vector `{'-inf'}`

Continuous Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
ShowSaturationPort	Show saturation port	{'off'} | 'on'
ShowStatePort	Show state port	{'off'} | 'on'
AbsoluteTolerance	Absolute tolerance	string {'auto'}
ZeroCross	Enable zero–crossing detection	'off' | {'on'}
State-Space (StateSpace)		
A	A	matrix {'1'}
B	B	matrix {'1'}
C	C	matrix {'1'}
D	D	matrix {'1'}
X0	Initial conditions	vector {'0'}
AbsoluteTolerance	Absolute tolerance	string {'auto'}
Transfer Fcn (TransferFcn)		
Numerator	Numerator	vector or matrix {'[1]'}
Denominator	Denominator	vector {'[1 1]'}
AbsoluteTolerance	Absolute tolerance	string {'auto'}
Transport Delay (TransportDelay)		
DelayTime	Time delay	scalar or vector {'1'}
InitialInput	Initial input	scalar or vector {'0'}
BufferSize	Initial buffer size	scalar {'1024'}
PadeOrder	Pade order (for linearization)	string {'0'}
TransDelayFeedthrough	Direct feedthrough of input during linearization	{'off'} | 'on'
Variable Transport Delay (VariableTransportDelay)		
MaximumDelay	Maximum delay	scalar or vector {'10'}
InitialInput	Initial input	scalar or vector {'0'}

Continuous Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`MaximumPoints`	Buffer size	scalar `{'1024'}`
`PadeOrder`	Pade order (for linearization)	string `{'0'}`
`TransDelayFeedthrough`	Direct feedthrough of input during linearization	`{'off'} | 'on'`
Zero-Pole (`ZeroPole`)		
`Zeros`	Zeros	vector `{'[1]'}`
`Poles`	Poles	vector `{'[0 -1]'}`
`Gain`	Gain	vector `{'[1]'}`
`AbsoluteTolerance`	Absolute tolerance	string `{'auto'}`

Discontinuities Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Backlash (`Backlash`)		
`BacklashWidth`	Deadband width	scalar or vector `{1}`
`InitialOutput`	Initial output	scalar or vector `{0}`
`ZeroCross`	Enable zero crossing detection	`'off' | {'on'}`
`SampleTime`	Sample time (-1 for inherited)	string `{'-1'}`
Coulomb & Viscous Friction (`Coulombic and Viscous Friction`) (masked subsystem)		
`offset`	Coulomb friction value (Offset)	string `{'[1 3 2 0]'}`
`gain`	Coefficient of viscous friction (Gain)	string `{'1'}`
Dead Zone (`DeadZone`)		
`LowerValue`	Start of dead zone	scalar or vector `{-0.5}`
`UpperValue`	End of dead zone	scalar or vector `{0.5}`

Discontinuities Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SaturateOnInteger Overflow	Saturate on integer overflow	'off' | {'on'}
LinearizeAsGain	Treat as gain when linearizing	'off' | {'on'}
ZeroCross	Enable zero crossing detection	'off' | {'on'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Dead Zone Dynamic (Dead Zone Dynamic) (masked subsystem)		
Hit Crossing (HitCross)		
HitCrossingOffset	Hit crossing offset	scalar or vector {'0'}
HitCrossingDirection	Hit crossing direction	'rising' | 'falling' | {'either'}
ShowOutputPort	Show output port	{'on'} | 'off'
ZeroCross	Enable zero crossing detection	'off' | {'on'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Quantizer (Quantizer)		
QuantizationInterval	Quantization interval	scalar or vector {'0.5'}
LinearizeAsGain	Treat as gain when linearizing	'off' | {'on'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Rate Limiter (RateLimiter)		
RisingSlewLimit	Rising slew rate	string {'1'}
FallingSlewLimit	Falling slew rate	string {'-1'}
SampleTimeMode	Sample time mode	'continuous' | {'inherited'}
InitialCondition	Initial condition	string {'0'}
LinearizeAsGain	Treat as gain when linearizing	'off' | {'on'}
Rate Limiter Dynamic (Rate Limiter Dynamic) (masked subsystem)		
Relay (Relay)		

Discontinuities Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OnSwitchValue`	Switch on point	string `{'eps'}`
`OffSwitchValue`	Switch off point	string `{'eps'}`
`OnOutputValue`	Output when on	string `{'1'}`
`OffOutputValue`	Output when off	string `{'0'}`
`OutputDataTypeScaling Mode`	All ports same datatype	`'Specify via dialog' | 'Inherit via back propagation' | {'All ports same datatype'}`
`OutDataType`	Output data type (e.g., sfix(16), uint(8), float(`'single'`))	string `{'sfix(16)'}`
`OutScaling`	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string `{'2^0'}`
`ConRadixGroup`	Parameter scaling mode	`{'Use specified scaling'} | 'Best Precision: Vector-wise'`
`ZeroCross`	Enable zero crossing detection	`'off' | {'on'}`
`SampleTime`	Sample time (-1 for inherited)	string `{'-1'}`
Saturation (`Saturate`)		
`UpperLimit`	Upper limit	scalar or vector `{'0.5'}`
`LowerLimit`	Lower limit	scalar or vector `{'-0.5'}`
`LinearizeAsGain`	Treat as gain when linearizing	`'off' | {'on'}`
`ZeroCross`	Enable zero crossing detection	`'off' | {'on'}`
`SampleTime`	Sample time (-1 for inherited)	string `{'-1'}`
Saturation Dynamic (`Saturation Dynamic`) (masked subsystem)		

Discontinuities Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Wrap To Zero (`Wrap To Zero`) (masked subsystem)		
`Threshold`	Threshold	string `{'255'}`

Discrete Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Difference (`Difference`) (masked subsystem)		
`ICPrevInput`	Initial condition for previous input	string `{'0.0'}`
`OutputDataTypeScaling Mode`	Output data type and scaling	`'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation'`
`OutDataType`	Output data type: ex. sfix(16), uint(8), float(`'single'`)	string `{'sfix(16)'}`
`OutScaling`	Output scaling: Slope or [Slope Bias] ex. 2^-9	string `{'2^-10'}`
`LockScale`	Lock output scaling so autoscaling tool can't change it	`{'off'} | 'on'`
`RndMeth`	Round toward	`'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}`
`DoSatur`	Saturate to max or min when overflows occur	`{'off'} | 'on'`
Discrete Derivative (`Discrete Derivative`) (masked subsystem)		
`gainval`	Gain value	string `{'1.0'}`
`ICPrevScaledInput`	Initial condition for previous weighted input K*u/Ts	string `{'0.0'}`

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutputDataTypeScaling Mode	Output data type and scaling	'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation'
OutDataType	Output data type: ex. sfix(16), uint(8), float('single')	string {'sfix(16)'}
OutScaling	Output scaling: Slope or [Slope Bias] ex. 2^-10	string {'2^-10'}
LockScale	Lock output scaling so autoscaling tool can't change it	{'off'} | 'on'
RndMeth	Round toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
DoSatur	Saturate to max or min when overflows occur	{'off'} | 'on'
Discrete Filter (DiscreteFilter)		
Numerator	Numerator	vector {'[1]'}
Denominator	Denominator	vector {'[1 0.5]'}
SampleTime	Sample time (-1 for inherited)	string {'1'}
StateIdentifier	State name	string {}
StateMustResolveTo SignalObject	State name must resolve to Simulink signal object	{'off'} | 'on'
RTWStateStorageClass	RTW storage class	{'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'
RTWStateStorageType Qualifier	RTW storage type qualifier	string {}
Discrete State-Space (DiscreteStateSpace)		
A	A	string {'1'}

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
B	B	string {'1'}
C	C	string {'1'}
D	D	string {'1'}
X0	Initial conditions	string {'0'}
SampleTime	Sample time	string {'1'}
StateIdentifier	State name	string {}
StateMustResolveTo SignalObject	State name must resolve to Simulink signal object	{'off'} | 'on'
RTWStateStorageClass	RTW storage class	{'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'
RTWStateStorageType Qualifier	RTW storage type qualifier	string {}
Discrete Transfer Fcn (DiscreteTransferFcn)		
Numerator	Numerator	vector {'[1]'}
Denominator	Denominator	vector {'[1 0.5]'}
SampleTime	Sample time (-1 for inherited)	string {'1'}
StateIdentifier	State name	string {}
StateMustResolveTo SignalObject	State name must resolve to Simulink signal object	{'off'} | 'on'
RTWStateStorageClass	RTW storage class	{'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'
RTWStateStorageType Qualifier	RTW storage type qualifier	string {}
Discrete Zero-Pole (DiscreteZeroPole)		

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Zeros	Zeros	vector {'[1]'}
Poles	Poles	vector {'[0 0.5]'}
Gain	Gain	string {'1'}
SampleTime	Sample time (-1 for inherited)	string {'1'}
StateIdentifier	State name	string {}
StateMustResolveTo SignalObject	State name must resolve to Simulink signal object	{'off'} | 'on'
RTWStateStorageClass	RTW storage class	{'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'
RTWStateStorageType Qualifier	RTW storage type qualifier	string {}
Discrete-Time Integrator (DiscreteIntegrator)		
IntegratorMethod	Integrator method	{'Integration: Forward Euler'} | 'Integration: Backward Euler' | 'Integration: Trapezoidal' | 'Accumulation: Forward Euler' | 'Accumulation: Backward Euler' | 'Accumulation: Trapezoidal'
gainval	Gain value	string {'1.0'}
ExternalReset	External reset	{'none'} | 'rising' | 'falling' | 'either' | 'level'
InitialConditionSource	Initial condition source	{'internal'} | 'external'
InitialCondition	Initial condition	scalar or vector {'0'}

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
InitialConditionMode	Use initial condition as initial and reset value for	'State only (most efficient)' | {'State and output'}
SampleTime	Sample time (-1 for inherited)	string {'1'}
OutDataTypeMode	Output data type mode	'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
LockScale	Lock output scaling against changes by the autoscaling tool	{'off'} | 'on'
RndMeth	Round integer calculations toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	{'off'} | 'on'
LimitOutput	Limit output	{'off'} | 'on'
UpperSaturationLimit	Upper saturation limit	scalar or vector {inf}
LowerSaturationLimit	Lower saturation limit	scalar or vector {-inf}
ShowSaturationPort	Show saturation port	{'off'} | 'on'
ShowStatePort	Show state port	{'off'} | 'on'

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
StateIdentifier	State name	string {}
StateMustResolveTo SignalObject	State name must resolve to Simulink signal object	{'off'} | 'on'
RTWStateStorageClass	RTW storage class	{'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'
RTWStateStorageType Qualifier	RTW storage type qualifier	string {}
First-Order Hold (First-Order Hold) (masked subsystem)		
Ts	Sample time	string {'1'}
Integer Delay (S-Function) (Integer Delay) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
samptime	Sample time	string {'-1'}
NumDelays	Number of delays	string {'4'}
Memory (Memory)		
X0	Initial condition	scalar or vector {'0'}
InheritSampleTime	Inherit sample time	{'off'} | 'on'
LinearizeMemory	Direct feedthrough of input during linearization	{'off'} | 'on'
StateIdentifier	State name	string {}
StateMustResolveTo SignalObject	State name must resolve to Simulink signal object	{'off'} | 'on'
RTWStateStorageClass	RTW storage class	{'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RTWStateStorageType Qualifier	RTW storage type qualifier	string {}
Tapped Delay (S-Function) (Tapped Delay Line) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
samptime	Sample time	string {'-1'}
NumDelays	Number of delays	string {'4'}
DelayOrder	Order output vector starting with	{'Oldest'} | 'Newest'
includeCurrent	Include current input in output vector	{'off'} | 'on'
Transfer Fcn (First Order Transfer Fcn) (masked subsystem)		
PoleZ	Pole (in Z plane)	string {'0.95'}
ICPrevOutput	Initial condition for previous output	string {'0.0'}
RndMeth	Round toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
DoSatur	Saturate to max or min when overflows occur	{'off'} | 'on'
Transfer Fcn Lead or Lag (Lead or Lag Compensator) (masked subsystem)		
PoleZ	Pole of compensator (in Z plane)	string {'0.95'}
ZeroZ	Zero of compensator (in Z plane)	string {'0.75'}
ICPrevOutput	Initial condition for previous output	string {'0.0'}
ICPrevInput	Initial condition for previous input	string {'0.0'}

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RndMeth	Round toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
DoSatur	Saturate to max or min when overflows occur	{'off'} | 'on'
Transfer Fcn Real Zero (Transfer Fcn Real Zero) (masked subsystem)		
ZeroZ	Zero (in Z plane)	string {'0.75'}
ICPrevInput	Initial condition for previous input	string {'0.0'}
RndMeth	Round toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
DoSatur	Saturate to max or min when overflows occur	{'off'} | 'on'
Unit Delay (UnitDelay)		
X0	Initial condition	scalar or vector {'0'}
SampleTime	Sample time (-1 for inherited)	string {'1'}
StateIdentifier	State name	string {}
StateMustResolveTo SignalObject	State name must resolve to Simulink signal object	{'off'} | 'on'
RTWStateStorageClass	RTW storage class	{'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'
RTWStateStorageType Qualifier	RTW storage type qualifier	string {}
Weighted Moving Average (S-Function) (Weighted Moving Average) (masked subsystem)		
mgainval	Weights	string {'[0.1:0.1:1 0.9:-0.1:0.1]'}
vinit	Initial condition	string {'0.0'}

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`samptime`	Sample time	string `{'-1'}`
`GainDataTypeScalingMode`	Gain data type and scaling	`'Specify via dialog' | {'Inherit via internal rule'}`
`GainDataType`	Parameter data type: ex. sfix(16), uint(8), float('single')	string `{'sfix( 16 )'}`
`MatRadixGroup`	Parameter scaling mode	`'Use Specified Scaling' | 'Best Precision: Element-wise' | 'Best Precision: Row-wise' | 'Best Precision: Column-wise' | {'Best Precision: Matrix-wise'}`
`GainScaling`	Parameter scaling: Slope ex. 2^-9	string `{'2^-10'}`
`OutputDataTypeScaling Mode`	Output data type and scaling	`'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation'`
`OutDataType`	Output data type: ex. sfix(16), uint(8), float (`'single'`)	string `{'sfix(16)'}`
`OutScaling`	Output scaling: Slope or [Slope Bias] ex. 2^-9	string `{'2^-10'}`
`LockScale`	Lock output scaling so autoscaling tool can't change it	`{'off'} | 'on'`
`RndMeth`	Round toward	`'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}`
`DoSatur`	Saturate to max or min when overflows occur	`{'off'} | 'on'`

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Zero-Order Hold (`ZeroOrderHold`)		
`SampleTime`	Sample time (-1 for inherited)	string `{'1'}`

Logic and Bit Operations Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Bit Clear (`Bit Clear`) (masked subsystem)		
`iBit`	Index of bit (0 is least significant)	string `{'0'}`
Bit Set (`Bit Set`) (masked subsystem)		
`iBit`	Index of bit (0 is least significant)	string `{'0'}`
Bitwise Operator (S-Function) (`Bitwise Operator`) (masked subsystem)		
`logicop`	Operator	`{'AND'} | 'OR' | 'NAND' | 'NOR' | 'XOR' | 'NOT'`
`UseBitMask`	Use bit mask ...	`'off' | {'on'}`
`NumInputPorts`	Number of input ports	string `{'1'}`
`BitMask`	Bit mask	string `{'bin2dec('11011001')'}`
`BitMaskRealWorld`	Treat mask as	`'Real World Value' | {'Stored Integer'}`
Combinatorial Logic (`CombinatorialLogic`)		
`TruthTable`	Truth table	string `{'[0 0;0 1;0 1;1 0;0 1;1 0;1 0;1 1]'}`
`SampleTime`	Sample time (-1 for inherited	string `{'-1'}`
Compare To Constant (`Compare To Constant`) (masked subsystem)		
`relop`	Operator	`'==' | '~=' | '<' | {'<='} | '>=' | '>'`

Logic and Bit Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
const	Constant value	string {'3.0'}
LogicOutDataTypeMode	Output data type mode	{'uint8'} | 'boolean'
Compare To Zero (Compare To Zero) (masked subsystem)		
relop	Operator	'==' | '~=' | '<' | {'<='} | '>=' | '>'
LogicOutDataTypeMode	Output data type mode	{'uint8'} | 'boolean'
Detect Change (Detect Change) (masked subsystem)		
vinit	Initial condition	string {'0'}
Detect Decrease (Detect Decrease) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
Detect Fall Negative (Detect Fall Negative) (masked subsystem)		
vinit	Initial condition	string {'0'}
Detect Fall Nonpositive (Detect Fall Nonpositive) (masked subsystem)		
vinit	Initial condition	string {'0'}
Detect Increase (Detect Increase) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
Detect Rise Nonnegative (Detect Rise Nonnegative) (masked subsystem)		
vinit	Initial condition	string {'0'}
Detect Rise Positive (Detect Rise Positive) (masked subsystem)		
vinit	Initial condition	string {'0'}
Extract Bits (Extract Bits) (masked subsystem)		
bitsToExtract	Bits to extract	{'Upper half'} | 'Lower half' | 'Range starting with most | significant bit' | 'Range ending with least significant bit' | 'Range of bits'

Logic and Bit Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
numBits	Number of bits	string {'8'}
bitIdxRange	Bit indices ([start end], 0-based relative to LSB)	string {'[0 7]'}
outScalingMode	Output scaling mode	{'Preserve fixed-point scaling'} | 'Treat bit field as an integer'
Interval Test (Interval Test) (masked subsystem)		
IntervalClosedRight	Interval closed on right	'off' | {'on'}
uplimit	Upper limit	string {'0.5'}
IntervalClosedLeft	Interval closed on left	'off' | {'on'}
lowlimit	Lower limit	string {'-0.5'}
LogicOutDataTypeMode	Output data type mode	'uint8' | {'boolean'}
Interval Test Dynamic (Interval Test Dynamic) (masked subsystem)		
IntervalClosedRight	Interval closed on right	'off' | {'on'}
IntervalClosedLeft	Interval closed on left	'off' | {'on'}
LogicOutDataTypeMode	Output data type mode	'uint8' | {'boolean'}
Logical Operator (Logic)		
Operator	Operator	{'AND'} | 'OR' | 'NAND' | 'NOR' | 'XOR' | 'NOT'
Inputs	Number of input ports	string {'2'}
AllPortsSameDT	Require all inputs and output to have same data type	{'off'} | 'on'
OutDataTypeMode	Output data type mode	{'Boolean'} | 'Logical (see Advanced Sim. Parameters)' | 'Specify via dialog'
LogicDataType	Output data type (e.g., uint(8), sint(32))	string {'uint(8)'}

Logic and Bit Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Relational Operator (RelationalOperator)		
Operator	Relational Operator	'==' | '~=' | '<' | {'<='} | '>=' | '>'
InputSameDT	Require all inputs to have same data type	{'off'} | 'on'
LogicOutDataTypeMode	Output data type mode	'uint8' | {'boolean'}
LogicDataType	Output data type (e.g., uint(8), sint(32))	string {'uint(8)'}
ZeroCross	Enable zero crossing detection	'off' | {'on'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Shift Arithmetic (Shift Arithmetic) (masked subsystem)		
nBitShiftRight	Number of bits to shift right (use negative value to shift left)	string {'0'}
nBinPtShiftRight	Number of places by which binary point shifts right (use negative value to shift left)	string {'0'}

Lookup Tables Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Cosine (Cosine) (masked subsystem)		
NumDataPoints	Number of data points for lookup table	string {'(2^5)+1'}
Direct Lookup Table (n-D) (S-Function) (LookupNDDirect) (masked subsystem)		
maskTabDims	Number of table dimensions	'1' | {'2'} | '3' | '4' | 'More...'

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`explicitNumDims`	Explicit number of table dimensions	string `{'1'}`
`outDims`	Inputs select this object from table	`{'Element'} | 'Column' | '2-D Matrix'`
`tabIsInput`	Make table an input	`{'off'} | 'on'`
`mxTable`	Table data	string `{'[4 5 6;16 19 20;10 18 23]'}`
`clipFlag`	Action for out of range input	`'None' | {'Warning'} | 'Error'`
Interpolation (n-D) using PreLookup (`LookupNDInterpIdx`) (masked subsystem)		
`numDimsPopupSelect`	Number of table dimensions	`'1' | {'2'} | '3' | '4' | 'More...'`
`explicitNumDims`	Explicit number of table dimensions	string `{'2'}`
`table`	Table data	string `{'sqrt([1:10]'*[1:10])'}`
`interpMethod`	Interpolation method	'None - Flat' | `{'Linear'}`
`extrapMethod`	Extrapolation method	`'None - Flat' | {'Linear'}`
`rangeErrorMode`	Action for out of range input	`{'None'} | 'Warning' | 'Error' | 'Error - No index checking in generated code' | 'Warning - No index checking in generated code' | 'None - No index checking in generated code'`
`NumSelectionDims`	Number of sub-table selection dimensions	string `{'0'}`
Lookup Table (`Lookup`)		
`InputValues`	Vector of input values	`vector {'[-5:5]'}`

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutputValues	Vector of output values	vector {'tanh([-5:5])'}
LookUpMeth	Look-up method	{'Interpolation-Extrapolation'} | 'Interpolation-Use End Values' | 'Use Input Nearest' | 'Use Input Below' | 'Use Input Above'
OutDataTypeMode	Output data type mode	'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'Specify via dialog' | 'Inherit via back propagation' | {'Same as input'}
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
LockScale	Lock output scaling against changes by the autoscaling tool	{'off'} | 'on'
RndMeth	Round integer calculations toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	{'off'} | 'on'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Lookup Table (2-D) (Lookup2D)		
RowIndex	Row index input values	string {'[1:3]'}
ColumnIndex	Column index input values	string {'[1:3]'}

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutputValues	Matrix of output values	string {'[4 5 6;16 19 20;10 18 23]'}
LookUpMeth	Lookup method	{'Interpolation-Extrapolation'} | 'Interpolation-Use End Values' | 'Use Input Nearest' | 'Use Input Below' | 'Use Input Above'
InputSameDT	Require all inputs to have same data type	'on' | {'off'}
OutDataTypeMode	Output data type mode	'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'Specify via dialog' | 'Inherit via back propagation' | {'Same as first input'}
OutDataType	Output data type (e.g., sfix(16), uint(8), float ('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
LockScale	Lock output scaling against changes by the autoscaling tool	'on' | {'off'}
RndMeth	Round integer calculations toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on' | {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Lookup Table (n-D) (`LookupNDInterp`) (masked subsystem)		
`numDimsPopupSelect`	Number of table dimensions	`' 1 ' | {' 2 '} | ' 3 ' | ' 4 ' | 'More...'`
`bp1`	First input (row) breakpoint set	string `{'[10,22,31]'}`
`bp2`	Second (column) input breakpoint set	string `{'[10,22,31]'}`
`bp3`	Third input breakpoint set	string `{'[1:3]'}`
`bp4`	Fourth input breakpoint set	string `{'[1:3]'}`
`bpcell`	Fifth...Nth breakpoint sets (cell array)	string `{'{ [1:3], [1:3] }'}`
`explicitNumDims`	Explicit number of dimensions	string `{'2'}`
`searchMode`	Index search method	`'Evenly Spaced Points' | 'Linear Search' | {'Binary Search'}`
`cacheBpFlag`	Begin index searches using previous index results	`'on' | {'off'}`
`vectorInputFlag`	Use one (vector) input port instead of N ports	`'on' | {'off'}`
`tableData`	Table data	string `{'[4 5 6;16 19 20;10 18 23]'}`
`interpMethod`	Interpolation method	`'None - Flat' | {'Linear'} | 'Cubic Spline'`
`extrapMethod`	Extrapolation method	`'None - Clip' | {'Linear'} | 'Cubic Spline'`
`rangeErrorMode`	Action for out of range input	`{'None'} | 'Warning' | 'Error'`

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Lookup Table Dynamic (`Lookup Table Dynamic`) (masked subsystem)		
`LookUpMeth`	Look-Up Method	`'Interpolation-Extrapolation' | {'Interpolation-Use End Values'} | 'Use Input Nearest' | 'Use Input Below' | 'Use Input Above'`
`OutputDataTypeScaling Mode`	Output data type and scaling	`{'Specify via dialog'} | 'Inherit via back propagation'`
`OutDataType`	Output data type: ex. sfix(16), uint(8), float (`'single'`)	string `{'float('double')'}`
`OutScaling`	Output scaling: Slope or [Slope Bias] ex. 2^-9	string `{'2^-10'}`
`LockScale`	Lock output scaling so autoscaling tool can't change it	`'on' | {'off'}`
`RndMeth`	Round toward	`'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}`
`DoSatur`	Saturate to max or min when overflows occur	`'on' | {'off'}`
PreLookup Index Search (`LookupIdxSearch`) (masked subsystem)		
`bpData`	Breakpoint data	string `{'[10:10:100]'}`
`searchMode`	Index search method	`'Evenly Spaced Points' | 'Linear Search' | {'Binary Search'}`
`cacheBpFlag`	Begin index search using previous index result	`'on' | {'off'}`
`outputFlag`	Output only the index	`'on' | {'off'}`
`IndexDataType`	Index data type	`{'uint32'} | 'int32'`

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
extrapMode	Process out of range input	'Clip to Range' | {'Linear Extrapolation'}
rangeErrorMode	Action for out of range input	{'None'} | 'Warning' | 'Error'
Sine (Sine) (masked subsystem)		
NumDataPoints	Number of data points for lookup table	string {'(2^5)+1'}

Math Operations Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Abs (Abs)		
SaturateOnIntegerOverflow	Saturate on integer overflow	'on' | {'off'}
ZeroCross	Enable zero crossing detection	{'on'} | 'off'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Add (Sum)		
IconShape	Icon shape	{'rectangular'} | 'round'
Inputs	List of signs	string {'++'}
InputSameDT	Require all inputs to have same data type	'on' | {'off'}
OutDataTypeMode	Output data type mode	'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation' | 'Same as first input'

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutDataType	Output data type (e.g., sfix(16), uint(8), float ('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^-10'}
LockScale	Lock output scaling against changes by the autoscaling tool	'on' | {'off'}
RndMeth	Round integer calculations toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on' | {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Algebraic Constraint (Algebraic Constraint) (masked subsystem)		
z0	Initial guess	string {'0'}
Assignment (Assignment)		
InputType	Input type	{'Vector'} | 'Matrix'
IndexMode	Index mode	'Zero-based' | {'One-based'}
IndexIsStartValue	Use index as starting value	'on' | {'off'}
ElementSrc	Source of element indices (E)	{'Internal'} | 'External'
Elements	Elements (-1 for all elements)	string {'1'}
RowSrc	Source of row indices (R)	{'Internal'} | 'External'
Rows	Rows (-1 for all rows)	string {'1'}
ColumnSrc	Source of column indices (C)	{'Internal'} | 'External'
Columns	Columns (-1 for all columns)	string {'1'}

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutputInitialize	Output (Y)	'Initialize using input (U1)' | 'Specify required dimensions'
OutputDimensions	Output dimensions	string {'[1 1]'}
DiagnosticForDimensions	Diagnostic if not all required dimensions are populated	{'Error'} | 'Warning' | 'None'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Bias (Bias)		
Bias	Bias	string {'0.0'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on' | {'off'}
Complex to Magnitude-Angle (ComplexToMagnitudeAngle)		
Output	Output	'Magnitude' | 'Angle' | {'Magnitude and angle'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Complex to Real-Imag (ComplexToRealImag)		
Output	Output	'Real' | 'Imag' | {'Real and imag'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Divide (Product)		
Inputs	Number of inputs	string {'*/'}
Multiplication	Multiplication	{'Element-wise(.*)'} | 'Matrix(*)'
InputSameDT	Require all inputs to have same data type	'on' | {'off'}

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutDataTypeMode`	Output data type mode	`'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation' | 'Same as first input'`
`OutDataType`	Output data type (e.g., sfix(16), uint(8), float(`'single'`))	string `{'sfix(16)'}`
`OutScaling`	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string `{'2^-10'}`
`LockScale`	Lock output scaling against changes by the autoscaling tool	`'on' | {'off'}`
`RndMeth`	Round integer calculations toward	`'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}`
`SaturateOnIntegerOverflow`	Saturate on integer overflow	`'on' | {'off'}`
`SampleTime`	Sample time (-1 for inherited)	string `{'-1'}`
Dot Product (`Dot Product`) (masked subsystem)		
`InputSameDT`	Require all inputs to have same data type	`'on' | {'off'}`
`OutputDataTypeScalingMode`	Output data type mode	`'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation' | 'Same as first input'`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^-10'}
LockScale	Lock output scaling against changes by the autoscaling tool	'on' | {'off'}
RndMeth	Round integer calculation toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
DoSatur	Saturate on integer overflow	'on' | {'off'}
Gain (Gain)		
Gain	Gain	string {'1'}
Multiplication	Multiplication	{'Element-wise(K.*u)'} | 'Matrix(K*u)' | 'Matrix(u*K)' | 'Matrix(K*u) (u vector)'
ParameterDataTypeMode	Parameter data type mode	'Specify via dialog' | {'Inherit via internal rule'} | 'Same as input'
ParameterDataType	Parameter data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
ParameterScalingMode	Parameter scaling mode	'Use specified scaling' | 'Best Precision: Element-wise' | 'Best Precision: Row-wise' | 'Best Precision: Column-wise' | {'Best Precision: Matrix-wise'}

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
ParameterScaling	Parameter scaling (Slope or [Slope Bias], e.g., 2^-9)	string {'2^0'}
OutDataTypeMode	Output data type mode	'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation' | 'Same as input'
OutDataType	Output data type (e.g., sfix(16), uint(8), float ('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
LockScale	Lock output scaling against changes by the autoscaling tool	'on' | {'off'}
RndMeth	Round integer calculations toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on' | {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Magnitude-Angle to Complex (MagnitudeAngleToComplex)		
Input	Input	'Magnitude' | 'Angle' | {'Magnitude and angle'}
ConstantPart		string {'0'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Math Function (Math)		

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`Operator`	Function	`{'exp'} | 'log' | '10^u' | 'log10' | 'magnitude^2' | 'square' | 'sqrt' | 'pow' | 'conj' | 'reciprocal' | 'hypot' | 'rem' | 'mod' | 'transpose' | 'hermitian'`
`OutputSignalType`	Output signal type	`{'auto'} | 'real' | 'complex'`
`SampleTime`	Sample time (-1 for inherited)	string `{'-1'}`
`OutDataTypeMode`	Output data type mode	`'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'Specify via dialog' | 'Inherit via internal rule' | 'Inherit via back propagation' | {'Same as first input'}`
`OutDataType`	Output data type (e.g., sfix(16), uint(8), float (`'single'`))	string `{'sfix(16)'}`
`OutScaling`	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string `{'2^0'}`
`LockScale`	Lock output scaling against changes by the autoscaling tool	`'on' | {'off'}`
`RndMeth`	Round integer calculations toward	`'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}`
`SaturateOnInteger Overflow`	Saturate on integer overflow	`{'on'} | 'off'`
Matrix Concatenation (`Matrix Concatenation`) (masked subsystem)		

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`numInports`	Number of inputs	string `{'2'}`
`catMethod`	Concatenation method	`{'Horizontal'} | 'Vertical'`
MinMax (`MinMax`)		
`Function`	Function	`{'min'} | 'max'`
`Inputs`	Number of input ports	string `{'1'}`
`InputSameDT`	Require all inputs to have same data type	`'on' | {'off'}`
`OutDataTypeMode`	Output data type mode	`'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation'`
`OutDataType`	Output data type (e.g., sfix(16), uint(8), float(`'single'`))	string `{'sfix(16)'}`
`OutScaling`	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string `{'2^0'}`
`LockScale`	Lock output scaling against changes by the autoscaling tool	`'on' | {'off'}`
`RndMeth`	Round integer calculations toward	`'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}`
`SaturateOnIntegerOverflow`	Saturate on integer overflow	`'on' | {'off'}`
`ZeroCross`	Enable zero crossing detection	`{'on'} | 'off'`
`SampleTime`	Sample time (-1 for inherited)	string `{'-1'}`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
MinMax Running Resettable (`MinMax Running Resettable`) (masked subsystem)		
`Function`	Function	`{'min'} | 'max'`
`vinit`	Initial condition	string `{'0.0'}`
Polynomial (`Polyval`) (masked subsystem)		
`coefs`	Polynomial coefficients	string `{'[ +2.081618890e-019, -1.441693666e-014, +4.719686976e-010, -8.536869453e-006, +1.621573104e-001, -8.087801117e+001 ]'}`
Product (`Product`)		
`Inputs`	Number of inputs	string `{'2'}`
`Multiplication`	Multiplication	`{'Element-wise(.*)'} | 'Matrix(*)'`
`InputSameDT`	Require all inputs to have same data type	`'on' | {'off'}`
`OutDataTypeMode`	Output data type mode	`'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation' | 'Same as first input'`
`OutDataType`	Output data type (e.g., sfix(16), uint(8), float(`'single'`))	string `{'sfix(16)'}`
`OutScaling`	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string `{'2^0'}`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
LockScale	Lock output scaling against changes by the autoscaling tool	'on' | {'off'}
RndMeth	Round integer calculations toward	{'Zero'} | 'Nearest' | 'Ceiling' | 'Floor'
SaturateOnIntegerOverflow	Saturate on integer overflow	'on' | {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Product of Elements (Product)		
Inputs	Number of inputs	string {'*'}
Multiplication	Multiplication	{'Element-wise(.*)'} | 'Matrix(*)'
InputSameDT	Require all inputs to have same data type	'on' | {'off'}
OutDataTypeMode	Output data type mode	'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation' | 'Same as first input'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^-10'}
LockScale	Lock output scaling against changes by the autoscaling tool	'on' | {'off'}

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RndMeth	Round integer calculations toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on' | {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Real-Imag toComplex (RealImagToComplex)		
Input	Input	'Real' | 'Imag' | {'Real and imag'}
ConstantPart		string {'0'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Reshape (Reshape) (masked subsystem)		
OutputDimensionality	Output dimensionality	{'1-D array'} | 'Column vector' | 'Row vector' | 'Customize'
OutputDimensions	Output dimensions	string {'[1,1]'}
Rounding Function (Rounding)		
Operator	Function	{'floor'} | 'ceil' | 'round' | 'fix'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Sign (Signum)		
ZeroCross	Enable zero crossing detection	{'on'} | 'off'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Sine Wave Function (Sin)		
SineType	Sine type	{'Time based'} | 'Sample based'
TimeSource	Time (t)	'Use simulation time' | {'Use external signal'}

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`Amplitude`	Amplitude	string `{'1'}`
`Bias`	Bias	string `{'0'}`
`Frequency`	Frequency (rad/sec)	string `{'1'}`
`Phase`	Phase (rad)	string `{'0'}`
`Samples`	Samples per period	string `{'10'}`
`Offset`	Number of offset samples	string `{'0'}`
`SampleTime`	Sample time	string `{'0'}`
`VectorParams1D`	Interpret vector parameters as 1-D	`{'on'} | 'off'`
Slider Gain (`Slider Gain`) (masked subsystem)		
`low`	Low	string `{'0'}`
`gain`	Gain	string `{'1'}`
`high`	High	string `{'2'}`
Subtract (`Sum`)		
`IconShape`	Icon shape	`{'rectangular'} | 'round'`
`Inputs`	List of signs	string `{'+-'}`
`InputSameDT`	Require all inputs to have same data type	`'on' | {'off'}`
`OutDataTypeMode`	Output data type mode	`'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation' | 'Same as first input'`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^-10'}
LockScale	Lock output scaling against changes by the autoscaling tool	'on' | {'off'}
RndMeth	Round integer calculations toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on' | {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Sum (Sum)		
IconShape	Icon shape	'rectangular' | {'round'}
Inputs	List of signs	string {'|++'}
InputSameDT	Require all inputs to have same data type	'on' | {'off'}
OutDataTypeMode	Output data type mode	'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation' | 'Same as first input'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
LockScale	Lock output scaling against changes by the autoscaling tool	'on' | {'off'}
RndMeth	Round integer calculations toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on' | {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Sum of Elements (Sum)		
IconShape	Icon shape	{'rectangular'} | 'round'
Inputs	List of signs	string {'+'}
InputSameDT	Require all inputs to have same data type	'on' | {'off'}
OutDataTypeMode	Output data type mode	'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'Specify via dialog' | {'Inherit via internal rule'} | 'Inherit via back propagation' | 'Same as first input'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^-10'}

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
LockScale	Lock output scaling against changes by the autoscaling tool	'on' | {'off'}
RndMeth	Round integer calculations toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on' | {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Trigonometric Function (Trigonometry)		
Operator	Function	{'sin'} | 'cos' | 'tan' | 'asin' | 'acos' | 'atan' | 'atan2' | 'sinh' | 'cosh' | 'tanh' | 'asinh' | 'acosh' | 'atanh'
OutputSignalType	Output signal type	{'auto'} | 'real' | 'complex'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Unary Minus (Unary Minus) (masked subsystem)		
DoSatur	Saturate to max or min when overflows occur	'on' | {'off'}
Weighted Sample Time Math (Sample Time Math) (masked subsystem)		
TsampMathOp	Operation	{'+'} | '-' | '*' | '/' | 'Ts Only' | '1/Ts Only'
weightValue	Weight value	string {'1.0'}
TsampMathImp	Implement using	{'Online Calculations'} | 'Offline Scaling Adjustment'
OutputDataTypeScalingMode	Output data type and scaling	{'Inherit via internal rule'} | 'Inherit via back propagation'

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RndMeth	Round toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
DoSatur	Saturate to max or min when overflows occur	'on' | {'off'}

Model Verification Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Assertion (Assertion)		
Enabled	Enable assertion	{'on'} | 'off'
AssertionFailFcn	Simulation callback when assertion fails	string {''}
StopWhenAssertionFail	Stop simulation when assertion fails	{'on'} | 'off'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Check Discrete Gradient (Checks_Gradient) (masked subsystem)		
gradient	Maximum gradient	string {'1'}
enabled	Enable assertion	{'on'} | 'off'
callback	Simulation callback when assertion fails (optional)	string {''}
stopWhenAssertionFail	Stop simulation when assertion fails	{'on'} | 'off'
export	Output assertion signal	'on' | {'off'}
icon	Select icon type	{'graphic'} | 'text'
Check Dynamic Gap (Checks_DGap) (masked subsystem)		
enabled	Enable assertion	{'on'} | 'off'
callback	Simulation callback when assertion fails (optional)	string {''}

Model Verification Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
stopWhenAssertionFail	Stop simulation when assertion fails	{'on'} | 'off'
export	Output assertion signal	'on' | {'off'}
icon	Select icon type	{'graphic'} | 'text'
Check Dynamic Lower Bound (Checks_DMin) (masked subsystem)		
Enabled	Enable assertion	{'on'} | 'off'
callback	Simulation callback when assertion fails (optional)	string {''}
stopWhenAssertionFail	Stop simulation when assertion fails	{'on'} | 'off'
export	Output assertion signal	'on' | {'off'}
icon	Select icon type	{'graphic'} | 'text'
Check Dynamic Range (Checks_DRange) (masked subsystem)		
enabled	Enable assertion	{'on'} | 'off'
callback	Simulation callback when assertion fails (optional)	string {''}
stopWhenAssertionFail	Stop simulation when assertion fails	{'on'} | 'off'
export	Output assertion signal	'on' | {'off'}
icon	Select icon type	{'graphic'} | 'text'
Check Dynamic Upper Bound (Checks_DMax) (masked subsystem)		
enabled	Enable assertion	{'on'} | 'off'
callback	Simulation callback when assertion fails (optional)	string {''}
stopWhenAssertionFail	Stop simulation when assertion fails	{'on'} | 'off'
export	Output assertion signal	'on' | {'off'}

Model Verification Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
icon	Select icon type	{'graphic'} | 'text'
Check Input Resolution (Checks_Resolution) (masked subsystem)		
resolution	Resolution	string {'1'}
enabled	Enable assertion	{'on'} | 'off'
callback	Simulation callback when assertion fails (optional)	string {''}
stopWhenAssertionFail	Stop simulation when assertion fails	{'on'} | 'off'
export	Output assertion signal	'on' | {'off'}
Check Static Gap (Checks_SGap) (masked subsystem)		
max	Upper bound	string {'100'}
max_included	Inclusive upper bound	{'on'} | 'off'
min	Lower bound	string {'0'}
min_included	Inclusive lower bound	{'on'} | 'off'
enabled	Enable assertion	{'on'} | 'off'
callback	Simulation callback when assertion fails (optional)	string {''}
stopWhenAssertionFail	Stop simulation when assertion fails	{'on'} | 'off'
export	Output assertion signal	'on' | {'off'}
icon	Select icon type	{'graphic'} | 'text'
min	Lower bound	string {'0'}
Check Static Lower Bound (Checks_SMin) (masked subsystem)		
min_included	Inclusive boundary	{'on'} | 'off'
enabled	Enable assertion	{'on'} | 'off'

Model Verification Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
callback	Simulation callback when assertion fails (optional)	string {''}
stopWhenAssertionFail	Stop simulation when assertion fails	{'on'} | 'off'
export	Output assertion signal	'on' | {'off'}
icon	Select icon type	{'graphic'} | 'text'
Check Static Range (Checks_SRange) (masked subsystem)		
max	Upper bound	string {'100'}
max_included	Inclusive upper bound	{'on'} | 'off'
min	Lower bound	string {'0'}
min_included	Inclusive lower bound	{'on'} | 'off'
enabled	Enable assertion	{'on'} | 'off'
callback	Simulation callback when assertion fails (optional)	string {''}
stopWhenAssertionFail	Stop simulation when assertion fails	{'on'} | 'off'
export	Output assertion signal	'on' | {'off'}
icon	Select icon type	{'graphic'} | 'text'
Check Static Upper Bound (Checks_SMax) (masked subsystem)		
max	Upper bound	string {'0'}
max_included	Inclusive boundary	{'on'} | 'off'
enabled	Enable assertion	{'on'} | 'off'
callback	Simulation callback when assertion fails (optional)	string {''}
stopWhenAssertionFail	Stop simulation when assertion fails	{'on'} | 'off'

Model Verification Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`export`	Output assertion signal	`'on' | {'off'}`
`icon`	Select icon type	`{'graphic'} | 'text'`

Model-Wide Utilities Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Block Support Table (`Block Support Table`) (masked subsystem)		
DocBlock (`DocBlock`) (masked subsystem)		
`ECoderFlag`	RTW Embedded Coder Flag	string `{''}`
`DocumentType`	Document Type	`{'Text'} | 'RTF' | 'HTML'`
Model Advisor (`Model Advisor`) (masked subsystem)		
Model Info (`CMBlock`) (masked subsystem)		
`InitialSaveTempField`	InitialSaveTempField	string `{''}`
`InitialBlockCM`	InitialBlockCM	string `{'None'}`
`BlockCM`	BlockCM	string `{'None'}`
`Frame`	Frame	string `{'on'}`
`SaveTempField`	SaveTempField	string `{''}`
`DisplayStringWithTags`	DisplayStringWithTags	string `{'Model Info'}`
`MaskDisplayString`	MaskDisplayString	string `{'Model Info'}`
`HorizontalTextAlignment`	HorizontalTextAlignment	string `{'Center'}`
`LeftAlignmentValue`	LeftAlignmentValue	string `{'0.5'}`
`SourceBlockDiagram`	SourceBlockDiagram	string `{'untitled'}`
`TagMaxNumber`	TagMaxNumber	string `{'20'}`
`CMTag1`	CMTag1	string `{''}`
`CMTag2`	CMTag2	string `{''}`
`CMTag3`	CMTag3	string `{''}`

Model-Wide Utilities Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
CMTag4	CMTag4	string {''}
CMTag5	CMTag5	string {''}
CMTag6	CMTag6	string {''}
CMTag7	CMTag7	string {''}
CMTag8	CMTag8	string {''}
CMTag9	CMTag9	string {''}
CMTag10	CMTag10	string {''}
CMTag11	CMTag11	string {''}
CMTag12	CMTag12	string {''}
CMTag13	CMTag13	string {''}
CMTag14	CMTag14	string {''}
CMTag15	CMTag15	string {''}
CMTag16	CMTag16	string {''}
CMTag17	CMTag17	string {''}
CMTag18	CMTag18	string {''}
CMTag19	CMTag19	string {''}
CMTag20	CMTag20	string {''}
Timed-Based Linearization (Timed Linearization) (masked subsystem)		
LinearizationTime	Linearization time	string {'1'}
SampleTime	Sample time (of linearized model)	string {'0'}
Trigger-Based Linearization (Triggered Linearization) (masked subsystem)		

Model-Wide Utilities Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`TriggerType`	Trigger type	`{'rising'} | 'falling' | 'either' | 'function-call'`
`SampleTime`	Sample time (of linearized model)	string `{'0'}`

Ports & Subsystems Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Configurable Subsystem (`SubSystem`)		
`ShowPortLabels`	Show port labels	`{'on'} | 'off'`
`BlockChoice`	Block choice	`{''}`
`TemplateBlock`	Template block	string `{'self'}`
`MemberBlocks`	Member blocks	string `{''}`
`Permissions`	Read/write permissions	`{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'`
`ErrorFcn`	Name of error callback function	string `{''}`
`PermitHierarchical Resolution`	Permit hierarchical resolution	`{'All'} | 'ParametersOnly' | 'None'`
`TreatAsAtomicUnit`	Treat as atomic unit	`'on' | {'off'}`
`MinAlgLoopOccurrences`	Minimize algebraic loop occurrences	`'on' | {'off'}`
`PropExecContextAcross SSBoundary`	Propagate execution context across subsystem boundary	`{'on'} | 'off'`
`SystemSampleTime`	Sample time (-1 for inherited)	string `{'-1'}`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RTWSystemCode	RTW system code	{'Auto'} | 'Inline' | 'Function' | 'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'} | 'Use subsystem name' | 'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'
Atomic Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'} | 'off'
BlockChoice	Block choice	{''}
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'
ErrorFcn	Name of error callback function	string {''}

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
PermitHierarchical Resolution	Permit hierarchical resolution	{'All'} | 'ParametersOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'} | 'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on' | {'off'}
PropExecContextAcross SSBoundary	Propagate execution context across subsystem boundary	{'on'} | 'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	{'Auto'} | 'Inline' | 'Function' | 'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'} | 'Use subsystem name' | 'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'
Code Reuse Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'} | 'off'

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
BlockChoice	Block choice	{''}
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'
ErrorFcn	Name of error callback function	string {''}
PermitHierarchical Resolution	Permit hierarchical resolution	{'All'} | 'ParametersOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'} | 'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on' | {'off'}
PropExecContextAcross SSBoundary	Propagate execution context across subsystem boundary	{'on'} | 'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	'Auto' | 'Inline' | 'Function' | {'Reusable function'}
RTWFcnNameOpts	RTW function name options	'Auto' | {'Use subsystem name'} | 'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	'Auto' | 'Use subsystem name' | {'Use function name'} | 'User specified'
RTWFileName	RTW filename (no extension)	string {''}

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
DataTypeOverride		{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'
Enable (EnablePort)		
StatesWhenEnabling	States when enabling	{'held'} | 'reset'
ShowOutputPort	Show output port	'on' | {'off'}
ZeroCross	Enable zero crossing detection	{'on'} | 'off'
Enabled and Triggered Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'} | 'off'
BlockChoice	Block choice	{''}
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'
ErrorFcn	Name of error callback function	string {''}
PermitHierarchical Resolution	Permit hierarchical resolution	{'All'} | 'ParametersOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'} | 'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on' | {'off'}

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{'on'} | 'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	{'Auto'} | 'Inline' | 'Function' | 'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'} | 'Use subsystem name' | 'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'
EnabledSubsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'} | 'off'
BlockChoice	Block choice	{''}
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
ErrorFcn	Name of error callback function	string {''}
PermitHierarchical Resolution	Permit hierarchical resolution	{'All'} | 'ParametersOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'} | 'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on' | {'off'}
PropExecContextAcross SSBoundary	Propagate execution context across subsystem boundary	{'on'} | 'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	{'Auto'} | 'Inline' | 'Function' | 'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'} | 'Use subsystem name' | 'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
For Iterator Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'} | 'off'
BlockChoice	Block choice	{''}
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'
ErrorFcn	Name of error callback function	string {''}
PermitHierarchical Resolution	Permit hierarchical resolution	{'All'} | 'ParametersOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'} | 'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on' | {'off'}
PropExecContextAcross SSBoundary	Propagate execution context across subsystem boundary	{'on'} | 'off'
SystemSampleTime	Sample time (-1 for inherited)	string{'-1'}
RTWSystemCode	RTW system code	{'Auto'} | 'Inline' | 'Function' | 'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'} | 'Use subsystem name' | 'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'
RTWFileName	RTW filename (no extension)	string {''}

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
DataTypeOverride		{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'
Function-Call Generator (Function-Call Generator) (masked subsystem)		
sample_time	Sample time	string {'1'}
numberOfIterations	Number of iterations	string {'1'}
Function-Call Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'} | 'off'
BlockChoice	Block choice	{''}
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'
ErrorFcn	Name of error callback function	string {''}
PermitHierarchical Resolution	Permit hierarchical resolution	{'All'} | 'ParametersOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'} | 'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on' | {'off'}

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{'on'} | 'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	{'Auto'} | 'Inline' | 'Function' | 'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'} | 'Use subsystem name' | 'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'
If (If)		
NumInputs	Number of inputs	string {'1'}
IfExpression	If expression (e.g., u1 ~= 0)	string {'u1 > 0'}
ElseIfExpressions	Elseif expressions (comma-separated list, e.g., u2 ~= 0, u3(2) < u2)	string {''}
ShowElse	Show else condition	{'on'} | 'off'
ZeroCross	Enable zero crossing detection	{'on'} | 'off'

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time (-1 for inherited)	string {'-1'}
If Action Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'} | 'off'
BlockChoice	Block choice	{''}
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'
ErrorFcn	Name of error callback function	string {''}
PermitHierarchical Resolution	Permit hierarchical resolution	{'All'} | 'ParametersOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'} | 'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on' | {'off'}
PropExecContextAcross SSBoundary	Propagate execution context across subsystem boundary	{'on'} | 'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	{'Auto'} | 'Inline' | 'Function' | 'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'} | 'Use subsystem name' | 'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'
In1 (Inport)		
Port	Port number	string {'1'}
IconDisplay	Icon display	'Signal name' | {'Port number'} | 'Port number and signal name'
UseBusObject	Specify properties via bus object	'on' | {'off'}
BusObject	For internal use	
BusOutputAsStruct	Output as nonvirtual bus	'on' | {'off'}
PortDimensions	Port dimensions (-1 for inherited)	string {'-1'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
DataType	Data type	{'auto'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Specify via dialog'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutScaling`	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string `{'2^0'}`
`SignalType`	Signal type	`{'auto'} | 'real' | 'complex'`
`SamplingMode`	Sampling mode	`{'auto'} | 'Sample based' | 'Frame based'`
`LatchInput`	Latch (buffer) input	`'on' | {'off'}`
`Interpolate`	Interpolate data	`{'on'} | 'off'`
Model (`ModelReference`)		
`ModelName`	Model name (without the .mdl extension)	string `{'<Enter Model Name>'}`
`ParameterArgumentNames`	Model arguments	string `{''}`
`ParameterArgumentValues`	Model argument values (for this instance)	string `{''}`
`AvailSigsInstanceProps`		`handle vector {''}`
`AvailSigsDefaultProps`		`handle vector {''}`
`DefaultDataLogging`		`'on' | {'off'}`
Out1 (`Outport`)		
`Port`	Port number	string `{'1'}`
`IconDisplay`	Icon display	`'Signal name' | {'Port number'} | 'Port number and signal name'`
`UseBusObject`	Specify properties via bus object	`'on' | {'off'}`
`BusObject`	For internal use	
`BusOutputAsStruct`	Output as nonvirtual bus in parent model	`'on' | {'off'}`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
PortDimensions	Port dimensions (-1 for inherited)	string {'-1'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
DataType	Data type	{'auto'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Specify via dialog'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
SignalType	Signal type	{'auto'} | 'real' | 'complex'
SamplingMode	Sampling mode	{'auto'} | 'Sample based' | 'Frame based'
OutputWhenDisabled	Output when disabled	{'held'} | 'reset'
InitialOutput	Initial output	string {'[]'}
Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'} | 'off'
BlockChoice	Block choice	{''}
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
ErrorFcn	Name of error callback function	string {''}
PermitHierarchical Resolution	Permit hierarchical resolution	{'All'} | 'ParametersOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	'on' | {'off'}
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on' | {'off'}
PropExecContextAcross SSBoundary	Propagate execution context across subsystem boundary	{'on'} | 'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	{'Auto'} | 'Inline' | 'Function' | 'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'} | 'Use subsystem name' | 'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Virtual	For internal use	
Switch Case (SwitchCase)		
CaseConditions	Case conditions (e.g., {1,[2,3]})	string {'{1}'}
CaseShowDefault	Show default case	{'on'} | 'off'
ZeroCross	Enable zero-crossing detection	{'on'} | 'off'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Switch Case Action Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'} | 'off'
BlockChoice	Block choice	{''}
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'
ErrorFcn	Name of error callback function	string {''}
PermitHierarchical Resolution	Permit hierarchical resolution	{'All'} | 'ParametersOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'} | 'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on' | {'off'}
PropExecContextAcross SSBoundary	Propagate execution context across subsystem boundary	{'on'} | 'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	{'Auto'} | 'Inline' | 'Function' | 'Reusable function'

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RTWFcnNameOpts	RTW function name options	{'Auto'} | 'Use subsystem name' | 'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'
Trigger (TriggerPort)		
TriggerType	Trigger type	{'rising'} | 'falling' | 'either' | 'function-call'
StatesWhenEnabling	States when enabling	{'held'} | 'reset' | 'inherit'
ShowOutputPort	Show output port	'on' | {'off'}
OutputDataType	Output data type	{'auto'} | 'double' | 'int8'
SampleTimeType	Sample time type	{'triggered'} | 'periodic'
SampleTime	Sample time	string {'1'}
ZeroCross	Enable zero crossing detection	{'on'} | 'off'
Triggered Subsystem (SubSystem)		

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
ShowPortLabels	Show port labels	{'on'} | 'off'
BlockChoice	Block choice	{''}
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'
ErrorFcn	Name of error callback function	string {''}
PermitHierarchical Resolution	Permit hierarchical resolution	{'All'} | 'ParametersOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'} | 'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on' | {'off'}
PropExecContextAcross SSBoundary	Propagate execution context across subsystem boundary	{'on'} | 'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	{'Auto'} | 'Inline' | 'Function' | 'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'} | 'Use subsystem name' | 'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'
RTWFileName	RTW filename (no extension)	string {''}

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
DataTypeOverride		{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'
While Iterator Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'} | 'off'
BlockChoice	Block choice	{''}
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'
ErrorFcn	Name of error callback function	string {''}
PermitHierarchical Resolution	Permit hierarchical resolution	{'All'} | 'ParametersOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'} | 'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on' | {'off'}
PropExecContextAcross SSBoundary	Propagate execution context across subsystem boundary	{'on'} | 'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RTWSystemCode	RTW system code	{'Auto'} | 'Inline' | 'Function' | 'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'} | 'Use subsystem name' | 'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'

Signal Attributes Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Data Type Conversion (DataTypeConversion)		
OutDataTypeMode	Output data type mode	'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Specify via dialog' | {'Inherit via back propagation'}

Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutDataType`	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
`OutScaling`	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
`LockScale`	Lock output scaling against changes by the autoscaling tool	'on' | {'off'}
`ConvertRealWorld`	Input and output to have equal	{'Real World Value (RWV)'} | 'Stored Integer (SI)'
`RndMeth`	Round integer calculations toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
`SaturateOnIntegerOverflow`	Saturate on integer overflow	'on' | {'off'}
`SampleTime`	Sample time (-1 for inherited)	string {'-1'}
Data Type Conversion Inherited (`Conversion Inherited`) (masked subsystem)		
`ConvertRealWorld`	Input and Output to have equal	{'Real World Value'} | 'Stored Integer'
`RndMeth`	Round toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
`DoSatur`	Saturate to max or min when overflows occur	'on' | {'off'}
Data Type Duplicate (`Data Type Duplicate`) (masked subsystem)		
`NumInputPorts`	Number of input ports	string {'2'}
Data Type Propagation (`Data Type Propagation`) (masked subsystem)		
`PropDataTypeMode`	1. Propagated data type	'Specify via dialog' | {'Inherit via propagation rule'}

Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
PropDataType	1.1. Propagated data type: ex. sfix(16), uint(8), float('single')	string {'sfix(16)'}
IfRefDouble	1.1. If any reference input is double, output is	{'double'} | 'single'
IfRefSingle	1.2. If any reference input is single, output is	'double' | {'single'}
IsSigned	1.3. Is-Signed	'IsSigned1' | 'IsSigned2' | {'IsSigned1 or IsSigned2'} | 'TRUE' | 'FALSE'
NumBitsBase	1.4.1. Number-of-Bits: Base	'NumBits1' | 'NumBits2' | {'max([NumBits1 NumBits2])'} | 'min([NumBits1 NumBits2])' | 'NumBits1+NumBits2'
NumBitsMult	1.4.2. Number-of-Bits: Multiplicative adjustment	string {'1'}
NumBitsAdd	1.4.3. Number-of-Bits: Additive adjustment	string {'0'}
NumBitsAllowFinal	1.4.4. Number-of-Bits: Allowable final values	string {'1:128'}
PropScalingMode	2. Propagated scaling	'Specify via dialog' | {'Inherit via propagation rule'} | 'Obtain via best precision'
PropScaling	2.1. Propagated scaling: Slope or [Slope Bias] ex. 2^-9	string {'2^-10'}
ValuesUsedBestPrec	2.1. Values used to determine best precision scaling	string {'[5 -7]'}

Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SlopeBase	2.1.1. Slope: Base	'Slope1' | 'Slope2' | 'max([Slope1 Slope2])' | {'min([Slope1 Slope2])'} | 'Slope1*Slope2' | 'Slope1/Slope2' | 'PosRange1' | 'PosRange2' | 'max([PosRange1 PosRange2])' | 'min([PosRange1 PosRange2])' | 'PosRange1*PosRange2' | 'PosRange1/PosRange2'
SlopeMult	2.1.2. Slope: Multiplicative adjustment	string {'1'}
SlopeAdd	2.1.3. Slope: Additive adjustment	string {'0'}
BiasBase	2.2.1. Bias: Base	{'Bias1'} | 'Bias2' | 'max([Bias1 Bias2])' | 'min([Bias1 Bias2])' | 'Bias1*Bias2' | 'Bias1/Bias2' | 'Bias1+Bias2' | 'Bias1-Bias2'
BiasMult	2.2.2. Bias: Multiplicative adjustment	string {'1'}
BiasAdd	2.2.3. Bias: Additive adjustment	string {'0'}
Data Type Scaling Strip (Scaling Strip) (masked subsystem)		
IC (InitialCondition)		
Value	Initial value	string {'1'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}

Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Probe (`Probe`)		
`ProbeWidth`	Probe width	`{'on'} | 'off'`
`ProbeSampleTime`	Probe sample time	`{'on'} | 'off'`
`ProbeComplexSignal`	Detect complex signal	`{'on'} | 'off'`
`ProbeSignalDimensions`	Probe signal dimensions	`{'on'} | 'off'`
`ProbeFramedSignal`	Detect framed signal	`{'on'} | 'off'`
`ProbeWidthDataType`	Data type for width	`{'double'} | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Same as input'`
`ProbeSampleTimeDataType`	Data type for sample time	`{'double'} | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Same as input'`
`ProbeComplexityDataType`	Data type for signal complexity	`{'double'} | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Same as input'`
`ProbeDimensionsDataType`	Data type for signal dimensions	`{'double'} | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Same as input'`

Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
ProbeFrameDataType	Data type for signal frames	{'double'} | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Same as input'
Rate Transition (RateTransition)		
Integrity	Ensure data integrity during data transfer	{'on'} | 'off'
Deterministic	Ensure deterministic data transfer (maximum delay)	{'on'} | 'off'
X0	Initial conditions	string {'0'}
OutPortSampleTime	Output port sample time	string {'-1'}
Signal Conversion (SignalConversion)		
ConversionOutput	Output	{'Contiguous copy'} | 'Bus copy' | 'Virtual bus' | 'Nonvirtual bus'
OverrideOpt	Override optimizations and always copy signal	'on' | {'off'}
Signal Specification (SignalSpecification)		
Dimensions	Dimensions (-1 for inherited)	string {'-1'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
DataType	Data type	{'auto'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Specify via dialog'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}

Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
SignalType	Signal type	{'auto'} | 'real' | 'complex'
SamplingMode	Sampling mode	{'auto'} | 'Sample based' | 'Frame based'
Weighted Sample Time (Sample Time Math) (masked subsystem)		
TsampMathOp	Operation	'+' | '-' | '*' | '/' | {'Ts Only'} | '1/Ts Only'
weightValue	Weight value	string {'1.0'}
TsampMathImp	Implement using	{'Online Calculations'} | 'Offline Scaling Adjustment'
OutputDataTypeScaling Mode	Output data type and scaling	{'Inherit via internal rule'} | 'Inherit via back propagation'
RndMeth	Round toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
DoSatur	Saturate to max or min when overflows occur	'on' | {'off'}
Width (Width)		
OutputDataTypeScaling Mode	Output data type mode	{'Choose intrinsic data type'} | 'Inherit via back propagation' | 'All ports same datatype'
DataType	Output data type	{'double'} | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32'

Signal Routing Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Bus Creator (`BusCreator`)		
`Inputs`	Number of inputs. Can be an integer or a comma-separated list of signal names. For example, `set_param(gcb, '''a'',''b'');` sets the currently selected Bus Creator block two have two inputs named `a` and `b`.	string `{'2'}`
`DisplayOption`	Display option	`'none' | 'signals' | {'bar'}`
`UseBusObject`	Specify properties via bus object	`'on' | {'off'}`
`BusObject`	For internal use	
`NonVirtualBus`	Output as nonvirtual bus	`'on' | {'off'}`
Bus Selector (`BusSelector`)		
`OutputSignals`	Specifies the names of the input bus signals selected for output. Corresponds to the **Selected signals** list on the block's parameter dialog box.	string `{'signal1,signal2'}`

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
MuxedOutput	If on, this parameter causes the Bus Selector to multiplex the selected signals at its output. Corresponds to the Muxed output check box on the block's parameter dialog box.	'on' | {'off'}
InputSignals	Specifies the names of the signal elements of the bus connected to the Bus Selector's input port.	matrix {'{}'}
Data Store Memory (DataStoreMemory)		
DataStoreName	Data store name	string {'A'}
InitialValue	Initial value	string {'0'}
StateMustResolveTo SignalObject	Data store name must resolve to Simulink signal object	'on' | {'off'}
RTWStateStorageClass	RTW storage class	{'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'
RTWStateStorageType Qualifier	RTW type qualifier	string {''}
VectorParams1D	Interpret vector parameters as 1-D	{'on'} | 'off'
ShowAdditionalParam	Show additional parameters	'on' | {'off'}

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
DataType	Data type	{'auto'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Specify via dialog'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
SignalType	Signal type	{'auto'} | 'real' | 'complex'
Data Store Read (DataStoreRead)		
DataStoreName	Data store name	string {'A'}
SampleTime	Sample time	string {'0'}
Data Store Write (DataStoreWrite)		
DataStoreName	Data store name	string {'A'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Demux (Demux)		
Outputs	Number of outputs	string {'2'}
DisplayOption	Display option	'none' | {'bar'}
BusSelectionMode	Bus selection mode	'on' | {'off'}
Environment Controller (Environment Controller) (masked subsystem)		

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
From (`From`)		
`GotoTag`	Goto tag	string `{'A'}`
`IconDisplay`	Icon display	`'Signal name' | {'Tag'} | 'Tag and signal name'`
Goto (`Goto`)		
`GotoTag`	Tag	string `{'A'}`
`IconDisplay`	Icon display	`'Signal name' | {'Tag'} | 'Tag and signal name'`
`TagVisibility`	Tag visibility	`{'local'} | 'scoped' | 'global'`
Goto Tag Visibility (`GotoTagVisibility`)		
`GotoTag`	Goto tag	`string {'A'}`
Index Vector (`MultiPortSwitch`)		
`Inputs`	Number of inputs	string `{'1'}`
`zeroidx`	Use zero-based indexing	`{'on'} | 'off'`
`InputSameDT`	Require all data port inputs to have same data type	`'on' | {'off'}`
`OutDataTypeMode`	Output data type mode	`{'Inherit via internal rule'} | 'Inherit via back propagation'`
`RndMeth`	Round integer calculations toward	`'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}`

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SaturateOnIntegerOverflow	Saturate on integer overflow	'on' | {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Manual Switch (Manual Switch) (masked subsystem)		
sw	Current setting	string {'1'}
action	Action	string {'0'}
Merge (Merge)		
Inputs	Number of inputs	string {'2'}
InitialOutput	Initial output	string {'[]'}
AllowUnequalInputPortWidths	Allow unequal port widths	'on' | {'off'}
InputPortOffsets	Input port offsets	string {'[]'}
Multiport Switch (MultiPortSwitch)		
Inputs	Number of inputs	string {'3'}
zeroidx	Use zero-based indexing	'on' | {'off'}
InputSameDT	Require all data port inputs to have same data type	'on' | {'off'}
OutDataTypeMode	Output data type mode	{'Inherit via internal rule'} | 'Inherit via back propagation'
RndMeth	Round integer calculations toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SaturateOnIntegerOverflow	Saturate on integer overflow	'on' | {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Mux (Mux)		
Inputs	Number of inputs	string {'2'}
DisplayOption	Display option	'none' | 'signals' | {'bar'}
UseBusObject	Specify properties via bus object	'on' | {'off'}
BusObject	For internal use	
NonVirtualBus	Output as nonvirtual bus	'on' | {'off'}
Selector (Selector)		
InputType	Input type	{'Vector'} | 'Matrix'
IndexMode	Index mode	'Zero-based' | {'One-based'}
ElementSrc	Source of element indices (E)	{'Internal'} | 'External'
Elements	Elements (-1 for all elements)	string {'[1 3]'}
RowSrc	Source of row indices (R)	{'Internal'} | 'External'
Rows	Rows (-1 for all rows)	string {'1'}
ColumnSrc	Source of column indices (C)	{'Internal'} | 'External'

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Columns	Columns (-1 for all columns)	string {'1'}
InputPortWidth	Input port width	string {'3'}
IndexIsStartValue	Use index as starting value	'on' | {'off'}
OutputPortSize	Output port dimensions	string {'1'}
Switch (Switch)		
Criteria	Criteria for passing first input	{'u2 >= Threshold'} | 'u2 > Threshold' | 'u2 ~= 0'
Threshold	Threshold	string {'0'}
InputSameDT	Require all data port inputs to have same data type	'on' | {'off'}
OutDataTypeMode	Output data type mode	{'Inherit via internal rule'} | 'Inherit via back propagation'
RndMeth	Round integer calculations toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on' | {'off'}

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
ZeroCross	Enable zero crossing detection	{'on'} | 'off'
SampleTime	Sample time (-1 for inherited)	string {'-1'}

Sinks Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Display (Display)		
Format	Format	{'short'} | 'long' | 'short_e' | 'long_e' | 'bank' | 'hex (Stored Integer)' | 'binary (Stored Integer)' | 'decimal (Stored Integer)' | 'octal (Stored Integer)'
Decimation	Decimation	string {'1'}
Floating	Floating display	'on' | {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Floating Scope (Scope)		
Floating		{'on'} | 'off'
Location		rectangle {'[376 294 700 533]'}
Open		'on' | {'off'}
NumInputPorts		string {'1'}
TickLabels		'on' | 'off' | {'OneTimeTick'}

Sinks Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
ZoomMode		{'on'} | 'xonly' | 'yonly'
AxesTitles		list
Grid		'off' | {'on'} | 'xonly' | 'yonly'
TimeRange		string {'auto'}
YMin		string {'-5'}
YMax		string {'5'}
SaveToWorkspace		'on' | {'off'}
SaveName		string {'ScopeData'}
DataFormat		{'StructureWithTime'} | 'Structure' | 'Array'
LimitDataPoints		{'on'} | 'off'
MaxDataPoints		string {'5000'}
Decimation		string {'1'}
SampleInput		'on' | {'off'}
SampleTime		string {'0'}
Out1 (Outport)		
Port	Port number	string {'1'}
IconDisplay	Icon display	'Signal name' | {'Port number'} | 'Port number and signal name'
UseBusObject	Specify properties via bus object	'on' | {'off'}
BusObject	For internal use	
BusOutputAsStruct	Output as nonvirtual bus in parent model	'on' | {'off'}

Sinks Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
PortDimensions	Port dimensions (-1 for inherited)	string {'-1'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
DataType	Data type	{'auto'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Specify via dialog'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
SignalType	Signal type	{'auto'} | 'real' | 'complex'
SamplingMode	Sampling mode	{'auto'} | 'Sample based' | 'Frame based'
OutputWhenDisabled	Output when disabled	{'held'} | 'reset'
InitialOutput	Initial output	string {'[]'}
Scope (Scope)		
Floating		'on' | {'off'}
Location		rectangle {'[188 390 512 629]'}
Open		'on' | {'off'}

Sinks Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
NumInputPorts		string {'1'}
TickLabels		'on' | 'off' | {'OneTimeTick'}
ZoomMode		{'on'} | 'xonly' | 'yonly'
AxesTitles		list
Grid		'off' | {'on'} | 'xonly' | 'yonly'
TimeRange		string {'auto'}
YMin		string {'-5'}
YMax		string {'5'}
SaveToWorkspace		'on' | {'off'}
SaveName		string {'ScopeData1'}
DataFormat		{'StructureWithTime'} | 'Structure' | 'Array'
LimitDataPoints		{'on'} | 'off'
MaxDataPoints		string {'5000'}
Decimation		string {'1'}
SampleInput		'on' | {'off'}
SampleTime		string {'0'}
Stop Simulation		
Terminator		
To File (ToFile)		
Filename	Filename	string {'untitled.mat'}
MatrixName	Variable name	string {'ans'}
Decimation	Decimation	string {'1'}

Sinks Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time (-1 for inherited)	string {'-1'}
To Workspace (ToWorkspace)		
VariableName	Variable name	string {'simout'}
MaxDataPoints	Limit data points to last	string {'inf'}
Decimation	Decimation	string {'1'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
SaveFormat	Save format	'Structure With Time' | {'Structure'} | 'Array'
FixptAsFi	Log fixed-point data as a fi object	'on' | {'off'}
XY Graph (XY scope) (masked subsystem)		
xmin	x-min	string {'-1'}
xmax	x-max	string {'1'}
ymin	y-min	string {'-1'}
ymax	y-max	string {'1'}
st	Sample time	string {'-1'}

Sources Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Band-Limited White Noise (Band-Limited White Noise) (masked subsystem)		
Cov	Noise power	string {'[0.1]'}
Ts	Sample time	string {'0.1'}
seed	Seed	string {'[23341]'}
VectorParams1D	Interpret vector parameters as 1-D	{'on'} | 'off'
Chirp Signal (chirp) (masked subsystem)		

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
f1	Initial frequency (Hz)	string {'0.1'}
T	Target time (secs)	string {'100'}
f2	Frequency at target time (Hz)	string {'1'}
VectorParams1D	Interpret vectors parameters as 1-D	{'on'} | 'off'
Clock (Clock)		
DisplayTime	Display time	'on' | {'off'}
Decimation	Decimation	string {'10'}
Constant (Constant)		
Value	Constant value	string {'1'}
VectorParams1D	Interpret vector parameters as 1-D	{'on'} | 'off'
OutDataTypeMode	Output data type mode	'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Specify via dialog' | {'Inherit from 'Constant value''} | 'Inherit via back propagation'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
ConRadixGroup	Output scaling mode	{'Use specified scaling'} | 'Best Precision: Vector-wise'
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^0'}

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time	string {'inf'}
Counter Free-Running (Counter Free-Running) (masked subsystem)		
NumBits	Number of Bits	string {'16'}
tsamp	Sample time	string {'-1'}
Counter Limited (Counter Limited) (masked subsystem)		
uplimit	Upper limit	string {'7'}
tsamp	Sample time	string {'-1'}
Digital Clock (DigitalClock)		
SampleTime	Sample time	string {'1'}
From File (FromFile)		
FileName	Filename	string {'untitled.mat'}
SampleTime	Sample time	string {'0'}
From Workspace (FromWorkspace)		
VariableName	Data	string {'simin'}
SampleTime	Sample time	string {'0'}
Interpolate	Interpolate data	{'on'} | 'off'
OutputAfterFinalValue	Form output after final data value by	{'Extrapolation'} | 'Setting to zero' | 'Holding final value' | 'Cyclic repetition'
Ground		
In1 (Inport)		
Port	Port number	string {'1'}
IconDisplay	Icon display	'Signal name' | {'Port number'} | 'Port number and signal name'

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`UseBusObject`	Specify properties via bus object	`'on' | {'off'}`
`BusObject`	For internal use	
`BusOutputAsStruct`	Output as nonvirtual bus	`'on' | {'off'}`
`PortDimensions`	Port dimensions (-1 for inherited)	string `{'-1'}`
`SampleTime`	Sample time (-1 for inherited)	string `{'-1'}`
`DataType`	Data type	`{'auto'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Specify via dialog'`
`OutDataType`	Output data type (e.g., sfix(16), uint(8), float(`'single'`))	string `{'sfix(16)'}`
`OutScaling`	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string `{'2^0'}`
`SignalType`	Signal type	`{'auto'} | 'real' | 'complex'`
`SamplingMode`	Sampling mode	`{'auto'} | 'Sample based' | 'Frame based'`
`LatchInput`	Latch (buffer) input	`'on' | {'off'}`
`Interpolate`	Interpolate data	`{'on'} | 'off'`
Pulse Generator (`DiscretePulseGenerator`)		
`PulseType`	Pulse type	`{'Time based'} | 'Sample based'`
`TimeSource`	Time (t)	`{'Use simulation time'} | 'Use external signal'`

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Amplitude	Amplitude	string {'1'}
Period	Period	string {'2'}
PulseWidth	Pulse width	string {'50'}
PhaseDelay	Phase delay	string {'0'}
SampleTime	Sample time	string {'1'}
VectorParams1D	Interpret vector parameters as 1-D	{'on'} | 'off'
Ramp (Ramp) (masked subsystem)		
slope	Slope	string {'1'}
start	Start time	string {'0'}
X0	Initial output	string {'0'}
VectorParams1D	Interpret vector parameters as 1-D	{'on'} | 'off'
Random Number (RandomNumber)		
Mean	Mean	string {'0'}
Variance	Variance	string {'1'}
Seed	Initial seed	string {'0'}
SampleTime	Sample time	string {'0'}
VectorParams1D	Interpret vector parameters as 1-D	{'on'} | 'off'
Repeating Sequence (Repeating table) (masked subsystem)		
rep_seq_t	Time values	string {'[0 2]'}
rep_seq_y	Output values	string {'[0 2]'}
Repeating Sequence Interpolated (Repeating Sequence Interpolated) (masked subsystem)		
OutValues	Vector of output values	string {'[3 1 4 2 1].''}

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
TimeValues	Vector of time values	string {'[0 0.1 0.5 0.6 1].''}
LookUpMeth	Lookup method	{'Interpolation-Use End Values'} | 'Use Input Nearest' | 'Use Input Below' | 'Use Input Above'
tsamp	Sample time	string {'0.01'}
OutputDataTypeScaling Mode	Output data type and scaling	{'Specify via dialog'} | 'Inherit via back propagation'
OutDataType	Output data type: ex. sfix(16), uint(8), float('single')	string {'float('double')'}
OutScaling	Output scaling: Slope or [Slope Bias] ex. 2^-9	string {'2^-10'}
LockScale	Lock output scaling so autoscaling tool can't change it	'on' | {'off'}
Repeating Sequence Stair (Repeating Sequence Stair) (masked subsystem)		
OutValues	Vector of output values	string {'[3 1 4 2 1].''}
tsamp	Sample time	string {'-1'}
OutputDataTypeScaling Mode	Output data type and scaling	{'Specify via dialog'} | 'Inherit via back propagation'
OutDataType	Output data type: ex. sfix(16), uint(8), float('single')	string {'float('double')'}
ConRadixGroup	Output scaling mode	'Use Specified Scaling' | {'Best Precision: Vector-wise'}
OutScaling	Output scaling: Slope or [Slope Bias] ex. 2^-9	string {'2^-12'}

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
LockScale	Lock output scaling so autoscaling tool can't change it	'on' | {'off'}
Signal Builder (Sigbuilder block) (masked subsystem)		
Signal Generator (SignalGenerator)		
WaveForm	Wave form	{'sine'} | 'square' | 'sawtooth' | 'random'
TimeSource	Time (t)	{'Use simulation time'} | 'Use external signal'
Amplitude	Amplitude	string {'1'}
Frequency	Frequency	string {'1'}
Units	Units	'rad/sec' | {'Hertz'}
VectorParams1D	Interpret vector parameters as 1-D	{'on'} | 'off'
Sine Wave (Sin)		
SineType	Sine type	{'Time based'} | 'Sample based'
TimeSource	Time (t)	{'Use simulation time'} | 'Use external signal'
Amplitude	Amplitude	string {'1'}
Bias	Bias	string {'0'}
Frequency	Frequency (rad/sec)	string {'1'}
Phase	Phase (rad)	string {'0'}
Samples	Samples per period	string {'10'}
Offset	Number of offset samples	string {'0'}
SampleTime	Sample time	string {'0'}
VectorParams1D	Interpret vector parameters as 1-D	{'on'} | 'off'

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Step (`Step`)		
`Time`	Step time	string `{'1'}`
`Before`	Initial value	string `{'0'}`
`After`	Final value	string `{'1'}`
`SampleTime`	Sample time	string `{'0'}`
`VectorParams1D`	Interpret vector parameters as 1-D	`{'on'} | 'off'`
`ZeroCross`	Enable zero crossing detection	`{'on'} | 'off'`
Uniform Random Number (`UniformRandomNumber`)		
`Minimum`	Minimum	string `{'-1'}`
`Maximum`	Maximum	string `{'1'}`
`Seed`	Initial seed	string `{'0'}`
`SampleTime`	Sample time	string `{'0'}`
`VectorParams1D`	Interpret vector parameters as 1-D	`{'on'} | 'off'`

User-Defined Functions Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Embedded MATLAB Fcn (`Stateflow`) (masked subsystem)		
Fcn (`Fcn`)		
`Expr`	Expression	string `{'sin(u(1)*exp(2.3*(-u(2))))'}`
`SampleTime`	Sample time (-1 for inherited)	string `{'-1'}`
M-file S-Function (`M-S-Function`)		
`MFile`	M-file name	string `{'mlfile'}`

User-Defined Functions Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Parameters	Parameters	string {''}
MATLAB Fcn (MATLABFcn)		
MATLABFcn	MATLAB function	string {'sin'}
OutputDimensions	Output dimensions	string {'-1'}
OutputSignalType	Output signal type	{'auto'} | 'real' | 'complex'
Output1D	Collapse 2-D results to 1-D	{'on'} | 'off'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
S-Function (S-Function)		
FunctionName	S-function name	string {'system'}
Parameters	S-function parameters	string {''}
S-Function Builder (S-Function Builder) (masked subsystem)		
FunctionName	S-function name	string {'system'}
Parameters	S-function parameters	string {''}

Additional Discrete Block Library Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Fixed-Point State-Space (Fixed-Point State-Space) (masked subsystem)		
A	State Matrix A	string {'[2.6020 -2.2793 0.6708; 1 0 0; 0 1 0]'}
B	Input Matrix B	string {'[1; 0; 0]'}
C	Output Matrix C	string {'[0.0184 0.0024 0.0055]'}
D	Direct Feedthrough Matrix D	string {'[0.0033]'}

Additional Discrete Block Library Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`X0`	Initial condition for state	string `{'0.0'}`
`InternalDataType`	Data type for internal calculations: ex. sfix(16), uint(8), float(`'single'`)	string `{'float('double')'}`
`StateEqScaling`	Scaling for State Equation AX+BU: ex. 2^-9	string `{'2^0'}`
`OutputEqScaling`	Scaling for Output Equation CX+DU: ex. 2^-9	string `{'2^0'}`
`LockScale`	Lock output scaling so autoscaling tool can't change it	`'on' | {'off'}`
`RndMeth`	Round toward	`'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}`
`DoSatur`	Saturate to max or min when overflows occur	`'on' | {'off'}`
Transfer Fcn Direct Form II (`Transfer Fcn Direct Form II`) (masked subsystem)		
`NumCoefVec`	Numerator coefficients	string `{'[0.2 0.3 0.2]'}`
`DenCoefVec`	Denominator coefficients excluding lead (which must be 1.0)	string `{'[-0.9 0.6]'}`
`vinit`	Initial condition	string `{'0.0'}`
`RndMeth`	Round toward	`'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}`
`DoSatur`	Saturate to max or min when overflows occur	`'on' | {'off'}`
Transfer Fcn Direct Form II Time Varying (`Transfer Fcn Direct Form II Time Varying`) (masked subsystem)		
`vinit`	Initial condition	string `{'0.0'}`

Additional Discrete Block Library Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RndMeth	Round toward	'Zero' | 'Nearest' | 'Ceiling' | {'Floor'}
DoSatur	Saturate to max or min when overflows occur	'on' | {'off'}
Unit Delay Enabled (Unit Delay Enabled) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
tsamp	Sample time	string {'-1'}
Unit Delay Enabled External IC (Unit Delay Enabled External Initial Condition) (masked subsystem)		
tsamp	Sample time	string {'-1'}
Unit Delay Enabled Resettable (Unit Delay Enabled Resettable) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
tsamp	Sample time	string {'-1'}
Unit Delay Enabled Resettable External IC (Unit Delay Enabled Resettable External Initial Condition) (masked subsystem)		
tsamp	Sample time	string {'-1'}
Unit Delay External IC (Unit Delay External Initial Condition) (masked subsystem)		
tsamp	Sample time	string {'-1'}
Unit Delay Resettable (Unit Delay Resettable) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
tsamp	Sample time	string {'-1'}
Unit Delay Resettable External IC (Unit Delay Resettable External Initial Condition) (masked subsystem)		
tsamp	Sample time	string {'-1'}

Additional Discrete Block Library Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Unit Delay With Preview Enabled (`Unit Delay With Preview Enabled`) (masked subsystem)		
`vinit`	Initial condition	string `{'0.0'}`
`tsamp`	Sample time	string `{'-1'}`
Unit Delay With Preview Enabled Resettable (`Unit Delay With Preview Enabled Resettable`) (masked subsystem)		
`vinit`	Initial condition	string `{'0.0'}`
`tsamp`	Sample time	string `{'-1'}`
Unit Delay With Preview Enabled Resettable External RV (`Unit Delay With Preview Enabled Resettable External RV`) (masked subsystem)		
`vinit`	Initial condition	string `{'0.0'}`
`tsamp`	Sample time	string `{'-1'}`
Unit Delay With Preview Resettable (`Unit Delay With Preview Resettable`) (masked subsystem)		
`vinit`	Initial condition	`string` {'0.0'}
`tsamp`	Sample time	`string` {'-1'}
Unit Delay With Preview Resettable External RV (`Unit Delay With Preview Resettable External RV`) (masked subsystem)		

Additional Discrete Block Library Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
vinit	Initial condition	string {'0.0'}
tsamp	Sample time	string {'-1'}

Additional Math: Increment - Decrement Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Decrement Real World (Real World Value Decrement) (masked subsystem)		
Decrement Stored Integer (Stored Integer Value Decrement) (masked subsystem)		
Decrement Time To Zero (Decrement Time To Zero) (masked subsystem)		
Decrement To Zero (Decrement To Zero) (masked subsystem)		
Increment Real World (Real World Value Increment) (masked subsystem)		
Increment Stored Integer (Stored Integer Value Increment) (masked subsystem)		

Mask Parameters

This section lists parameters that describe masked blocks. This table lists masking parameters, which correspond to **Mask Editor** dialog box parameters (see “Setting Mask Parameters” on page 10-149).

Mask Parameters

Parameter	Description/Prompt	Values
`Mask`	Turns mask on or off.	`{'on'} | 'off'`
`MaskCallbackString`	Mask parameter callbacks that are executed when the respective parameter is changed on the dialog. Set by the **Dialog callback** field on the **Parameters** pane of the **Mask editor** dialog box.	pipe-delimited string `{''}`
`MaskCallbacks`	Cell array version of `MaskCallbackString`.	cell array `{'[]'}`
`MaskDescription`	Block description. Set by the **Mask description** field on the **Documentation** pane of the **Mask editor** dialog box.	string `{''}`
`MaskDisplay`	Drawing commands for the block icon. Set by the **Drawing commands** field on the **Icon** pane of the **Mask editor** dialog box.	string `{''}`
`MaskEditorHandle`	For internal use.	
`MaskEnableString`	Option that determines whether a parameter is greyed out in the dialog. Set by the **Enable parameter** check box on the **Parameters** pane of the **Mask editor** dialog box.	pipe-delimited string `{''}`

Mask Parameters (Continued)

Parameter	Description/Prompt	Values
`MaskEnables`	Cell array version of `MaskEnableString`.	cell array of strings, each either `'on'` or `''off'` `{'[]'}`
`MaskHelp`	Block help. Set by the **Mask help** field on the **Documentation** pane of the **Mask editor** dialog box.	`string {''}`
`MaskIconFrame`	Set the visibility of the icon frame (Visible is `on`, Invisible is `off`). Set by the **Frame** option on the **Icon** pane of the **Mask editor** dialog box.	`{'on'} | 'off'`
`MaskIconOpaque`	Set the transparency of the icon (Opaque is `on`, Transparent is `off`). Set by the **Transparency** option on the **Icon** pane of the **Mask editor** dialog box.	`{'on'} | 'off'`
`MaskIconRotate`	Set the rotation of the icon (Rotates is `on`, Fixed is `off`). Set by the **Rotation** option on the **Icon** pane of the **Mask editor** dialog box.	`'on' | {'off'}`
`MaskIconUnits`	Set the units for the drawing commands. Set by the **Units** option on the **Icon** pane of the **Mask editor** dialog box.	`'pixel' | {'autoscale'} | 'normalized'`
`MaskInitialization`	Initialization commands. Set by the **Initialization commands** field on the **Initialization** pane of the **Mask editor** dialog box.	MATLAB command `{''}`

Mask Parameters (Continued)

Parameter	Description/Prompt	Values
`MaskNames`	Cell array of mask dialog parameter names. Set inside the **Variable** column in the **Parameters** pane of the **Mask editor** dialog box.	matrix `{'[]'}`
`MaskPrompts`	List of dialog parameter prompts (see below). Set inside the **Dialog parameters** area on the **Parameters** pane of the **Mask editor** dialog box.	cell array of strings `{'[]'}`
`MaskPromptString`	List of dialog parameter prompts (see below). Set inside the **Dialog parameters** area on the **Parameters** pane of the **Mask editor** dialog box.	string `{''}`
`MaskPropertyName String`	Pipe-delimited version of **MaskNames**.	string `{''}`
`MaskRunInitForIconRedraw`	For internal use.	
`MaskSelfModifiable`	Indicates that the block can modify itself. Set by the **Allow library block to modify its contents** check box on the **Initialization** pane of the **Mask editor** dialog box.	`'on' | {'off'}`
`MaskStyles`	Determines whether the dialog parameter is a check box, edit field, or pop-up list. Set by the **Type** column in the **Parameters** pane of the **Mask editor** dialog box.	cell array `{'[]'}`
`MaskStyleString`	Comma-separated version of **MaskStyles**.	string `{''}`

Mask Parameters (Continued)

Parameter	Description/Prompt	Values
MaskTabNameString	For internal use.	
MaskTabNames	For internal use.	
MaskToolTips Display	Determines which of the mask dialog parameters to display.	cell array of 'on' and 'off' {"}
MaskToolTipString	Comma-delimited version of MaskToolTipsDisplay.	string {''}
MaskTunableValues	Allows the changing of mask dialog values during simulation. Set by the **Tunable** column in the **Parameters** pane of the **Mask editor** dialog box.	cell array of strings {'[]'}
MaskTunableValue String	Delimited string version of MaskTunableValueString.	delimited string {''}
MaskType	Mask type. Set by the **Mask type** field on the **Documentation** pane of the **Mask editor** dialog box.	string {'Stateflow'}
MaskValues	Dialog parameter values.	cell array {'[]'}
MaskValueString	Delimited string version of MaskValues.	delimited string {''}
MaskVarAliases	Specify aliases for a block's mask parameters. The aliases must appear in the same order as the parameters appear in the block's MaskValues parameter.	cell array {'[]'}
MaskVarAliasString	For internal use.	

Mask Parameters (Continued)

Parameter	Description/Prompt	Values
`MaskVariables`	List of the dialog parameters' variables (see below). Set inside the **Dialog parameters** area on the **Parameters** pane of the **Mask editor** dialog box.	string `{''}`
`MaskVisibilities`	Specifies visibility of parameters.	matrix `{'[]'}`
`MaskVisibility String`	Cell array version of `MaskVisibilities`.	string `{''}`
`MaskWSVariables`	List of the variables defined in the mask workspace (read only).	matrix `{'[]'}`

Setting Mask Parameters

When you use the **Mask Editor** to create a dialog box parameter for a masked block, you provide this information:

- The prompt, which you enter in the **Prompt** field
- The variable that holds the parameter value, which you enter in the **Variable** field
- The type of field created, which you specify by selecting a **Control type**
- Whether the value entered in the field is to be evaluated or stored as a literal, which you specify by selecting an **Assignment** type

How Masked Parameters are Stored

The mask parameters, listed in the preceding table, store the values specified for the dialog box parameters in these ways:

- The **Prompt** field values for all dialog box parameters are stored in the `MaskPromptString` parameter as a string, with individual values separated by a vertical bar (`|`), as shown in this example:

```
"Slope:|Intercept:"
```

- The **Variable** field values for all dialog box parameters are stored in the `MaskVariables` parameter as a string, with individual assignments separated by a semicolon. A sequence number indicates the prompt that is associated with a variable. A special character preceding the sequence number indicates the **Assignment** type: @ indicates **Evaluate**, & indicates **Literal**.

  For example, "a=@1;b=&2;" indicates that the value entered in the first parameter field is assigned to variable `a` and is evaluated in MATLAB before assignment, and the value entered in the second field is assigned to variable `b` and is stored as a literal, which means that its value is the string entered in the dialog box.

- The **Control type** field values for all dialog box parameters are stored in the `MaskStyleString` parameter as a string, with individual values separated by a comma. The **Popup strings** values appear after the `popup` type, as shown in this example:

  ```
  "edit,checkbox,popup(red|blue|green)"
  ```

- The parameter values are stored in the `MaskValueString` mask parameter as a string, with individual values separated by a vertical bar. The order of the values is the same as the order in which the parameters appear on the dialog box. For example, these statements define values for the parameter field prompts and the values for those parameters:

  ```
  MaskPromptString     "Slope:|Intercept:"
  MaskValueString      "2|5"
  ```

11

Model File Format

This section describes the format of a Simulink model file.

Model File Contents

A model file is a structured ASCII file that contains keywords and parameter-value pairs that describe the model. The file describes model components in hierarchical order.

The structure of the model file is as follows.

```
Model {
  <Model Parameter Name> <Model Parameter Value>
  ...
  BlockDefaults {
    <Block Parameter Name> <Block Parameter Value>
    ...
  }
  AnnotationDefaults {
    <Annotation Parameter Name> <Annotation Parameter Value>
    ...
  }
  System {
    <System Parameter Name> <System Parameter Value>
    ...
    Block {
      <Block Parameter Name> <Block Parameter Value>
      ...
    }
    Line {
      <Line Parameter Name> <Line Parameter Value>
      ...
      Branch {
        <Branch Parameter Name> <Branch Parameter Value>
        ...
      }
    }
    Annotation {
      <Annotation Parameter Name> <Annotation Parameter Value>
      ...
    }
  }
}
```

The model file consists of sections that describe different model components:

- The `Model` section defines model parameters.
- The `BlockDefaults` section contains default settings for blocks in the model.
- The `AnnotationDefaults` section contains default settings for annotations in the model.
- The `System` section contains parameters that describe each system (including the top-level system and each subsystem) in the model. Each `System` section contains block, line, and annotation descriptions.

See Chapter 10, "Model and Block Parameters" for descriptions of model and block parameters.

Model Section

The `Model` section, located at the top of the model file, defines the values for model-level parameters. These parameters include the model name, the version of Simulink last used to modify the model, and simulation parameters.

BlockDefaults Section

The `BlockDefaults` section appears after the simulation parameters and defines the default values for block parameters within this model. These values can be overridden by individual block parameters, defined in the `Block` sections.

AnnotationDefaults Section

The `AnnotationDefaults` section appears after the `BlockDefaults` section. This section defines the default parameters for all annotations in the model. These parameter values cannot be modified using the `set_param` command.

System Section

The top-level system and each subsystem in the model are described in a separate `System` section. Each `System` section defines system-level parameters

and includes `Block`, `Line`, and `Annotation` sections for each block, line, and annotation in the system. Each `Line` that contains a branch point includes a `Branch` section that defines the branch line.

12

Embedded MATLAB Function Block Reference

An Embedded MATLAB Function block lets you compose a MATLAB language function in a Simulink model that generates embeddable code. In an Embedded MATLAB Function block, you create functions with a rich subset of the MATLAB language. Later, when you simulate the model or generate code for a target environment, a function in an Embedded MATLAB Function block generates efficient C code. This code meets the strict memory and data type requirements of embedded target environments. In this way, Embedded MATLAB Function blocks bring the power of MATLAB for the embedded environment into Simulink.

The following sections describe the supported MATLAB features in Embedded MATLAB functions:

Supported Variable Types in Embedded MATLAB Functions (p. 12-2)	Data types supported by Embedded MATLAB Function blocks.
Operators in Embedded MATLAB Functions (p. 12-4)	Operators supported by Embedded MATLAB Function blocks.
Embedded MATLAB Run-Time Function Library (p. 12-8)	Lists of run-time library functions that you can call in an Embedded MATLAB function.
Unsupported MATLAB Features and Limitations (p. 12-32)	Features not supported by Embedded MATLAB Function blocks.

Supported Variable Types in Embedded MATLAB Functions

Embedded MATLAB functions support a subset of MATLAB data types for input (argument) and output (return) variables, and variables local to the function. This section lists and describes the data types supported in Embedded MATLAB functions, along with any exceptions or deviations from MATLAB behavior.

Embedded MATLAB functions support scalar, vector, and matrix (two-dimensional) variables of the types represented by the following cast functions.

Type/Function	Description
`char`	Character array (string)
`complex`	Complex data. Cast function takes real and imaginary components. See "Declaring Local Complex Variables Implicitly" in Using Simulink.
`double`	Double-precision floating point
`int8, int16, int32`	Signed integer
`logical`	Boolean `true` or `false`
`single`	Single-precision floating point
`uint8, uint16, uint32`	Unsigned integer

Note For more information on fixed-point support in Embedded MATLAB, refer to "Using the Fixed-Point Toolbox with Embedded MATLAB" in the Fixed-Point Toolbox User's Guide documentation.

For further information on Embedded MATLAB, see

- "Using the Embedded MATLAB Function Block" in the Simulink user's documentation
- Embedded MATLAB Function in the Simulink reference documentation

- “Using Embedded MATLAB Functions” in the Stateflow user’s documentation

Operators in Embedded MATLAB Functions

Embedded MATLAB functions support a large subset of MATLAB operators for data operands of type `double` and `single`. Allowed MATLAB operators in Embedded MATLAB functions are listed in the following topics:

- "Control Flow Statements in Embedded MATLAB Functions" on page 12-4 – Common programming statements supported in Embedded MATLAB functions
- "Arithmetic Operators in Embedded MATLAB Functions" on page 12-5 – Operators for arithmetic operations supported in Embedded MATLAB functions.
- "Relational Operators in Embedded MATLAB Functions" on page 12-6 – Operators for relational operations supported in Embedded MATLAB functions.
- "Logical Operators in Embedded MATLAB Functions" on page 12-6 – Lists operators for logical operations supported in Embedded MATLAB functions.

Each listing includes a link to the MATLAB Function Reference documentation (help) for the equivalent MATLAB function along with a one-line description and any limitations that apply.

Control Flow Statements in Embedded MATLAB Functions

Embedded MATLAB functions support the following MATLAB program statements:

Statement	Description
`break`	`break` statement
`continue`	`continue` statement
`for`	`for` statement

Statement	Description
`if`	`if` statement The conditions of an `if` statement cannot use `&` and `|` operators. In their place, use the `&&` and `||` operators, respectively. To logically collapse vectors into scalars, use the function `all`.
`return`	`return` statement
`switch`	`switch` statement
`while`	`while` statement The conditions of `while` statements cannot use `&` and `|` operators. In their place, use the `&&` and `||` operators, respectively. To logically collapse vectors into scalars, use the function `all`.

Arithmetic Operators in Embedded MATLAB Functions

Embedded MATLAB functions support the following MATLAB arithmetic operations for matrix operands (maximum two-dimensional) of type `double` or `single`. See "Arithmetic Operators + - * / \ ^ '" in the MATLAB Function Reference documentation for detailed descriptions of each operation and any requirements for the data types of the operands.

Operator	Description
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`.*`	Elementwise multiplication
`/`	Slash or matrix right division
`./`	Elementwise division
`\`	Backslash or matrix left division
`.\`	Array left division
`^`	Matrix power

Operator	Description
.^	Elementwise raising to a scalar power
[]	Concatenation of matrices
'	Complex conjugate transpose
.'	Transpose
(r, c)	Matrix indexing, where r and c are vectors of row and column indices, respectively

Relational Operators in Embedded MATLAB Functions

Embedded MATLAB functions support the following elementwise relational operators between matrix operands (maximum two-dimensional) of type `double` or `single` of the same size. See "Relational Operators < > <= >= == ~=" in the MATLAB Function Reference documentation for detailed descriptions of each operation and any requirements for the data types of the operands.

Operation	Description
<	Less than
<=	Less than or equal to
>=	Greater than or equal to
>	Greater than
==	Equal
~=	Not equal

Logical Operators in Embedded MATLAB Functions

Embedded MATLAB functions support the following elementwise logical operators between matrix operands (maximum two-dimensional) of type `double` or `single` of the same size. See "Logical Operators, Element-wise & | ~" and "Logical Operators, Short-circuit && ||" in the MATLAB Function Reference documenation for detailed descriptions of each operation and any requirements for the data types of the operands.

Operation	Description
`&`	Logical AND This `&` operator is limited to use outside `if` and `while` statement conditions. In its place, use the `&&` operator. To logically collapse vectors into scalars, use the Embedded MATLAB function `all`.
`|`	Logical OR This `|` operator is limited to use outside `if` and `while` statements. In its place, use the `||` operator. To logically collapse vectors into scalars use, the Embedded MATLAB function `all`.
`-`	Element complement
`xor`	Logical XOR
`&&`	Logical AND (short-circuiting)
`||`	Logical OR (short-circuiting)

Embedded MATLAB Run-Time Function Library

This section lists the callable functions of the Embedded MATLAB run-time function library. Functions in the Embedded MATLAB run-time library have the same name, same arguments (with limitations), and the same functionality (with limitations) as their MATLAB or Fixed-Point Toolbox counterparts, but are written in C to produce efficient embeddable generated code. If you restrict your function calls in Embedded MATLAB functions to these functions, you can use the code generated for an Embedded MATLAB function to build a stand-alone executable target that you can move to different platforms for execution.

For more information on fixed-point support in Embedded MATLAB, refer to "Using the Fixed-Point Toolbox with Embedded MATLAB" in the Fixed-Point Toolbox documentation.

The following topics list and describe the functions supported by the Embedded MATLAB run-time library:

- "Embedded MATLAB Run-Time Function Library – Alphabetical List" on page 12-8
- "Embedded MATLAB Run-Time Library – Categorical List" on page 12-20

Embedded MATLAB Run-Time Function Library – Alphabetical List

This topic lists functions in the Embedded MATLAB run-time library in alphabetical order. Each entry includes a link to the help for the equivalent MATLAB or Fixed-Point Toolbox function along with any limitations that apply to it. See also "Embedded MATLAB Run-Time Library – Categorical List" on page 12-20.

Function	Product	Remarks/Limitations
`abs`	MATLAB	—
`abs`	Fixed-Point Toolbox	—

Function	Product	Remarks/Limitations
`acos`	MATLAB	• Returns `NaN` when the input value `x` is real, but the output should be complex. To get the complex result, make the input value complex by passing in `complex(x)`.
`acosh`	MATLAB	• Returns `NaN` when the input value `x` is real, but the output should be complex. To get the complex result, make the input value complex by passing in `complex(x)`.
`all`	MATLAB	—
`all`	Fixed-Point Toolbox	—
`and`	MATLAB	—
`angle`	MATLAB	—
`any`	MATLAB	—
`any`	Fixed-Point Toolbox	—
`asin`	MATLAB	• Returns `NaN` when the input value `x` is real, but the output should be complex. To get the complex result, make the input value complex by passing in `complex(x)`.
`asinh`	MATLAB	—
`atan`	MATLAB	—
`atan2`	MATLAB	—
`atanh`	MATLAB	• Returns `NaN` when the input value `x` is real, but the output should be complex. To get the complex result, make the input value complex by passing in `complex(x)`.
`bitand`	MATLAB	• Does not support floating point inputs. The arguments must belong to an integer class.
`bitcmp`	MATLAB	• Does not support floating point input for the first argument. The first argument must belong to an integer class.

Function	Product	Remarks/Limitations
`bitget`	MATLAB	• Does not support floating point input for the first argument. The first argument must belong to an integer class.
`bitor`	MATLAB	• Does not support floating point inputs. The arguments must belong to an integer class.
`bitset`	MATLAB	• Does not support floating point input for the first argument. The first argument must belong to an integer class.
`bitshift`	MATLAB	• Does not support floating point input for the first argument. The first argument must belong to an integer class.
`bitxor`	MATLAB	• Does not support floating point inputs. The arguments must belong to an integer class.
`cast`	MATLAB	
`ceil`	MATLAB	—
`char`	MATLAB	—
`chol`	MATLAB	• Does not allow two output arguments
`class`	MATLAB	—
`complex`	MATLAB	—
`complex`	Fixed-Point Toolbox	—
`conj`	MATLAB	—
`conj`	Fixed-Point Toolbox	—
`conv`	MATLAB	—
`cos`	MATLAB	—
`cosh`	MATLAB	—
`cross`	MATLAB	• If supplied, `dim` must be a constant
`ctranspose`	MATLAB	—

Function	Product	Remarks/Limitations
`ctranspose`	Fixed-Point Toolbox	—
`cumprod`	MATLAB	—
`cumsum`	MATLAB	—
`diag`	MATLAB	• If supplied, k must be a real and scalar integer value
`diff`	MATLAB	• If applied, `n` and `dim` must be constants
`disp`	Fixed-Point Toolbox	—
`divide`	Fixed-Point Toolbox	• Any non-`fi` input must be constant; that is, its value must be known at compile time so that it can be cast to a `fi` • Complex and imaginary divisors are not supported
`double`	MATLAB	—
`double`	Fixed-Point Toolbox	—
`end`	Fixed-Point Toolbox	—
`eps`	MATLAB	
`eps`	Fixed-Point Toolbox	—
`eq`	MATLAB	—
`eq`	Fixed-Point Toolbox	• Not supported for fixed-point signals with different biases
`exp`	MATLAB	—
`eye`	MATLAB	• Dimensions must be real, non-negative, integer constants
`factorial`	MATLAB	—
`false`	MATLAB	
`fft`	MATLAB	• Length of input vector must be a power of 2
`fftshift`	MATLAB	—

Function	Product	Remarks/Limitations
`fi`	Fixed-Point Toolbox	• Use to create a fixed-point constant or variable in Embedded MATLAB • The syntax `fi('PropertyName',PropertyValue...)` is not supported. To use property name/property value pairs, you must first specify the value `v` of the `fi` object as in `fi(v,'PropertyName',PropertyValue...)` • Works for constant input values only; that is, the value of the input must be known at compile time • `Numerictype` information must be available for nonfixed-point Simulink inputs
`filter`	MATLAB	• Results might differ from MATLAB if the input contains NaNs
`fimath`	Fixed-Point Toolbox	• Fixed-point signals coming in to an Embedded MATLAB Function block from Simulink are assigned the default `fimath` object • fimath objects in Embedded MATLAB
`fix`	MATLAB	—
`floor`	MATLAB	—
`ge`	MATLAB	—
`ge`	Fixed-Point Toolbox	• Not supported for fixed-point signals with different biases
`gt`	MATLAB	—
`gt`	Fixed-Point Toolbox	• Not supported for fixed-point signals with different biases
`horzcat`	Fixed-Point Toolbox	—

Function	Product	Remarks/Limitations
`ifft`	MATLAB	• Length of input vector must be a power of 2 • Output of `ifft` block is always complex
`imag`	MATLAB	—
`imag`	Fixed-Point Toolbox	—
`int8`, `int16`, `int32`	MATLAB	—
`int8`, `int16`, `int32`	Fixed-Point Toolbox	—
`Inf` or `inf`	MATLAB	• Dimensions must be real, non-negative, integer constants • Supports only one or two dimension arguments
`interp1`	MATLAB	• Supports only `linear` and `nearest` interpolation methods • Does not handle evenly spaced `X` indices separately • `X` must be strictly monotonically increasing or strictly monotonically decreasing; does not reorder indices
`interp1q`, see `interp1`	MATLAB	• `X` must be strictly monotonically increasing or strictly monotonically decreasing; does not reorder indices
`intmax`	MATLAB	
`intmin`	MATLAB	
`inv`	MATLAB	—
`isa`	MATLAB	—
`ischar`	MATLAB	—
`iscolumn`	Fixed-Point Toolbox	—
`isempty`	MATLAB	—

Function	Product	Remarks/Limitations
isempty	Fixed-Point Toolbox	—
isfi	Fixed-Point Toolbox	—
isfimath	Fixed-Point Toolbox	—
isfinite	MATLAB	—
isfinite	Fixed-Point Toolbox	—
isfloat	MATLAB	—
isinf	MATLAB	—
isinf	Fixed-Point Toolbox	—
isnan	MATLAB	—
isnan	Fixed-Point Toolbox	—
isnumeric	MATLAB	—
isnumeric	Fixed-Point Toolbox	—
isnumerictype	Fixed-Point Toolbox	—
isreal	MATLAB	—
isreal	Fixed-Point Toolbox	—
isrow	Fixed-Point Toolbox	—
isscalar	MATLAB	—
isscalar	Fixed-Point Toolbox	—
issigned	Fixed-Point Toolbox	—
isvector	MATLAB	—
isvector	Fixed-Point Toolbox	—
ldivide	MATLAB	—
le	MATLAB	—
le	Fixed-Point Toolbox	• Not supported for fixed-point signals with different biases
length	MATLAB	—
length	Fixed-Point Toolbox	—

Function	Product	Remarks/Limitations
`log`	MATLAB	• Returns `NaN` when the input value `x` is real, but the output should be complex. To get the complex result, make the input value complex by passing in `complex(x)`.
`logical`	MATLAB	—
`logical`	Fixed-Point Toolbox	—
`lowerbound`	Fixed-Point Toolbox	—
`lsb`	Fixed-Point Toolbox	—
`lt`	MATLAB	—
`lt`	Fixed-Point Toolbox	• Not supported for fixed-point signals with different biases
`lu`	MATLAB	—
`max`	MATLAB	—
`max`	Fixed-Point Toolbox	• Supported for 1-D and 2-D arrays only
`mean`	MATLAB	—
`min`	MATLAB	—
`min`	Fixed-Point Toolbox	• Supported for 1-D and 2-D arrays only
`minus`	MATLAB	—
`minus`	Fixed-Point Toolbox	• Any non-`fi` input must be constant; that is, its value must be known at compile time so that it can be cast to a `fi`
`mldivide`	MATLAB	—
`mod`	MATLAB	—
`mpower`	MATLAB	• Does not support fractional powers of matrices
`mrdivide`	MATLAB	—

Function	Product	Remarks/Limitations
`mtimes`	MATLAB	—
`mtimes`	Fixed-Point Toolbox	• Any non-`fi` input must be constant; that is, its value must be known at compile time so that it can be cast to a `fi`
NaN or `nan`	MATLAB	• Dimensions must be real, non-negative, integer constants • Supports only one or two dimension arguments
`nargin`	MATLAB	—
`nargout`	MATLAB	—
`ndims`	MATLAB	—
`ndims`	Fixed-Point Toolbox	—
`ne`	MATLAB	—
`ne`	Fixed-Point Toolbox	• Not supported for fixed-point signals with different biases
`norm`	MATLAB	• If `NaN` is present in `X`, the result is `NaN`.
`not`	MATLAB	—
`numberofelements`	Fixed-Point Toolbox	• `numberofelements` and `numel` both work the same as MATLAB `numel` for `fi` objects in Embedded MATLAB

Function	Product	Remarks/Limitations
`numerictype`	Fixed-Point Toolbox	• Fixed-point signals coming in to an Embedded MATLAB Function block from Simulink are assigned a `numerictype` object that is populated with the signal's data type and scaling information • Returns the data type when the input is a nonfixed-point signal
`ones`	MATLAB	—
`or`	MATLAB	—
`pi`	MATLAB	—
`pinv`	MATLAB	—
`plus`	MATLAB	—
`plus`	Fixed-Point Toolbox	• Any non-`fi` input must be constant; that is, its value must be known at compile time so that it can be cast to a `fi`
`pow2`	Fixed-Point Toolbox	—
`power`	MATLAB	• Returns `NaN` when both `X` and `Y` are real, but `power(X,Y)` is complex. To get the complex result, make the input value `X` complex by passing in `complex(X)`. For example, `power(complex(X),Y)`. • Returns `NaN` when both `X` and `Y` are real, but `X .^ Y` is complex. To get the complex result, make the input value `X` complex by using `complex(X)`. For example, `complex(X).^Y`.
`prod`	MATLAB	—
`range`	Fixed-Point Toolbox	—
`rdivide`	MATLAB	—
`real`	MATLAB	—
`real`	Fixed-Point Toolbox	—
`realmax`	MATLAB	—
`realmax`	Fixed-Point Toolbox	—

Function	Product	Remarks/Limitations
realmin	MATLAB	—
realmin	Fixed-Point Toolbox	—
rem	MATLAB	—
repmat	MATLAB	
repmat	Fixed-Point Toolbox	—
reshape	MATLAB	• Accepts a maximum of three arguments
reshape	Fixed-Point Toolbox	• Supported for 1-D and 2-D arrays only
round	MATLAB	—
sign	MATLAB	—
sign	Fixed-Point Toolbox	—
sin	MATLAB	—
single	MATLAB	—
single	Fixed-Point Toolbox	—
sinh	MATLAB	—
size	MATLAB	—
size	Fixed-Point Toolbox	—
sort	MATLAB	—
sosfilt	Signal Processing Toolbox	—
sqrt	MATLAB	• Returns NaN when the input value x is real, but the output should be complex. To get the complex result, make the input value complex by passing in complex(x).
strcmp	MATLAB	—
subsasgn	Fixed-Point Toolbox	—
subsref	Fixed-Point Toolbox	—
sum	MATLAB	—

Function	Product	Remarks/Limitations
`sum`	Fixed-Point Toolbox	• Supported for 1-D and 2-D arrays only
`svd`	MATLAB	—
`tan`	MATLAB	—
`tanh`	MATLAB	—
`times`	MATLAB	—
`times`	Fixed-Point Toolbox	• Any non-`fi` input must be constant; that is, its value must be known at compile time so that it can be cast to a `fi`
`tril`	MATLAB	
`triu`	MATLAB	
`transpose`	MATLAB	—
`transpose`	Fixed-Point Toolbox	—
`true`	MATLAB	
`uint8`, `uint16`, `uint32`	MATLAB	—
`uint8`, `uint16`, `uint32`	Fixed-Point Toolbox	—
`uminus`	MATLAB	—
`uminus`	Fixed-Point Toolbox	—
`uplus`	MATLAB	—
`uplus`	Fixed-Point Toolbox	—
`upperbound`	Fixed-Point Toolbox	—
`vertcat`	Fixed-Point Toolbox	—
`xcorr`	Signal Processing Toolbox	• Does not support the case where A is a matrix • Does not support partial (abbreviated) strings of biased, unbiased, coeff, or none
`zeros`	MATLAB	—

Embedded MATLAB Run-Time Library – Categorical List

The following topics list functions in the Embedded MATLAB run-time library by different function types. Each entry includes a function name link to online help for the equivalent MATLAB of Fixed-Point Toolbox function along with a one-line description.

- "Arithmetic Operator Functions" on page 12-20
- "Casting Functions" on page 12-21
- "Complex Number Functions" on page 12-22
- "Exponential Functions" on page 12-22
- "Fixed-Point Toolbox Functions" on page 12-23
- "Input and Output Functions" on page 12-25
- "Logical Operator Functions" on page 12-26
- "Matrix/Array Functions" on page 12-26
- "Relational Operator Functions" on page 12-28
- "Rounding and Remainder Functions" on page 12-28
- "Signal Processing Functions" on page 12-28
- "Special Values" on page 12-29
- "String Functions" on page 12-30
- "Trigonometric Functions" on page 12-30

For an alphabetical list of these functions, and remarks and limitations for them, see "Embedded MATLAB Run-Time Function Library – Alphabetical List" on page 12-8.

Arithmetic Operator Functions

See "Arithmetic Operators + - * / \ ^ '" in the MATLAB Function Reference documentation for detailed descriptions on any of the following operator equivalent functions.

Function	Description
`ctranspose`	Complex conjugate transpose (`'`)
`isa`	Determine if input is object of given class
`ldivide`	Arraywise left division
`minus`	Minus (`-`)
`mldivide`	Left matrix divide (`\`)
`mpower`	Equivalent of elementwise power operator (`.^`)
`mrdivide`	Matrix right division
`mtimes`	Matrix multiply (`*`)
`plus`	Plus (`+`)
`power`	Arraywise power
`rdivide`	Arraywise right division
`times`	Arraywise multiplication
`transpose`	Matrix transpose (`'`)
`uminus`	Unary minus (`-`)
`uplus`	Unary plus (`+`)

Casting Functions

Embedded MATLAB functions support the following functions for converting one type of data to another:

Data Type	Description
`cast`	Cast variable to different data type
`char`	Create character array (string)
`class`	Query class of object argument
`double`	Convert to double-precision floating point
`int8, int16, int32`	Convert to signed integer data type
`logical`	Convert to Boolean `true` or `false` data type

Data Type	Description
`single`	Convert to single-precision floating point
`uint8, uint16, uint32`	Convert to unsigned integer data type

Complex Number Functions

Embedded MATLAB functions support the following functions for complex numbers:

Function	Description
`complex`	Construct complex data from real and imaginary components; see "Declaring Local Complex Variables Implicitly" in Using Simulink
`conj`	Return the conjugate of a complex number
`imag`	Return the imaginary part of a complex number
`isnumeric`	True for numeric arrays
`isreal`	Return false (0) for a complex number
`isscalar`	True if array is a scalar
`real`	Return the real part of a complex number

Exponential Functions

Embedded MATLAB functions support the following elementary exponential functions:

Function	Description
`exp`	Exponential
`factorial`	Factorial function
`log`	Natural logarithm
`sqrt`	Square root

Fixed-Point Toolbox Functions

For more information on fixed-point support in Embedded MATLAB, see "Using the Fixed-Point Toolbox with Embedded MATLAB" in the Fixed-Point Toolbox documentation. Embedded MATLAB supports the following functions from the Fixed-Point Toolbox:

Function	Description
`abs`	Return the absolute value of a `fi` object
`all`	Determine if all array elements are nonzero
`any`	Determine if any array elements are nonzero
`complex`	Construct a complex `fi` object from real and imaginary parts
`conj`	Return the complex conjugate of a `fi` object
`ctranspose`	Return the complex conjugate transpose of a `fi` object
`disp`	Display an object
`divide`	Divide two objects using a `numerictype` object
`double`	Return the double-precision floating-point real-world value of a `fi` object
`end`	Indicate last index of array
`eps`	Return the quantized relative accuracy for `fi` objects or `quantizer` objects
`eq`	Determine whether the real-world values of two `fi` objects are equal
`fi`	Construct a `fi` object
`fimath`	Construct a `fimath` object
`ge`	Determine whether the real-world value of one `fi` object is greater than or equal to another
`gt`	Determine whether the real-world value of one `fi` object is greater than another
`horzcat`	Horizontally concatenate two or more `fi` objects
`imag`	Return the imaginary part
`int8`, `int16`, or `int32`	Return the stored integer value of a `fi` object as a built-in `int8`, `int16`, or `int32`
`iscolumn`	Determine whether a `fi` object is a column vector

Function	Description
isempty	Determine if array is empty
isfi	Determine whether a variable is a fi object
isfimath	Determine whether a variable is a fimath object
isfinite	Determine if array elements are finite
isinf	Determine if array elements are infinite
isnan	Determine if array elements are NaN
isnumeric	Determine if input is numeric array
isnumerictype	Determine whether a variable is a numerictype object
isreal	Determine if array elements are real
isrow	Determine whether a fi object is a row vector
isscalar	Determine if input is scalar
issigned	Determine whether a fi object is signed
isvector	Determine if input is vector
le	Determine whether the real-world value of a fi object is less than or equal to another
length	Return the length of a vector
logical	Convert numeric values to logical
lowerbound	Return lower bound of range of fi object
lsb	Return the scaling of the least significant bit of a fi object
lt	Determine whether the real-world value of a fi object is less than another
max	Return largest element in array of fi objects
min	Return smallest element in array of fi objects
minus	Return the matrix difference between fi objects
mtimes	Return the matrix product of fi objects
ndims	Return number of array dimensions
ne	Determine whether the real-world values of two fi objects are not equal
numberofelements	Return number of data elements in fi array

Function	Description
numerictype	Construct a numerictype object
plus	Return the matrix sum of fi objects
pow2	Multiply by a power of 2
range	Return the numerical range of a fi object or quantizer object
real	Return real part of complex number
realmax	Return the largest positive fixed-point value or quantized number
realmin	Return the smallest positive normalized fixed-point value or quantized number
repmat	Replicate and tile an array
reshape	Reshape array
sign	Perform signum function on array
single	Return the single-precision floating-point real-world value of a fi object
size	Return array dimensions
subsasgn	Subscripted assignment
subsref	Subscripted reference
sum	Return sum of array elements
times	Return the result of element-by-element multiplication of fi objects
transpose	Return the transpose
uint8, uint16, or uint32	Return the stored integer value of a fi object as a built-in uint8, uint16, or uint32
uminus	Negate the elements of a fi object array
uplus	Unary plus
upperbound	Return upper bound of range of fi object
vertcat	Vertically concatenate two or more fi objects

Input and Output Functions

Embedded MATLAB functions support the following functions for accessing argument and return values:

Function	Description
nargin	Return the number of input arguments a user has supplied
nargout	Return the number of output return values a user has requested

Logical Operator Functions

Embedded MATLAB functions support the following functions for performing logical operations:

Function	Description
and	Logical AND (&)
not	Logical NOT (~)
or	Logical OR (|)

Matrix/Array Functions

Embedded MATLAB functions support the following functions:

Function	Description
abs	Return absolute value and complex magnitude of an array
all	Test if all elements are nonzero
angle	Phase angle
any	Test for any nonzero elements
cross	Vector cross product
cumprod	Cumulative product of array elements
cumsum	Cumulative sum of array elements
diag	Return a matrix formed around the specified diagonal vector and the specified diagonal (0, 1, 2,...) it occupies
diff	Differences and approximate derivatives
eye	Identity matrix
false	Return an array of 0's for the specified dimensions

Function	Description
`interp1`	One-dimensional interpolation (table lookup)
`interp1q`, see `interp1`	One-dimensional data interpolation (table lookup)
`inv`	Inverse of a square matrix
`isempty`	Test if an array is empty
`isfloat`	Determine if input is floating-point array
`isfinite`	Detect finite elements of an array
`isinf`	Detect infinite elements of an array (simulation only)
`isnan`	Detect NaN elements of an array (simulation only)
`length`	Return the length of a matrix
`lu`	Matrix factorization
`max`	Maximum elements of a matrix
`min`	Minimum elements of a matrix
`ndims`	Number of dimensions
`ones`	Create a matrix of all 1s
`pinv`	Pseudoinverse of a matrix
`prod`	Product of array element
`repmat`	Replicate and tile an array
`reshape`	Reshape one array into the dimensions of another
`sign`	Signum function
`size`	Return the size of a matrix
`sort`	Sort elements in ascending or descending order
`sum`	Sum of matrix elements
`tril`	Extract lower triangular part
`triu`	Extract upper triangular part
`true`	Return an array of logical (Boolean) 1s for the specified dimensions
`zeros`	Create a matrix of all zeros

Relational Operator Functions

Embedded MATLAB functions support the following functions for performing relational operations:

Function	Description
`eq`	Equal (==)
`ge`	Greater than or equal to (>=)
`gt`	Greater than (>)
`le`	Less than or equal to (<=)
`lt`	Less than (<)
`ne`	Not equal (~=)

Rounding and Remainder Functions

Embedded MATLAB functions support the following rounding and remainder functions:

Function	Description
`ceil`	Round toward plus infinity
`fix`	Round toward zero
`floor`	Round toward minus infinity
`mean`	Average or mean value
`mod`	Modulus (signed remainder after division)
`rem`	Remainder after division
`round`	Round toward nearest integer

Signal Processing Functions

Embedded MATLAB supports the following signal processing functions:

Function	Description
`chol`	Cholesky factorization
`conv`	Convolution and polynomial multiplication (license required – see the following note)
`fft`	Discrete Fourier transform (license required – see the following note)
`filter`	Filter a data sequence using a digital filter that works for both real and complex inputs (license required – see the following note)
`ifft`	Inverse discrete Fourier transform (license required – see the following note)
`sosfilt`	Second order (biquadratic) IIR filtering (license required – see the following note)
`svd`	Singular value decomposition
`xcorr`	Cross-correlation function estimates (license required – see the following note)

Note Some signal processing functions for the Embedded MATLAB Run-Time Function library require a Signal Processing Blockset license. These functions also have file dependencies on static functions supplied for some of the Signal Processing blocks. For code deployment to another environment, be sure to consult the generated make file for the Real-Time Workshop target you build to identify and locate these file dependencies.

Special Values

Embedded MATLAB functions support the following special data values:

Symbol	Description
`eps`	Return floating-point relative accuracy
Inf or `inf`	Return IEEE arithmetic representation for positive infinity (simulation only)
`intmax`	Largest possible value of specified integer type
`intmin`	Smallest possible value of specified integer type

Symbol	Description
NaN or nan	Return not a number (simulation only)
pi	Return the ratio of the circumference to the diameter for a circle
realmax	Return the largest positive floating-point number
realmin	Return the smallest positive floating-point number

String Functions

Embedded MATLAB functions support the following functions for handling strings:

Function	Description
char	Create character array (string)
ischar	True for character array (string)
strcmp	Return a logical result for the comparison of two strings; limited to strings known at compile time

Trigonometric Functions

Embedded MATLAB functions support the following elementary trigonometric functions:

Function	Description
acos	Inverse cosine
acosh	Inverse hyperbolic cosine
asin	Inverse sine
asinh	Inverse hyperbolic sine
atan	Inverse tangent
atan2	Four quadrant inverse tangent
atanh	Inverse hyperbolic tangent
cos	Cosine
cosh	Hyperbolic cosine

Function	Description
`sin`	Sine
`sinh`	Hyperbolic sine
`tan`	Tangent
`tanh`	Hyperbolic tangent

Unsupported MATLAB Features and Limitations

Embedded MATLAB Function blocks provide a subset of the full list of MATLAB features. Unsupported features and features with limitations are described in the following topics:

- "List of Unsupported Features" on page 12-32
- "Limitations on Indexing Operations" on page 12-33
- "Limitations with Complex Numbers" on page 12-34

Note For information about fixed-point support in Embedded MATLAB and its limitations, see "Using the Fixed-Point Toolbox with Embedded MATLAB" in the Fixed-Point Toolbox documentation.

List of Unsupported Features

The following is a list of MATLAB features that are not supported by Embedded MATLAB functions.

Feature Not Supported	Remarks
Cell arrays	Supported data types are listed in "Supported Variable Types in Embedded MATLAB Functions" on page 12-2.
Command-line duality	—
Dynamic variables	You cannot use variables of dynamic size, or variables of different sizes.
Function handles	—
Global	—
Java	—
M-files	User M-files in the MATLAB path are not supported for code generation, but they can be called during simulation.

Feature Not Supported	Remarks
Matrix deletion	
N-dimensional matrices	Supported sizes are scalar and two-dimensional matrices. Vectors are two-dimensional matrices with a row or column dimension of 1.
Nested functions	—
Objects	—
Sparse matrices	—
Structures	—
Try/catch	—

Limitations on Indexing Operations

Matrix indexing operations for a matrix `M` are supported with limitations for the following types of expressions:

- `M(i:j)` where `i` and `j` change in a loop

  Embedded MATLAB never dynamically allocates memory for the size of the expressions that change as the program executes. The workaround is to use `for` loops as shown in the following example:

  ```
  for i=1:10
   for j = i:10
    M(i,j) = 2 * M(i,j);
   end
  end
  ```

- `M(i:i+k)` where `i` is unknown but `k` is known

  In this case, since `i` and therefore `i+k` are not known, memory cannot be allocated for the numerical result. However, memory can be allocated for the following workaround:

  ```
  M(i + (0:k))
  ```

In this case, an unkown scalar value `i` is added to each element of the known index vector `0...k`. This means that memory for `k+1` elements of `M` is allocated.

- Initialization of the following style:

```
for i = 1:10
 M(i) = 5;
end
```

In this case, the size of `M` changes as the loop is executed.

Limitations with Complex Numbers

Complex numbers and operations are supported in Embedded MATLAB Function blocks with the following exceptions:

- The first use of a variable that is later assigned a complex result must also be complex. For example,

```
X = 3;
 .
 .
 .
X = 4 + 5i;
```

fails because `X` is not defined as a complex variable by its first assignment. However,

```
X = 3 + 0i;
 .
 .
 .
X = 4 + 5i;
```

succeeds because `X` is defined as a complex variable in its first assignment.

- Even if the imaginary part is zero, if the result might be complex, Embedded MATLAB will treat it as complex. For example, although

```
X = ifft(fft(Y));
```

yields a real answer, Embedded MATLAB assumes that the function `ifft` might return a complex result. The workaround is to use the `real` function:

```
X = real(ifft(fft(Y)));
```

Index

A

Abs block 2-8
absolute tolerance
  simset parameter 5-16
  specifying for a block state 2-326
absolute value
  generating 2-8
Accumulator Resettable block 2-9
Accumulator Resettable Limited block 2-9
Action Port block 2-10
Action subsystems
  creating 2-10
  with If block 2-294
  with SwitchCase block 2-555
add_block command 4-6
add_line command 4-8
add_param command 4-10
Additional Discrete block library
  block parameters 10-140
Additional Math: Increment - Decrement block library
  block parameters 10-144
addterms command 4-11
Algebraic Constraint block 2-14
algebraic equations
  modeling 2-14
algebraic loops
  integrator block reset or IC port 2-205
analysis functions
  perturbing model 2-308
animate 7-7
AnnotationDefaults section of mdl file 11-3
annotations
  annotation block, *see* Model Info block
ashow debug command 7-8
Assert block 2-16
Assignment block 2-19
Atomic Subsystem block 2-540
atrace debug command 7-9
automatic scaling 8-24
  and Look-Up Table (2D) block 2-353
  autoscale safety margin 8-28
  fixptbestprec 8-7
autoscaling
  fixptbestprec 8-7
autoscaling Scope axes 2-481

B

Backlash block 2-27
Backward Euler method 2-202
Backward Rectangular method 2-202
Band-Limited White Noise block 2-32
bdclose command 4-13
bdroot command 4-14
bits
  clear 2-43
  mask 2-43
  set 2-43
block dialog boxes
  closing 4-15
  opening 4-43
block parameters
  Additional Discrete library 10-140
  Additional Math: Increment - Decrement library 10-144
  changing during simulation 4-51
  common 10-41
  Continuous library 10-51
  Discontinuities library 10-53
  Discrete library 10-56
  Logic and Bit Operations library 10-65
  Lookup Tables library 10-68
  Math library 10-74
  Model Verification block library 10-89
  Model-Wide Utilities library 10-93
  Ports & Subsystems library 10-95
  Signal Attributes library 10-114
  Signal Routing library 10-121
  Sinks library 10-128

Sources library 10-132
User-defiined functions library 10-139
`BlockDefaults` section of `mdl` file 11-3
blocks 10-51
Accumulator Resettable 2-9
Accumulator Resettable Limited 2-9
adding to model 4-6
Compare To Zero 2-96
Counter Limited 2-117
current 4-28
Data Type Propagation 2-141
Decrement Stored Integer 2-158
Decrement Time To Zero 2-159
Decrement To Zero 2-160
deleting
`delete_block` command 4-17
Detect Decrease 2-171
Detect Fall Negative 2-173
Detect Fall Nonpositive 2-174
Detect Increase 2-176
Detect Rise Nonnegative 2-177
Detect Rise Positive 2-179
Filter Direct Form II 2-580
Filter Direct Form II Time Varying 2-583
Filter First Order 2-585
Filter Lead or Lag 2-587
Filter Real Zero 2-589
handle of current 4-29
Increment Stored Integer 2-305
Index Vector 2-306
Interval Test Dynamic 2-335
Product of Elements Inverted 2-427
Repeating Sequence Stair 2-467
Sample Time Divide 2-656
Sample Time Multiply 2-657
Sample Time Probe 2-659
Sample Time Subtract 2-659
Sum of Elements Negated 2-550
Unit Delay Enabled External IC 2-615
Unit Delay Enabled Resettable 2-617
Unit Delay Enabled Resettable External IC 2-620
Unit Delay External IC 2-622
Unit Delay Resettable 2-624
Unit Delay Resettable External IC 2-626
Unit Delay With Preview Enabled 2-628
Unit Delay With Preview Enabled Resettable 2-631
Unit Delay With Preview Enabled Resettable External RV 2-634
Unit Delay With Preview Resettable 2-637
Unit Delay With Preview Resettable External RV 2-640
See also block parameters
`bode` function 3-7
Boolean expressions
modeling 2-90
`break` debug command 7-12
`bshow` debug command 7-14
Bus Creator block 2-49
Bus Selector block 2-54

C

capping unconnected blocks
using the Terminator block 2-564
character encoding, model 4-62
Check Discrete Gradient block 2-56
Check Dynamic Gap block 2-59
Check Dynamic Lower Bound block 2-62
Check Dynamic Range block 2-65
Check Dynamic Upper Bound block 2-68
Check Input Resolution block 2-71
Check Static Gap block 2-74
Check Static Lower Bound block 2-77
Check Static Range block 2-80
Check Static Upper Bound block 2-83
Chirp Signal block 2-86
`clear` debug command 7-15
clearing bits 2-43

Clock block 2-88
close_system command 4-15
clutch demo 2-290
code generation
  scaling 2-135
Combinatorial Logic block 2-90
combining input lines into vector line 2-404
Compare To Zero block 2-96
Complex to Magnitude-Angle block 2-98
Complex to Real-Imag block 2-100
concatenating matrices 2-381
Configurable Subsystem block 2-102
Constant block 2-108
constant value
  generating 2-108
continue debug command 7-16
Continuous block library
  block parameters 10-51
control flow diagrams
  Action subsystem 2-10
  do-while
    While Iterator block 2-660
  for
    For Iterator block 2-254
  if-else
    If block 2-294
  switch
    Switch Case block 2-555
  while
    While Iterator block 2-660
control flow statements in eML blocks 12-4
Coulomb and Viscous Friction block 2-113
Coulomb friction 2-113
Counter Limited block 2-117
Create Subsystem menu item 2-540
current block
  getting pathname 4-28
  handle 4-29
current system
  getting pathname 4-30

D

data object classes
  Simulink.AliasType 9-8
  Simulink.Bus 9-26
  Simulink.BusElement 9-29
  Simulink.ModelDataLogs 9-41
  Simulink.ModelWorkspace 9-45
  Simulink.NumericType 9-65
  Simulink.Parameter 9-72
  Simulink.ParamRTWInfo 9-75
  Simulink.Signal 9-89
  Simulink.StructElement 9-95
  Simulink.StructType 9-97
  Simulink.SubsysDataLogs 9-100
  Simulink.TimeInfo 9-102
  Simulink.TsArray 9-105
Data Store Memory block 2-119
Data Store Read block 2-125
Data Store Write block 2-128
Data Type Conversion block 2-130
Data Type Propagation block 2-141
data types
  propagation 2-141
Dead Zone block 2-153
deadband 2-27
debug commands
  ashow 7-8
  atrace 7-9
  break 7-12
  bshow 7-14
  clear 7-15
  continue 7-16
  disp 7-17
  emode 7-22
  etrace 7-23
  help 7-24
  ishow 7-40
  minor 7-25
  nanbreak 7-26
  next 7-27

probe 7-28
quit 7-29
run 7-31
states 7-34
status 7-35
step 7-36
stop 7-39
systems 7-41
tbreak 7-42
trace 7-43
undisp 7-44
untrace 7-45
xbreak 7-48
zcbreak 7-49
zclist 7-50
decimation factor 5-17
decision tables
modeling 2-90
Decrement Stored Integer block 2-158
Decrement Time To Zero block 2-159
Decrement To Zero block 2-160
delaying input by variable amount 2-643
delete_block command 4-17
delete_line command 4-18
delete_param command 4-19
demos
clutch 2-290
fohdemo 2-249
hardstop 2-290
lorenzs 2-669
Demux block 2-161
Derivative block 2-168
accuracy of 2-168
derivatives
calculating 2-168
limiting 2-443
Detect Decrease block 2-171
Detect Fall Negative block 2-173
Detect Fall Nonpositive block 2-174
Detect Increase block 2-176
Detect Rise Nonnegative block 2-177
Detect Rise Positive block 2-179
differential/algebraic systems
modeling 2-14
Digital Clock block 2-184
Discontinuities block library
block parameters 10-53
Discrete block library
block parameters 10-56
Discrete Filter block 2-195
Discrete State-Space block 2-198
discrete state-space model 3-6
Discrete Transfer Fcn block 2-217
Discrete Zero-Pole block 2-220
Discrete-Time Integrator block 2-201
discrete-time systems
linearization 3-6
disp command 6-4
disp debug command 7-17
Display block 2-223
as floating display 2-225
displaying
signals graphically 2-479
dlinmod function 3-3
DocBlock block 2-228
Dot Product block 2-229
dpoly command 6-5

E

eigenvalues of linearized matrix 3-7
Embedded MATLAB Function
signal processing functions 12-28
eML blocks
arithmetic operators 12-5
control flow statements 12-4
description 12-1
eML run-time library 12-8
logical operators 12-6
MATLAB features NOT supported 12-32

operators 12-4
relational operators 12-6
variable types 12-2
eML run-time library 12-8
alphabetical list of functions 12-8
casting functions 12-21
categorized list of functions 12-20
complex number functions 12-22
exponential functions 12-22
Fixed-Point Toolbox functions 12-23
input and output functions 12-25
logical operator functions 12-26
matrix/array functions 12-26
relational operator functions 12-28
rounding and remainder functions 12-28
special value functions 12-29
string functions 12-30
trigonometric functions 12-30
`emode` debug command 7-22
Enable block 2-237
Enabled and Triggered Subsystem block 2-239
Enabled Subsystem block 2-240
enabled subsystems
Enable block 2-237
`etrace` debug command 7-23
expressions
applying to block inputs 2-246
MATLAB Fcn block 2-378
external inputs
`flag` 5-8
from workspace 2-308
`ut` 5-12

F

Fcn block 2-246
compared to Math Function block 2-373
compared to Rounding Function block 2-473
compared to Trigonometric Function block 2-603
files
reading from 2-263
writing to
To File block 2-568
Filter Direct Form II block 2-580
Filter Direct Form II Time Varying block 2-583
Filter First Order block 2-585
Filter Lead or Lag block 2-587
Filter Real Zero block 2-589
`find_system` command 4-22
finding objects 4-22
Finite Impulse Response filter 2-195
finite-state machines
implementing 2-90
First-Order Hold block 2-249
compared to Zero-Order Hold block 2-249
fixdt function 8-3
fixed step size 5-17
Fixed-Point Interface Tool 8-24
fixpt_interp1 function 8-10
fixpt_look1_func_approx function 8-12
fixpt_look1_func_plot function 8-20
fixpt_set_all function 8-22
fixptbestexp function 8-4
fixptbestprec function 8-6
autoscaling 8-7
flip-flops
implementing 2-90
float function 8-23
floating scope
definition 2-489
Floating Scope block 2-479
`fohdemo` demo 2-249
`for` control flow diagram
creating 2-254
For Iterator block 2-254
For Iterator Subsystem block 2-259
For subsystems

creating 2-254
Forward Euler method 2-201
Forward Rectangular method 2-201
fprintf command 6-7
From block 2-260
From File block 2-263
From Workspace block 2-267
Function-Call Generator block 2-273
Function-Call Subsystem block 2-275
functions
fixdt 8-3
fixpt_interp1 8-10
fixpt_look1_func_approx 8-12
fixpt_look1_func_plot 8-20
fixpt_set_all 8-22
fixptbestexp 8-4
fixptbestprec 8-6
float 8-23
fxptdlg 8-24
num2fixpt 8-30
sfix 8-31
sfrac 8-32
sint 8-33
ufix 8-34
ufrac 8-35
uint 8-36
fxptdlg function 8-24

G

gain
varying during simulation 2-531
gcb command 4-28
gcbh command 4-29
gcs command 4-30
get_param command 4-35
global Goto tag visibility 2-282
Goto block 2-282
Goto Tag Visibility block 2-286
graphics
displaying on mask icon 6-11
Greek letters
displaying on mask icons 6-4
using the text function 6-15
Ground block 2-288
GUI 8-24
See also Fixed-Point Interface Tool

H

handle of current block 4-29
hardstop demo 2-290
help debug command 7-24
Hide Name menu item
suppressing display of port label 2-408
Hit Crossing block 2-290
hybrid systems
linearization 3-6

I

IC block 2-292
If Action Subsystem block 2-303
If block 2-294
if-else control flow diagram
creating 2-294
image
displaying on mask icon 6-8
drawing on mask icon using patch 6-10
image command 6-8
Increment Stored Integer block 2-305
Index Vector block 2-306
inf values
in mask plotting commands 6-11
Infinite Impulse Response filter 2-195
inherited
data types
by backpropagation 2-141
scaling
by backpropagation 2-141

initial conditions
setting 2-292
initial states 5-17
initial step size 5-18
Inport block 2-307
Inport blocks
in subsystem 2-540
linmod function 3-6
input ports
unconnected 2-288
inputs
applying expressions to 2-246
applying MATLAB function to
Fcn block 2-246
MATLAB Fcn block 2-378
combining into vector line 2-404
delaying by variable amount 2-643
external 5-12
from outside system 2-307
from previous time step 2-383
from workspace 2-308
generating step between two levels 2-536
interpolated mapping 2-358
logical operations on 2-339
multiplying block inputs during simulation 2-531
outputting minimum or maximum 2-391
passing through stair-step function 2-436
piecewise linear mapping of two 2-352
plotting 2-669
reading from file 2-263
width of 2-666
integration
block input 2-318
discrete-time 2-201
Integrator block 2-318
interpolated mapping 2-358
Interval Test Dynamic block 2-335
ishow debug command 7-40

J

Jacobians 3-6

L

left-hand approximation 2-201
limiting
signals 2-475
limiting derivative of signal 2-443
limiting integral 2-320
linear models
extracting
linmod function 3-6
linearization
discrete-time systems 3-6
linmod function 3-6
linearized matrix
eigenvalues 3-7
lines
adding 4-8
deleting 4-18
linmod function 3-3
Transport Delay block 2-592
linmod2 function 3-3
local Goto tag visibility 2-282
Logic and Bit Operations block library
block parameters 10-65
logic circuits
modeling 2-90
Logical Operator block 2-339
Look-Up Table (2-D) block 2-352
Look-Up Table (n-D) block 2-358
Lookup Tables block library
block parameters 10-68
lorenzs demo 2-669

M

MACs

propagating data type information for 2-145
Magnitude-Angle to Complex block 2-369
Manual Switch block 2-371
mask icon
displaying graphics on 6-11
displaying image on 6-8
displaying port label on 6-13
displaying symbols and Greek letters on 6-15
displaying text on 6-4
displaying text using `fprintf` 6-7
displaying text using `text` 6-15
displaying transfer function on 6-5
using the `patch` function 6-10
mask icons
displaying symbols and Greek letters on 6-4
question marks in 6-11
mask parameters
undefined 6-6
masked blocks
parameters 10-145
masked subsystems
question marks in icon 6-11
masking bits 2-43
Math block library
block parameters 10-74
Math Function block 2-372
mathematical functions
performing
Math Function block 2-372
Rounding Function block 2-473
Trigonometric Function block 2-603
mathematical symbols
displaying on mask icons 6-4
displaying on mask icons using `text` 6-15
MATLAB
features NOT supported in eML blocks 12-32
MATLAB character encoding, changing 4-62
MATLAB Fcn block 2-378
MATLAB functions
applying to block input
Fcn block 2-246
MATLAB Fcn block 2-378
matrices
concatenation 2-381
writing to 2-571
Matrix Concatenation block 2-381
maximum number of output rows 5-18
maximum order of `ode15s` solver 5-18
maximum step size
`simset` command 5-18
`mdl` file 11-2
Memory block 2-383
memory region
shared
Data Store Memory block 2-119
Data Store Read block 2-125
Data Store Write block 2-128
Merge block 2-386
MinMax block 2-391
`minor` debug command 7-25
model files 11-2
Model Info block 2-398
model parameters
table 10-2
Model Verification block library
block parameters 10-89
Model-Wide Utilities block library
block parameters 10-93
models
closing 4-13
creating
`new_system` command 4-41
getting name 4-14
parameters 10-2
replacing blocks 4-45
simulating 5-10

multiplying block inputs
during simulation 2-531
multirate systems
linearization 3-6
Mux block 2-404

N

Nan values
in mask plotting commands 6-11
nanbreak debug command 7-26
new_system command 4-41
next debug command 7-27
nonlinear systems
spectral analysis of 2-86
normally distributed random numbers 2-440
num2fixpt function 8-30

O

objects
finding 4-22
obsolete blocks, replacing 4-71
ode113 solver
Memory block 2-383
ode15s solver
maximum order property 5-18
Memory block 2-383
open_system command 4-43
opening
block dialog boxes 4-43
Simulink Library Browser 4-55
system windows 4-43
operating point 3-3
operators
arithmetic, in eML blocks 12-5
logical, in eML blocks 12-6
relational, in eML blocks 12-6
options structure
getting values 5-21
setting values 5-16
Outport blocks 2-407
in subsystem 2-540
linmod function 3-6
output
maximum rows 5-18
outside system 2-407
refine factor 5-18
selected elements of input vector 2-497
selected information about the signal on input 2-428
specifying points 5-18
switching between two inputs 2-371
values
displaying 2-223
variables 5-18
writing to file
To File block 2-568
writing to workspace
To Workspace block 2-571
zero within range 2-153
output ports
capping unconnected 2-564

P

parameters
adding 4-10
block
list 10-41
deleting 4-19
getting values of 4-35
masked blocks 10-145
model 10-2
setting values of
set_param command 4-51
patch command 6-10
phase-shifted wave 2-510
piecewise linear mapping
two inputs 2-352

piecewise linear signal
generating
Signal Builder block 2-510
plot command 6-11
Plot systems Interface 8-28
plotting input signals
Scope block 2-479
XY Graph block 2-669
plotting simulation data 5-14
port label
displaying on mask icon 6-13
port labels
suppressing display 2-408
port_label command 6-13
Ports & Subsystems block library
block parameters 10-95
precision
best 8-4
maximum 8-6
probe debug command 7-28
Product of Elements Inverted block 2-427
programmable logic arrays
modeling 2-90
propagation of data types 2-141
properties of Scope block 2-486
Pulse Generator block 2-431

Q

Quantizer block 2-436
question marks in mask icon 6-11
quit debug command 7-29

R

random noise
generating 2-440
Random Number block 2-440
and Band-Limited White Noise block 2-32
compared to Band-Limited White Noise block 2-440
random numbers
generating normally distributed 2-32
normally distributed 2-440
uniformly distributed 2-607
Rate Limiter block 2-443
Rate Transition block 2-447
reading data
from data store 2-125
from file 2-263
from workspace 2-267
Real-Imag to Complex block 2-452
refine factor
simset command 5-18
region of zero output 2-153
regular expressions 4-25
relative tolerance 5-19
Repeating Sequence block 2-462
Repeating Sequence Stair block 2-467
repeating signals 2-462
replace obsolete blocks 4-71
replace_block command 4-45
replacing blocks in model 4-45
Reshape block 2-470
right-hand approximation 2-202
Rounding Function block 2-473
run debug command 7-31

S

S-Function block 2-503
S-Function Builder block 2-505
Sample Time Divide block 2-656
Sample Time Multiply block 2-657
Sample Time Probe block 2-659
Sample Time Subtract block 2-659
sample-and-hold
applying to block input 2-383
sampling interval

generating simulation time 2-184
Saturation block 2-475
save_system command 4-47
sawtooth wave
generating 2-514
Scope axes
autoscaling 2-481
Scope block 2-479
properties 2-486
saving axes settings 2-485
scoped Goto tag visibility 2-282
Selector block 2-497
separating vector signal 2-161
sequence of signals 2-431
sequential circuits
implementing 2-92
set_param command 4-51
setting bits 2-43
setting parameter values 4-51
sfix function 8-31
sfrac function 8-32
shared data store
Data Store Memory block 2-119
Data Store Read block 2-125
Data Store Write block 2-128
Sign block 2-508
Signal Attributes block library
block parameters 10-114
Signal Generator block 2-514
Signal Inspection block 2-428
signal logging
enabling
simset command 5-20
signal logging name
specifying
simset command 5-20
signal processing functions
for Embedded MATLAB Function 12-28
Signal Routing block library
block parameters 10-121
Signal Specification block 2-517
Signal Viewer Scope 2-479
signals
displaying graphically 2-479
displaying vector 2-480
displaying X-Y plot of 2-669
generating pulses 2-431
limiting 2-475
limiting derivative of 2-443
passed from Goto block 2-260
passing to From block 2-282
plotting
Scope block 2-479
XY Graph block 2-669
repeating 2-462
sim command 5-10
simget command 5-21
simplot command
plotting simulation data 5-14
simset command 5-16
simulating models 5-10
simulation
parameters
specifying using simset
command 5-16
stopping
Stop Simulation block 2-539
simulation time
generating at sampling interval 2-184
outputting 2-88
simulink command 4-55
Simulink Library Browser
opening 4-55
Simulink.AliasType 9-8
Simulink.Bus 9-26
Simulink.BusElement 9-29
Simulink.ModelDataLogs 9-41
Simulink.ModelWorkspace 9-45
Simulink.NumericType 9-65
Simulink.Parameter 9-72

Simulink.ParamRTWInfo 9-75
Simulink.Signal 9-89
Simulink.StructElement 9-95
Simulink.StructType 9-97
Simulink.SubsysDataLogs 9-100
Simulink.TimeInfo 9-102
Simulink.TsArray 9-105
sine wave
  generating
    Signal Generator block 2-514
    Sine Wave block 2-524
  generating with increasing frequency
    Chirp Signal block 2-86
Sine Wave block 2-524
Sinks block library
  block parameters 10-128
sint function 8-33
Slider Gain block 2-531
slupdate command 4-71
solvers
  properties
    specifying 5-16
  specifying using simset command 5-19
Sources block library
  block parameters 10-132
spectral analysis of nonlinear systems 2-86
square wave
  generating 2-514
ss2tf function 3-8
ss2zp function 3-8
stair-step function
  passing signal through 2-436
state derivatives
  setting to zero 3-9
state space in discrete system 2-198
State-Space block 2-533
states
  initial 5-17
  outputting 5-18
  resetting 2-321
  saving at end of simulation 5-17
  specifying absolute tolerance for 2-326
states debug command 7-34
status debug command 7-35
Step block 2-536
step debug command 7-36
stop debug command 7-39
Stop Simulation block 2-539
stopping simulation 2-539
Subsystem block 2-540
subsystems
  and Inport blocks 2-307
  enabled 2-237
Sum block 2-546
Sum of Elements Negated block 2-550
Switch Case Action Subsystem block 2-560
switch control flow diagram
  creating 2-555
switching output between inputs
  Manual Switch block 2-371
switching output between two inputs 2-371
System section of mdl file 11-3
system windows
  closing 4-15
systems
  current 4-30
  saving 4-47
systems debug command 7-41

T

tbreak debug command 7-42
Terminator block 2-564
terminators
  adding 4-11
TeX formatting commands
  using in mask icon text 6-15
  using with disp 6-4
text command 6-15
tf2ss utility

converting Transfer Fcn to state-space form 2-576
time delay
simulating 2-591
Time-Based Linearization block 2-565
To File block 2-568
To Workspace block 2-571
trace debug command 7-43
tracing facilities 5-19
Transfer Fcn block 2-576
transfer function form
converting to 3-8
transfer functions
discrete 2-217
displaying on mask icon 6-5
linear 2-576
poles and zeros 2-673
discrete 2-220
Transport Delay block 2-591
Trapezoidal method 2-203
Trigger block 2-595
Trigger-Based Linearization block 2-600
Triggered Subsystem block 2-602
triggered subsystems
Trigger block 2-595
Trigonometric Function block 2-603
trim function 3-9
truth tables
implementing 2-90

U

ufix function 8-34
ufrac function 8-35
uint function 8-36
unconnected input ports 2-288
unconnected output ports
using the Terminator block 2-564
undisp debug command 7-44
Uniform Random Number block 2-607
compared to Band-Limited White Noise block 2-607
uniformly distributed random numbers 2-607
Unit Delay block
compared to Transport Delay block 2-591
Unit Delay Enabled External IC block 2-615
Unit Delay Enabled Resettable block 2-617
Unit Delay Enabled Resettable External IC block 2-620
Unit Delay External IC block 2-622
Unit Delay Resettable block 2-624
Unit Delay Resettable External IC block 2-626
Unit Delay With Preview Enabled block 2-628
Unit Delay With Preview Enabled Resettable block 2-631
Unit Delay With Preview Enabled Resettable External RV block 2-634
Unit Delay With Preview Resettable block 2-637
Unit Delay With Preview Resettable External RV block 2-640
untrace debug command 7-45
Update Diagram menu item
changing block parameters during simulation 4-51
User-defined functions block library
block parameters 10-139

V

variable time delay 2-643
Variable Time Delay block 2-643
Variable Transport Delay block 2-643
variable types of eML blocks 12-2
vdp model
Scope block 2-481
vector signals
displaying 2-480
generating from inputs 2-404
separating 2-161

viscous friction 2-113
visibility of Goto tag 2-286

W

`while` control flow diagram
  creating 2-660
While Iterator block 2-660
While Iterator Subsystem block 2-665
While subsystems
  creating 2-660
white noise
  generating 2-32
Width block 2-666
workspace
  destination 5-17
  reading data from 2-267
  source 5-19
  writing output to 2-571
writing data to data store 2-128
writing output to file 2-568
writing output to workspace 2-571

X

`xbreak` debug command 7-48
XY Graph block 2-669

Z

`zcbreak` debug command 7-49
`zclist` debug command 7-50
zero crossings
  detecting
    Hit Crossing block 2-290
    `simset` command 5-19
zero output in region
  generating 2-153
Zero-Order Hold block
  compared to First-Order Hold block 2-249
Zero-Pole block 2-673
zero-pole form
  converting to 3-8
zooming in on displayed data 2-482